# int_e_l®

# INTELLEC® 800
# MICROCOMPUTER DEVELOPMENT SYSTEM
# OPERATOR'S MANUAL

98-129A

PREFACE


This manual contains information on the use of the Intellec®
Microcomputer Development System.   The Intellec MDS is a design and
debugging tool for designers who are using the Intel® 8080 CPU or  Series
3000  computing  elements in OEM designs.   Through the use of MDS system
peripherals, optional plug-in modules, and the  software  provided,  the
system can be made to operate as a stand-alone computing system to check
out software techniques;  or as a partial system simulator which  checks
out a user´s hardware and software, while the software is executing from
RAM in the Intellec MDS;  with OEM hardware;  or as a chip simulator, to
verify  the  operation  of  the  user´s  system  after  the  software is
committed to ROM.


The following features of MDS software are discussed:

 • MDS MONITOR Program.  The MDS Monitor provides the basic
   utility functions for the Intellec MDS.  These functions
   include  program  loading,  memory  display,  memory
   modification,  PROM  programming,  program  checkout and
   debugging facilities, and a generalized  extensible  I/O
   system which is accessible by user programs.

 • TEXT  EDITOR  Program.   The  Text  Editor  provides
   facilities  for  creating  new  source  programs  or
   correcting existing ones.

 • 8080 ASSEMBLER Program.  The Assembler  program  accepts
   8080  source  language  statements  and generates a list
   file and object code file.


It is assumed that the reader of this  manual  has  prior  knowledge  of
computing systems and is familiar with 8080 programming.


Related Publications

 8080 Assembly Language Programming Manual
 Intel Publication # 98-004

Intellec MDS Hardware Reference Manual
Intel Publication # 98-132

Universal PROM Programmer Hardware Reference Manual
Intel Publication # 98-133

Intel High-Speed Paper Tape Reader
Instruction and Operation Guide
Intel Publication # 98-016

Reference manuals and specifications may be obtained by contacting:

Intel Corporation
Customer Services
3065 Bowers Avenue
Santa Clara, California  95051

# TABLE OF CONTENTS

# TABLES AND ILLUSTRATIONS

SECTION 1

GENERAL INFORMATION

## INTRODUCTION

The Intellec® Microcomputer Development System (MDS) is a complete design and debugging tool which allows the integration of both microcomputer hardware and software development. The system operates under control of an 8080 microcomputer which supervises all system resources such as main memory, I/O peripheral devices, Intellec bus facilities, and optional system facilities such as DMA (Direct Memory Access) and ICE (In-Circuit Emulator). The Intellec MDS is a self-contained modular microcomputer development system.

## SYSTEM DESCRIPTION

The standard Intellec® MDS consists of four microcomputer modules, an interconnecting printed circuit motherboard, power supplies, fans, a chassis, and a front panel. The four microcomputer modules mentioned above, consist of the 8080 CPU Module, 16K of RAM memory, Front Panel Control Module, and the Monitor Module. Modular expansion capability is provided by the additional 14 sockets on the motherboard.

## 8080 CPU Module

The CPU Module contains an Intel® 8080 CPU, an n-MOS 8-bit microprocessor. The CPU provides 2 us cycle time, 78 instructions, unlimited subroutine nesting, vectored interrupt, DMA capabilities, and a 16-line address bus which is associated with a bidirectional eight-line data bus.

Besides the 8080 itself, the CPU Module contains a real-time clock, CPU initialization logic, and logic to control the Intellec bus. It also implements a sophisticated priority interrupt mechanism. The CPU Module contains a priority encoder, which allows eight different levels of interrupts to be serviced. An interrupt mask register on the CPU Module allows the 8080 software to block interrupts at some levels, while allowing interrupts at other levels to be serviced. All signals

required by the 8080´s own interrupt system are generated by the CPU Module. A complete discussion of interrupt processing and programming requirements on the Intellec MDS is included in Appendix F.


## Monitor Module

The monitor module contains the Intellec® MDS Monitor and all MDS peripheral interface hardware. This module contains all necessary clocks, control, and data transfer circuitry to interface the following peripherals:

- Teletype

- CRT (RS232 Interface Specification)

- High Speed Paper Tape Reader

- High Speed Paper Tape Punch

- Line Printer

- Universal PROM Programmer

Use of the Monitor is discussed in Section 3.


## Front Panel Control Module

The Front Panel Control module contains circuits for controlling the front panel operations. It also provides much of the system overhead for bus control, clock generation, and the bootstrap program. A bus time-out system is included to prevent the CPU from halting operation if a nonexistent memory location or an incorrect I/O port is addressed.

The Intellec MDS front panel is intended to augment the console which is the primary user interactive device. Controls and indicators are kept to a minumum. These include eight interrupt initiation switches with corresponding indicators, RUN and HALT indicators, a BOOTSTRAP initialization switch, and a RESET switch. The use of these controls is discussed in detail below.


## Optional Modules

The basic Intellec MDS capabilities may be significantly enhanced by the addition of the following optional modules.

- ICE (In-Circuit Emulator)

- Customized User I/O Facilities with the installation of additional I/O modules

- Additional PROM or RAM memory modules to expand memory

in   increments of 6K (PROM) or 16K (RAM) 8-bit bytes, up
to a maximum of 12K of PROM and 64K of RAM.

- ●      DMA (Direct Memory Access)

## Software Support

The following software is provided for use in the Intellec  MDS.   These
software packages are described in detail in Sections 3, 4, and 5.

- ●      System Monitor
- ●      Text Editor
- ●      Macro Assembler

## MEMORY REQUIREMENTS

The Intellec®MDS memory is arranged into 65,536 addressable 8-bit bytes.
Figure  1-1 describes the memory in the form of a memory map.  The upper
2048 locations are reserved for a  2048  by  8-bit ROM containing  the
Intellec  MDS Monitor program.  The lower 256 bytes may be read from RAM
or a "Shadow PROM",  depending  on  the  setting  of  the  front  panel
BOOTSTRAP switch.  When the BOOTSTRAP switch is on, reading locations 00
through 255 retrieves the bytes from the "Shadow  PROM";   writing  into
these  same  locations  causes  the  data to be stored in RAM.  When the
BOOTSTRAP switch is off, reading and writing from locations  00  through
255  moves  data  to  and  from  RAM  only.  This 256 byte "Shadow PROM"
contains the BOOTSTRAP portion  of  the  Monitor  program.   Having  the
BOOTSTRAP   portion   of   the  Monitor  in  "Shadow  PROM"  allows  the
initialization code, invoked by the  front  panel  RESET  switch  to  be
executed  starting  at  location 00.  If a user program is run while the
system is in BOOTSTRAP mode, the results will be unpredictable.

User accessible RAM may be installed in locations 00 through  F7FFH,  in
increments   of  16,384  bytes.   Depending  on  the  amount  of  memory
available, it may be installed contiguously or in  segments.   The  only
restriction imposed with respect to memory installation is that at least
one block of memory be installed  in  the  low  order  memory  locations
(starting  at  location  00).   It should be noted that the user must be
aware of any addressing discontinuities and produce code accordingly.

The Intellec MDS Monitor reserves 331 bytes of  RAM  for  its  own  use.
These are:

- ●      Locations 0 through 7;
- ●      Locations 11 through 15 (0BH through 0FH);
- ●      The 318 locations at the top  of  the  first  contiguous
  block of RAM.

The location of the upper 318 bytes is determined when the system is
initialized during the BOOTSTRAP procedure.  The Monitor checks location
256 (100H) for a successful write/read operation, verifying the presence
of a block of RAM in locations 256 through 511 (100H through 1FFH).  If
the test is successful, it is repeated at address 512 (200H).  This
process is continued at memory address intervals of 256 (100H) until a
nonexistent memory block is found.  The reserved area is placed
immediately below the last contiguous memory location.  For example, in
a 16K system, the Monitor will use addresses 3EC0H through 3FFFH.

If memory is installed in noncontiguous segments, the 318 reserved
locations are placed immediately below the last contiguous location.
However, RAM may be installed in the locations above the missing
segment, providing usable memory above the reserved locations.
Normally, memory is installed from location 00, upward to the highest
available location without discontinuities.

When a full complement of memory is installed, the high order RAM which
would normally be accessible by addressing locations F800H through FFFFH
is shadowed by the Monitor ROM and cannot be accessed.  This restriction
also applies to any configuration of RAM which has portions in locations
F800H through FFFFH.

ADDRESS
FFFF

MONITOR

F800

USED BY
MONITOR                } 318 BYTES (decimal)

AVAILABLE RAM

Active area controlled
by front panel "BOOT" switch.

0100
00FF

"SHADOW"
(BOOTSTRAP
PROM)

0000
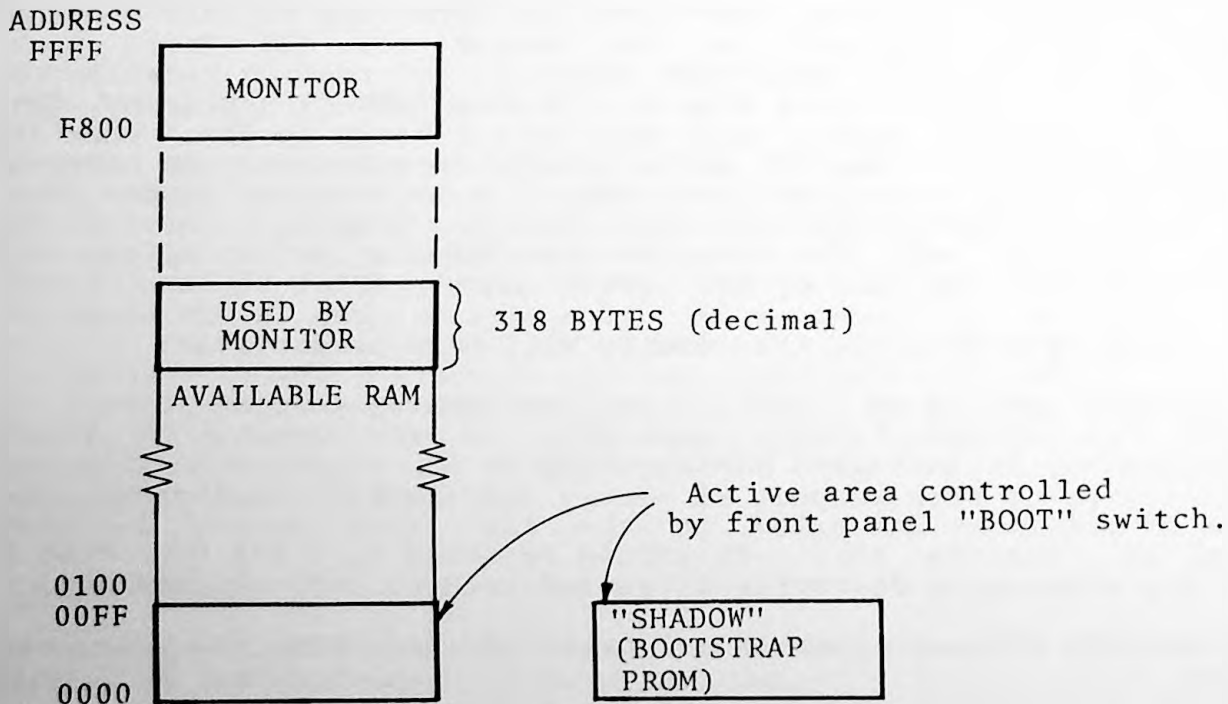
Figure 1-1:   Intellec® MDS Memory Map

## I/O SYSTEM CONFIGURATION

An Intellec MDS microcomputer system may be configured with a variety of peripheral devices ranging from a basic 10-character-per-second Teletype to higher speed devices such as a paper tape reader, punch, or CRT. The following logical devices are necessary as the minimum peripheral device complement to run the Intellec MDS Monitor. These devices may be assigned to various physical peripheral devices using the Monitor "Assign" Command, which is discussed in Section 3.

       CONSOLE  An interactive, character-oriented input/output device.

       READER   A character-oriented input stream device which transfers data upon command and signals the calling program when no more data is available (End Of File).

       PUNCH    A character-oriented output stream device which accepts a character from the calling program and records it on some extermal medium.

       LIST     A character-oriented output device which accepts a character from the calling program and records it on some external medium in human readable form.

Facilities are provided so that up to four distinct physical devices may be assigned to each logical device; the operator may select only one device at a time. Each logical device has a default physical device which is assigned at startup time (during the BOOTSTRAP procedure). However, the user has the option to specify any of the remaining three physical devices for that logical device.

SECTION 2

OPERATING THE INTELLEC© MDS

## INTRODUCTION

The Intellec MDS is designed with a simplified front panel which requires a minimum of button pushing. No data transfer facilities are incorporated into front panel operations. Except for cold-starting the MDS system, computer/operator dialogue is under control of the system Monitor, which provides facilities to operate an interactive console and other peripheral devices, and to accept operator commands. The system is at all times under control of the operator, through the Monitor.

The operator may select one of four console devices, one of four listing devices, one of four punching devices, and one of four paper tape reading devices. Once the system is configured, the operator may load a source 8080 program, edit it, and then assemble it into machine language object code. When the operation is completed, the operator may test the program by executing it in the MDS, then may output the code onto punched paper tape or may program a PROM.

The following section describes the operation of various peripherals, describes cold start and restart procedures, describes the procedures for loading and using the Assembler and Text Editor programs, and contains comprehensive instructions for the operation of the Monitor program which controls and supervises system operation.

## EQUIPMENT DESCRIPTION

The following paragraphs describe the operation of the Intellec MDS front panel and the various peripherals which are used during normal operation. This information is not intended to replace the operating procedures which accompany the equipment, but to serve as a convenient reference showing major operating controls and indicators. The discussion will include descriptions of the following:

        Intellec MDS Front Panel

Teletype Console

High Speed Paper Tape Reader

Universal PROM Programmer

## Intellec® MDS Front Panel Description

The Intellec MDS is pictured in Figure 2-1.  The  description  of  each
control and indicator is as follows:

| | |
|---|---|
| POWER Switch | A two-position, key-operated  switch  to turn main power on and off. |
| POWER Indicator | Lights to indicate power on status. |
| INTERRUPT Switches | Eight  momentary  switches,  numbered  0 through 7.  When a switch is pressed, an interrupt of the corresponding level  is generated on the Intellec MDS bus. |
| INTERRUPT Indicators | Eight indicator lights, each  associated with  the corresponding interrupt level. Lights  when  an  interrupt  at  the associated  level  is pending.  Goes out when the interrupt is serviced. |
| HALT Indicator | Lights when the Intellec MDS is  in  the HALT state. |
| RUN Indicator | Lights when the Intellec MDS is  in  the RUN state. |
| | If both the HALT and RUN indicators  are off,  and  power  is on, the MDS is in a WAIT state.  If both are on, the MDS  is oscillating  rapidly between the RUN and HALT states. |
| BOOT Switch | When operated, switches the Intellec MDS into the BOOTSTRAP mode. |
| RESET Switch | Momentary  action  switch which,  when operated,  resets the program counter in the  8080  to  address  0,  clears  all pending interrupts, and resets all other MDS logic to its initialized state.  The contents  of  RAM,  however,  are  not affected. |

## Attaching Peripherals to the Intellec® MDS

The Intellec MDS back panel contains two rows of six slots each, used for mounting cable connectors for various peripheral devices. In a standard Intellec MDS, the rightmost six slots are occupied by connectors for six standard peripherals: a Teletype, a CRT, a paper tape reader, a paper tape punch, a line printer, and the Universal PROM Programmer. The function of each connector is shown in a legend, which is stenciled in the upper right-hand corner of the MDS back panel. Additional spaces are provided on the legend for the user to indicate the functions of additional connectors he may install.



Figure 2-1: Intellec® MDS Front Panel Controls And Indicators

## Teletype Description

An ASR-33 Teletype is used in various applications. This peripheral contains a printer, keyboard, paper tape reader, and a paper tape punch. The ASR-33 Teletype is pictured in Figure 2-2. The controls are labeled in the figure; the function of each control is listed as follows:

| | |
|---|---|
| CONTROL KNOB | Three-position switch: |
| | OFF     Turns Teletype off. |
| | LINE    Teletype is on and connected to Intellec MDS. |
| | LOCAL   Teletype is on, but is not connected to Intellec MDS. |
| KEYBOARD | The keyboard contains the operating keys to produce printing, non-printing, and control characters. |
| 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 the output characters transmitted from the computer. When the control knob is in the LOCAL position, the printer prints the characters typed in on the keyboard. |
| 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. |
| 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 characters which are punched on paper tape. This information is read at a rate of 10 characters per second. |
| START | Begins reading tape. |
| STOP | Stops reading tape. |
| FREE | Disengages paper tape in the reader by releasing the sprocket wheel. Allows tape to be pulled through reader. |

Before an ASR-33 Teletype can be used with the Intellec MDS, several simple modifications must be made to the Teletype. These are described in the Intellec MDS Hardware Reference Manual.

Operation of the ASR-33 Teletype is described in the Teletype operator's

manual.    This   information    includes   loading   paper   into the printer,
loading paper tape into the punch, inserting punched paper tape into the
tape reader, and general operations of the ASR-33 Teletype.

DIRECTION
OF FEED

COPY HOLDER

ELECTRICAL
SERVICE
UNIT WITH
MODEM

MODEL 33 PAPER TAPE PUNCH

MANUAL
START

AUTO

MANUAL
STOP

FREE

START

STOP

FREE

CHAD
BOX

OFF

LOCAL

LINE

PEDESTAL

AUTOMATIC READER
CONTROLS

MANUAL READER
CONTROLS
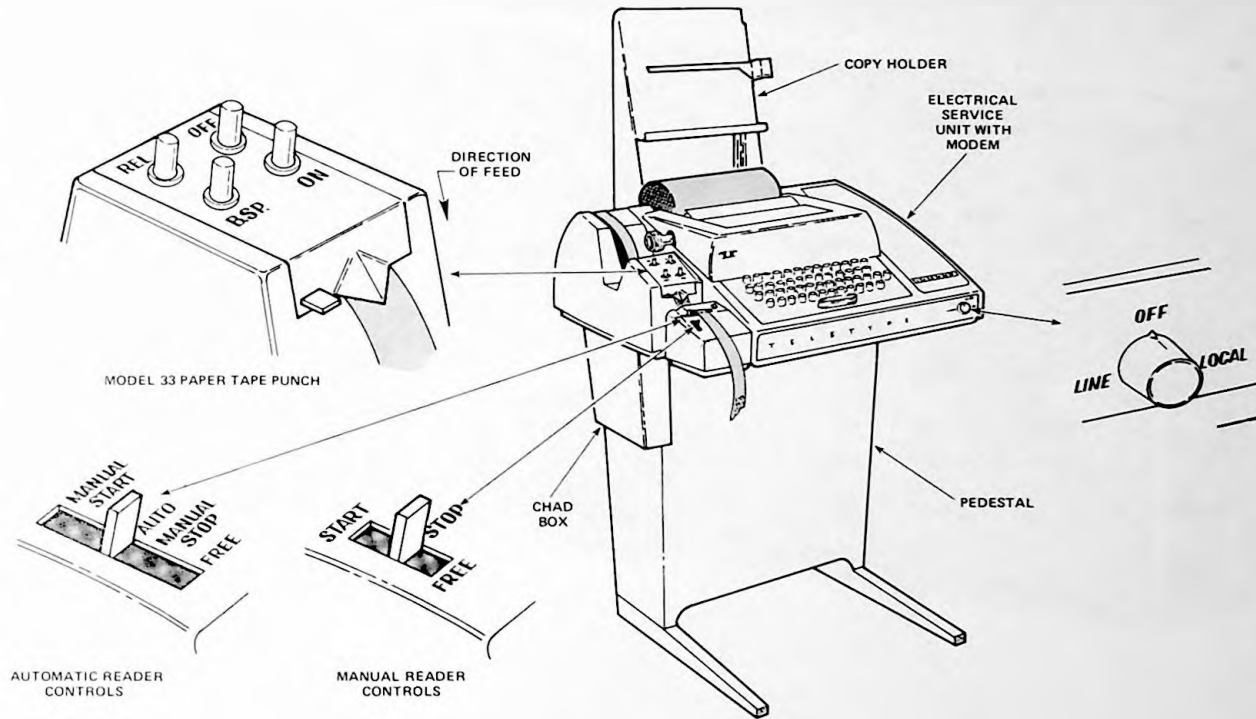
Figure 2-2: ASR-33 Teletype Controls

## High Speed Paper Tape Reader

The MDS-PTR High-Speed Paper Tape Reader, shown in Figure  2-3,  permits
paper tape to be loaded into the Intellec MDS approximately twenty times
as fast as is possible using the Teletype.  Operation of the  reader  is
very simple:

- Connect the cable leading from the Paper Tape Reader  to
  the socket labeled "PTR" on the Intellec MDS back panel.

- Turn the tape reader on, and load the paper tape  to  be
  read  into  the  left  hopper,  with  the sprocket holes
  toward the rear.

- Raise the tape gate over the sprocket wheel, and  thread
  the  tape  under  the metal tape guide on the left, over
  the sprocket wheel (making sure the  wheel  engages  the
  sprocket holes in the tape), and under the tape guide on
  the right.  If enough leader is available on  the  tape,
  arrange  one  or  two folds of tape in the bottom of the
  right hopper.

- Lower the tape gate onto the tape,  thereby  holding  it
  against the sprocket wheel.
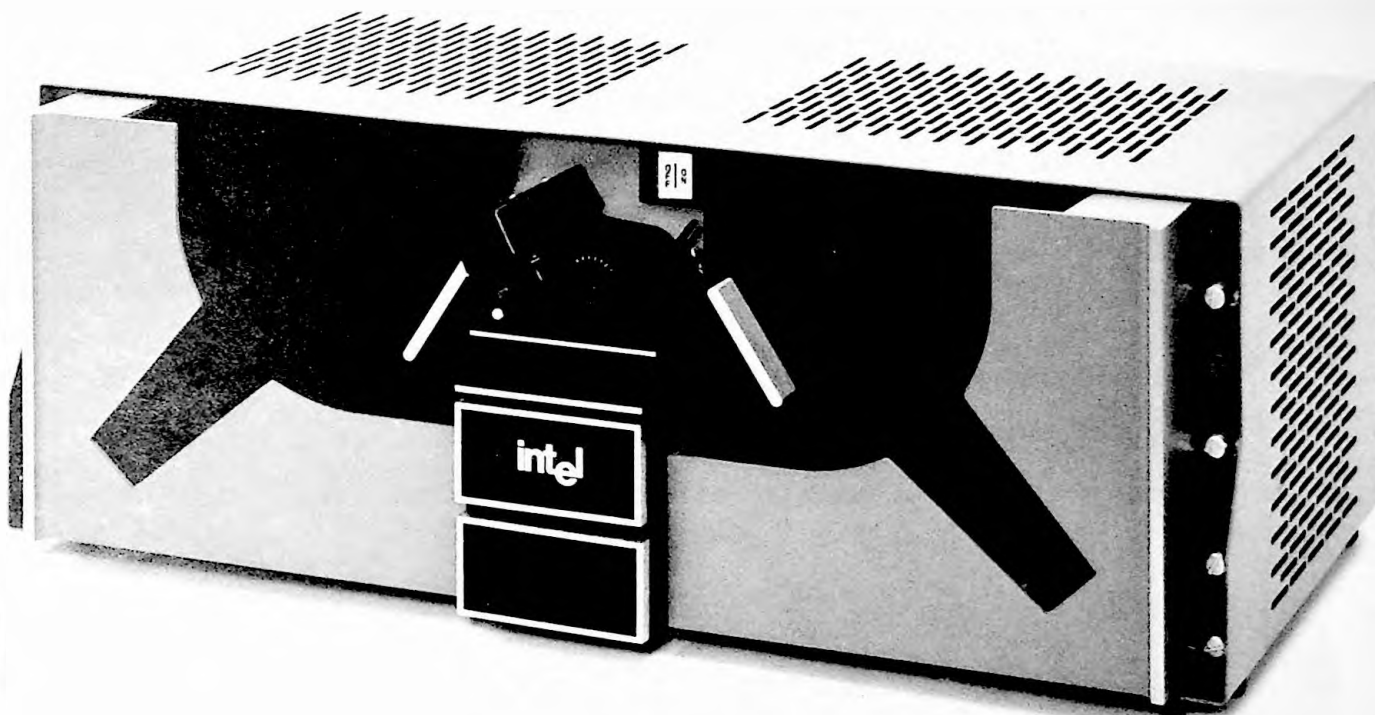
Figure 2-3: MDS-PTR High-Speed Paper Tape Reader

## Universal PROM Programmer

The MDS-UPP Universal PROM Programmer allows the Intellec MDS user to
program any Intel PROM.  The UPP contains two zero-insertion-force
sockets, labeled "Socket 1" and "Socket 2".  Socket 2 is used for
programming 24-pin PROMs;  Socket 1 can be ordered in both a 16-pin and
a 24-pin configuration. Programming different kinds of PROMs involves
merely inserting the proper "personality card" for that PROM (available
from Intel) into a slot in the UPP.

As shown in Figure 2-4, the front panel of the UPP is very simple and
straightforward.  Besides the PROM sockets, there are two switches
(POWER and RESET) and two lights (POWER and PROGRAMMING).  The POWER
light indicates that AC power is applied to the UPP; the PROGRAMMING
light indicates that the UPP is performing a programming operation.  The
RESET switch is not normally needed, but can be used to reinitialize the
UPP if it is necessary to abort a programming operation in progress.

The UPP is operated as follows:

- Connect the cable from the UPP to the socket labeled "PROM" on the MDS back panel.

- Flip the POWER switch on the UPP to its UP position, and check that the POWER light comes on.

- Move the locking arm on the appropriate PROM socket to its released position (sticking out). Insert the PROM in the socket, and move the locking arm to its locked position (up).

- Enter the appropriate Monitor command on the MDS CONSOLE, as described in Section 3 of this manual.

- When the PROM operation is complete, lower the locking arm on the UPP socket and remove the PROM.



Figure 2-4: Universal PROM Programmer

## STARTUP PROCEDURES

The startup procedures consist of turning the system power on and
initiating operation of the Monitor.  Procedures are included for a cold
start from a power off condition, executing a user program, interrupting
a user program temporarily, and manually terminating a user program.

## Cold Start Procedures

To start the Monitor, perform the following:

- Turn power on by inserting key in power switch and
  turning clockwise.  The POWER indicator should light.

- Set the BOOT switch to ON.

- Press the RESET switch.

- Type the "space" character on the device that is
  selected to be the system CONSOLE.  The Monitor polls
  all devices capable of assuming the console function
  (Teletype or CRT), and assigns the first device that is
  used as the CONSOLE.

  The Monitor then types the following message:

  MDS MONITOR, Vx.x

- After the above message is typed, set the BOOT switch to
  the OFF position.  The Monitor them prompts with a
  period (.) to indicate that it is ready to accept a
  command.

## Return to Monitor from User Program

If a user program is running, the Monitor can be reentered and the user
program terminated by pressing the INTERRUPT 0 switch.  The Monitor
responds by printing an asterisk (*) and the value of the user´s Program
Counter at the point where the interrupt occurred.  The Monitor message
may appear as follows:

        *FC98

If the user´s program has disabled the 8080 interrupt system (using a DI
instruction, for example), or the program or the operator has altered
the Intellec MDS Interrupt Mask Register (with an X command from the
CONSOLE, for example), the Intellec MDS may not be sensitive to an
INTERRUPT 0.  In this case, it will be necessary to repeat the BOOTSTRAP
procedure described above to re-enter the Monitor.

SECTION 3

THE MDS MONITOR

## INTRODUCTION

The Intellec®MDS Monitor is an Intel 8080 program available as eight 1702A PROMs or one 8316 Mask-Programmed ROM. The Monitor accepts and acts upon user commands to operate the Intellec MDS system. It also provides input and output facilities in the form of I/O drivers for user peripheral devices. The Monitor provides the following facilities:

- Punching selected areas of memory onto paper tape in BNPF or hexadecimal format.

- Checking the contents of a PROM by comparing with selected areas of memory.

- Displaying selected areas of memory.

- Initiating execution of user programs.

- Modifying contents of memory and processor registers.

- Inputting hexadecimal file from external device to memory

- Transferring the contents of an external PROM into memory.

- Inserting breakpoints into user programs before execution.

- Loading and executing the Text Editor and Assembler.

- Programming PROMs in conjunction with the Universal PROM Programmer (UPP).

The Monitor communicates with the user through an interactive console
device, normally a Teletype. The dialogue between the operator and
Monitor consists of user-originated commands in the Monitor's command
language, and Monitor responses, either in the form of a printed message
or an action being performed. After the cold start procedure (described
under the heading, "STARTUP PROCEDURES" in Section 2), the Monitor
begins the dialogue by typing the sign-on message on the console and
requesting a command by presenting a prompt character, '.' (period).


## MONITOR OPERATIONS

The Intellec®MDS Monitor is a command controlled operations supervisor
for the Intellec MDS system. Control commands are discussed in the
paragraphs titled, "Intellec MDS Command Structure". The following
discussion considers the control commands in functional order.


## Monitor I/O Configuration

The Monitor is designed to have versatile I/O facilities so that a user
may select the peripheral device that is to be used for each of the I/O
functions. The Monitor itself must have at least four system devices
assigned to perform the functions defined as CONSOLE, READER, PUNCH, and
LIST. The user is then given the option to select the actual peripheral
device which is to perform the required system operation. Where
applicable, a peripheral device may be used for more than one system I/O
function. Selection of I/O devices for system use is the function of
the ASSIGN or A command and is discussed under the paragraph titled,
"I/O ASSIGNMENT".

The user is also given the facility to use non-standard I/O peripheral
devices. No system coding (drivers) are provided to use these devices.
However, system linkages are provided so that a user may prepare a
device driver and link it to the Monitor I/O system. Procedures to code
and link user generated device drivers is discussed in Section 6.

The Q command is included to notify the user of the assignment status of
peripheral devices. When the Intellec MDS is initialized, default I/O
assignments are made by the system. Following the initialization
procedures, the device assignments may be changed as desired. The Q
command provides the required quick check of system configuration.


## Memory Control

The DISPLAY, MOVE, FILL, and SUBSTITUTE commands provide the user with
control of Intellec MDS memory. These commands allow the user to read
and write data in the MDS memory.

The DISPLAY command, as the name implies, allows the user to display the
contents of blocks of memory on the LIST device. The MOVE command
allows the user to move blocks of memory from ROM or RAM into RAM. The
FILL command allows the user to replace the contents of all locations in

a specified memory area with a constant value.  The  SUBSTITUTE  command
allows  the user to examine the contents of individual memory locations,
modifying each as it is displayed or leaving it unchanged.


## Register Control

The 8080 CPU registers and the Intellec MDS Interrupt Mask Register  may
be  examined and modified with the X command.  The Monitor maintains its
own environment and the user operating environment separate.   It  saves
and  restores  each  environment each time control is passed between the
user´s program and the Monitor.  The X command provides  the  user  with
the  capability  to examine amd modify the data in the user´s registers.
The registers associated with the Monitor cannot be accessed or modified
with this command.


## Paper Tape Input Control

Two commands, READ HEXADECIMAL FILE and LOAD BNPF  TAPE,  are  included.
These  commands  read  data from the PUNCH device and interpret the data
according to the specified format.  Hexadecimal  and  BNPF  formats  are
described in Appendix D.

The R command (READ  HEXADECIMAL  FILE)  is  used  to  read  a  file  of
hexadecimal  records,  terminated  with an End-of-File record.  An entry
point address may be optionally included  in  the  EOF  record.   The  L
command  (LOAD  BNPF  TAPE)  is  used  to read BNPF words from tape into
memory.  Because a low memory address and  a  high  memory  address  are
specified  in  the B command, only the number of words needed to fill the
specified memory area are read.  The  command  is  terminated  when  the
specified memory is filled.  The R command is open ended, waiting for an
EOF record to be read in to terminate the command.


## Paper Tape Output Control

Four commands are included to control data output to the  PUNCH  device.
The  B  command  formats memory data into BNPF words and outputs it onto
punched paper tape.  The  format  of  the  BNPF  tape  is  four  words,
separated  by single spaces, and ending with a carriage return/line feed
character pair.  Thus, when printing the contents of BNPF  tape,  it  is
four words to a line.

The WRITE MEMORY or W command formats memory into hexadecimal  formatted
records.  Each record contains its own load address, data, and checksum.

The END OF FILE or E command is used to terminate a file generated  with
the  W  command.   Facilities  are  provided  to  include an entry point
address which is used with the G command.

The NULL or N command is used to produce leader and trailer on the paper
tape files.  Use  of  this command causes sixty blank characters to be
punched on the tape (sprocket holes only).  The command may be used more

than once to generate greater lengths of leader.

## Universal PROM Programmer Control

Three commands are included to control the operations of the UPP.  These
consist  of the COMPARE or C command, the PROGRAM PROM or P command, and
the TRANSFER PROM or T command.

The COMPARE command is used to compare the contents of a  PROM   that  is
installed  in a socket on the UPP with the contents of a specified range
of memory locations.  If  differences  are  encountered,  an  output  is
generated for the CONSOLE device showing the contents of both memory and
PROM at the location that did not compare correctly.

The PROGRAM PROM command transfers the contents of a specified  area  of
Intellec  MDS memory into a PROM which is installed into a socket on the
UPP.

The TRANSFER PROM or T command transfers the contents of a PROM which is
installed in the UPP, into a specified area of Intellec MDS memory.

## User Program Execution

The PROGRAM EXECUTE command, G, is used to execute a user program.  This
command transfers control to the user´s program at a specified location.
Prior to transferring control, the Monitor sets up the user environment.
As  control  is passed back and forth between the Monitor and the user´s
program, the operating environment is saved and and reestablished.   The
state  of  user  registers,  as  displayed with the X command (described
above), are the values of the register contents just before  control  is
transferred to the Monitor.

The G command also  allows  the  operator  to  set  breakpoints  in  his
program.   When  the  program  arrives  at a breakpoint, control will be
returned to the Monitor, allowing the user to examine the status of  MDS
memory, the 8080 CPU registers, etc.

## Utility Routine

As  an  added  convenience,  the  Monitor  will  perform  addition   and
subtraction  of  any  two four-digit hexadecimal numbers, displaying the
results on the CONSOLE.

## COMMAND DESCRIPTIONS

Intellec MDS Monitor commands consist of a single  alphabetic  character
specifying  a  command,  followed  by  a  list  of numeric or alphabetic
characters.  Numeric parameters  are  entered  as  hexadecimal  numbers.
Leading zeros may be omitted.  Single commas (,) or space characters may
be used as delimiters between arguments  in  the  parameter  list.   The

valid range of numerical values is from 0000 through FFFFH (0 through
65,535 decimal). Longer numbers may be entered, but only the last four
hexadecimal digits are significant. Alphabetic parameters may be a
single character or a string of characters; the specific requirement is
described in the appropriate command description. The normal command
terminator character is the carriage return. Command syntax is
discussed in the individual command descriptions.

In the examples contained within the command descriptions, all
characters typed by the user are underlined to distinguish them from
characters generated by the Monitor. Commands may be terminated before
execution by entering a Control/C before the command terminator is typed
in.

Several Monitor commands require a pair of parameters which are referred
to below as "<low address>,<high address>". These parameters define a
section of MDS memory, which is to be used in the command. In all
commands in which this address pair is used, the following applies:

> The command begins execution at <low address>, and continues
> until <high address> is exceeded. The data at <high address>,
> therefore, is included in the command's scope.

> Normally, <low address> will be less than or equal to <high
> address>. If, however, the value specified for <low address>
> exceeds that specified for <high address>, the command acts on
> the data at <low address>, then terminates.

The following paragraphs contain descriptions of the individual
commands. A summary of this information is also included in Appendix A.


I/O System Configuration Commands - A, Q


I/O ASSIGNMENT - A

> A<logical device>=<physical device>


The Monitor requires that at least four system peripheral devices be
assigned as a minimum complement of peripheral equipment. These devices
may be selected at run time from the group of peripherals (physical
devices) which are available to the system. If no selection is made
with the A command, default values are assigned by the system.

The system devices are defined as follows:

> C or CONSOLE     An interactive, character-oriented input/output
>                  device.

> R or READER      A character-oriented input device, such as a
>                  paper tape reader, which transfers data upon

command and signals the calling program when no
more data is available (End-Or-File).

P or PUNCH          A character-oriented output device, such as a
paper tape punch, which accepts a character from
the calling program an records it on an external
medium.

L or LIST           A character-oriented output device which accepts
a character from the calling program and records
it on some external medium in human readable
form.

One of four peripheral devices may be selected for each system device.
The physical device specified must be able to perform the functions
which are defined for the system device. Thus, a line printer (LPT)
cannot be specified as the CONSOLE; or a paper tape punch (PTP) cannot
be specified as the READER. The standard device designations which are
discussed below refer to system devices normally supplied with the
Intellec MDS system. Drivers for these peripheral devices are included
in the Monitor. It should be noted that the designation ´CRT´ does not
apply to all CRT devices, but only to those that are compatible with the
MDS system. Similarly all of the device designations used herein apply
to system compatible devices. Facilities are included for the user to
write his own device-specific driver coding and link it to the Monitor.
This allows a user to use his own console, paper tape punch/reader, and
listing device which may appear as non-standard equipment to the
Intellec MDS system. The standard devices are designated with the
following symbols. Either the single letter abbreviation or three
letter (or more) designation may be used:

T or TTY            Teletype console with keyboard, printer, paper
tape reader, and paper tape punch.

C or CRT            Compatible CRT.

B or BATCH          BATCH mode is a non-interactive mode in which
CONSOLE input is read from the assigned READER
device and written to the assigned LIST device.
In preparing a command file for BATCH input, the
user should enter commands in exactly the same
way as if the system were in interactive mode.
Each command should end with a carriage
return/line feed pair. The period (prompt)
character which is generated by the Monitor in
interactive mode should not appear as part of
the command. Since the Monitor will continue to
read from the READER until the CONSOLE is
reassigned, the last command in the BATCH
command file should reassign the CONSOLE to
prevent the Monitor from reading off the end of
the tape.

| | |
|---|---|
| P or PTR | When used with the READER, this designation refers to the high speed paper tape reader. |
| P or PTP | When used with the PUNCH, this designation refers to the high speed paper tape punch |
| L or LPT | System line printer. |

The following are valid device assignments. Only the first character need be entered;  the additional characters in each device name are allowed for clarity.


CONSOLE Assignment

.AC=x     or  .ACONSOLE=x

Select one of the following for ´x´:

| | |
|---|---|
| T or TTY | (Default as assigned during Cold Start |
| C or CRT | sequence described in Section 2) |
| B or BATCH | |
| 1 | Specifies a user-defined device for which a user-written device driver is present. |


READER Assignment

.AR=x.   or  .AREADER=x

Select one of the following for ´x´:

| | |
|---|---|
| T or TTY | (Default) |
| P or PTR | |
| 1 | Specifies a user-defined device for which a user-written device driver is present. |
| 2 | Specifies a user-defined device for which a user-written device driver is present. |


PUNCH Assignment

.AP=x    or  .APUNCH=x

Select one of the following for ´x´:

| | |
|---|---|
| T or TTY | (Default) |
| P or PTP | |

1                          Specifies a user-defined device
                           for which a user-written device
                           driver is present.

2                          Specifies a user-defined device
                           for which a user-written device
                           driver is present.


LIST Assignment

.AL=x    or  .ALIST=x

Select one of the following for ´x´:

T or TTY          (Default)
C or CRT
L or LPT
1                          Specifies a user-defined device
                           for which a user-written device
                           driver is present.


SYSTEM STATUS QUERY - Q


        Q


The Q command allows the operator to determine the current status of the
assignable I/O system.   The  Q command produces a list of the logical
devices and the physical devices currently assigned to them.

The Q command causes the following to be printed on the CONSOLE device.

        .Q

        C=T
        R=P
        P=T
        L=T



Memory and Register Control Commands - D, F, M, S, X


DISPLAY MEMORY - D

        D<low address>,<high address>

The contents of the memory area defined by  the  input  parameters  <low
address> through <high address> are output as a formatted listing on the
LIST device.   Each line of the listing begins with the  address  of  the

first memory location displayed on that line. Following the four
hexadecimal digit memory address are up to 16 two-digit hexadecimal
numbers, which represent the contents of contiguous memory locations.

For example, if we wish to display the contents of memory locations 09H
through 2AH, the D command will appear as follows:

        .D9,2A

The requested data will be output on the LIST device, formatted as
follows:   (data are typical)

        0009 00 11 22 33 44 55 66
        0010 77 88 99 AA BB CC DD EE FF 10 20 30 40 50 60 70
        0020 80 90 A0 B0 C0 D0 E0 F0 01 02 03

FILL - F

        F<low address>,<high address>,<constant>

The F command may be used to initialize an area of RAM to a constant
value.   The <low address> parameter specifies the low memory address of
the range;  the <high address> parameter specifies the high memory
address of the range.  The <constant> parameter specifies the 8-bit
pattern to be stored in the designated memory.  If an area in ROM or
PROM is inadvertently specified within the range, the operation will
continue until its normal completion even though no writing will take
place in ROM/PROM.

For example, if the user wishes to write ´FF´ in all memory from
addresses 0020H through 0FFFH, the command will appear as follows:

        .F20,FFF,FF

MOVE MEMORY - M

        M<low address>,<high address>,<destination address>

The M command may be used to reposition blocks of data in memory.
Memory data is fetched from one location, specified by parameter <low
address>, and stored in location <destination address>.  Parameters <low
address> and <destination address> are then incremented and the process
continues until the value of <low address> is greater than or equal to
the value of <high address>.

For example, the following M command will move the contents of memory
locations 100 through 2FF to locations 600 through 7FF:

        .M100,2FF,600

SUBSTITUTE MEMORY - S

> S<address>

The S command displays memory locations on an individual basis and allows the user the option of modifying each location as it is displayed. The procedure for using the S command is as follows:

1. Type in an S, followed by the hexadecimal address of the first memory location that is to be examined. Type in a space. The contents of the location will be displayed, followed by a dash.

2. You now have the facility to type in new data, followed with a space or carriage return; a space character only; or a carriage return character only.

   Typing in a space character signifies that you wish to retain the displayed data, in the specified address. The contents of the next memory location will then be displayed followed with a dash

   To modify the contents of the displayed location, type in a two-digit hexadecimal number. The contents of the specified location will be changed. If more than two characters are typed, only the last two will be used.

   To terminate the command, type in a carriage return. If the previous operation was entry of modification data, the Monitor will replace the contents of the specified location with the new data and terminate the command. If the previous operation was simply to display the specified location (space was typed in) the Monitor will terminate the command.

For example, a user wishes to examine the contents of locations 20 through 27 and each time the value 7C is found replace it with FF. The S command will appear as follows:

        .S20 03- 1E- 41- 7C-FF D3- 46- E4- 39-
        .D20,27
        0020 03 1E 41 FF D3 46 E4 39


EXAMINE AND MODIFY CPU REGISTERS - X

> X<register identifier>

The X command displays and permits modification of the CPU registers. The <register identifier> parameter selects the particular register to be displayed, or if omitted, select all CPU registers for display. The

<register identifier> parameter is a single alphabetic character defined
as follows:

      A         8080 CPU A Register
      B         8080 CPU B Register
      C         8080 CPU C Register
      D         8080 CPU D Register
      E         8080 CPU E Register
      F         8080 CPU Flag byte.  The values of the 8080´s  condition
                flags are packed into this byte, as follows:

```
        7 6 5 4 3 2 1 0
        | | | | | | | |--------Carry
        | | | | | | |----------Always 1
        | | | | | |------------Parity
        | | | | |--------------Always 0
        | | | |----------------Auxiliary Carry
        | | |------------------Always 0
        | |--------------------Zero
        |----------------------Sign
```
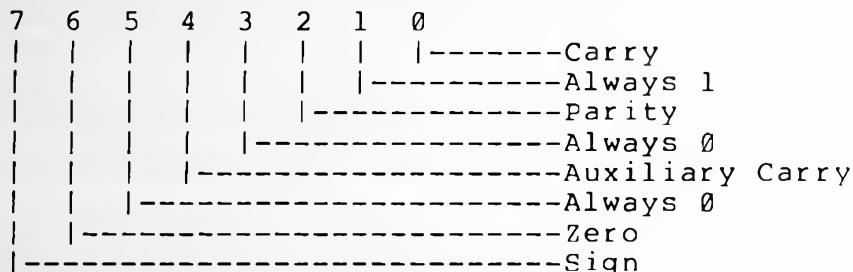
      H         8080 CPU H Register
      I         Intellec MDS Interrupt Mask
      L         8080 CPU L Register
      M         8080 CPU H and L Registers combined
      P         8080 CPU Program Counter
      S         8080 CPU Stack Pointer


The procedure for using the X command is as follows:

      1.        Type in an X, followed by either a <register identifier>
                or a carriage return.

      2.        If a carriage return is typed in, an  alphabetical  list
                of  all registers and their contents is displayed on the
                CONSOLE.  A typical list would appear as follows:

    A=AA B=23 C=CC D=01 E=EE F=FF H=12 I=FB L=34 M=1234 P=0100 S=2F00


      3.        If a register identifier is typed in  following  the  X,
                the  contents of the register are displayed (two or four
                hexadecimal  digits  are  displayed,  depending  on  the
                selected  register).   A  dash (-) character follows the
                last hexadecimal digit.

                .XE EE-


      4.        The register may be modified at this time by  typing  in
                the  new value, followed by either a space or a carriage
                return.  If no register modification is  required,  just
                type  in  the  space  character  or  carriage  return

character, without the register modification data.

5.      If a space character is typed in, the next register,  in
        alphabetic  sequence,  is  displayed.   The procedure in
        step 4 (above) may be repeated.

6.      If a carriage  return  is  typed  in,  the  register  in
        question  is  modified (if modification was performed in
        step 4), then the command is terminated.

For example, if the user  wishes  to  see  all  of  the  registers,  the
following command should be used:

        .X

This command will  display  all  of  the  registers  and  terminate  the
command.

If the user wishes to modify Register A, he may type in:

        .XA

The Monitor responds by printing the contents of Register A  immediately
following the command, as follows:

        .XA 7F-

If the user wishes to modify the contents from 7F to 00, he may type  in
the hexadecimal value 00, as follows:

        .XA 7F-00

Pressing the space key immediately following  the  modified  data  input
causes  the  Monitor to display the next register, which in this case is
the B Register.  The B Register may similarly be modified.  However,  if
instead of the space key, the carriage return key were pressed following
the modified data entry, the command would be terminated.


Paper Tape I/O Commands - B, L, R, W, E, N


BNPF PUNCH - B

        B<low address>,<high address>


The BNPF  PUNCH  command  punches  the  contents  of  the  memory  range

specified  by <low address> through <high address> in BNPF format on the
PUNCH device.  Output data is formatted into groups of four words,  each
group ending with a carriage return/line feed character combination, and
the  words  within  each  group  separated  with  spaces.   A  detailed
description of BNPF format is included in Appendix D.

The following example will punch the contents of memory  locations  100H
through 1FFH:

        .B100,1FF


LOAD BNPF TAPE - L

        L<low address>,<high address>


The L command loads a BNPF tape in the  Intellec  MDS  system  from  the
READER device.  Starting and ending locations of the loading process are
specified  by  the  parameters  <low  address>   and   <high  address>.
Additional  BNPF  words  on  the tape, in excess of the RAM memory range
specified by <low address> and <high address> are  disregarded.   if  an
insufficient number of BNPF words are available to fill in the specified
memory range, what is available  will  be  loaded;   the  excess  memory
locations will remain unchanged.  An error prompt (*) will be output.

An example of the L command is as follows:

        .L100,1FF


READ HEXADECIMAL FILE - R

        R<bias>


In the Intellec MDS  system,  object  programs  are  normally  saved  on
punched  paper  tape in hexadecimal format.  This format is described in
detail in Appendix D.  The R command reads a hexadecimal tape  from  the
READER  device  and  loads  the data into the locations specified by the
address fields in the hexadecimal records.

The bias address  is  added  to  the  load  addresses  in  each  of  the
hexadecimal  records  and the data is loaded into a memory area which is
offset by the value of the bias  address.   For  most  applications  the
address  is  zero.   The  data  which  is loaded remains unchanged;  the
addition of the bias address does not imply that  the  program  code  is
relocatable.   In  most cases, the code cannot be executed at the biased
location.

A typical R command will appear as follows:

        .R0

WRITE HEXADECIMAL FILE - W

        W<low address>,<high address>


The W command outputs portions of Intellec MDS memory to punched paper
tape, on the PUNCH device.  Data is in hexadecimal format.

Multiple W commands may be used to save non-contiguous memory areas as
one file.  The final record in the file must be an End-of-File record,
which is generated with the E command.  Refer to the description of the
E command for details.

An example of the Write Hexadecimal file operation is as follows:

        .W200,3AF

This command punches out the contents of memory locations 200 through
3AF.


END FILE - E

        E<entry point address>


The E command is used in conjunction with the W command during
production of a hexadecimal output file, to generate the End-of-File
record.  The <entry point address> input parameter defines the entry
point of the file for subsequent execution.  If this is non-zero, the
subsequent R command (which reads the hexadecimal tape) transfers the
<entry point address> to the user´s Program Counter.  This allows the
user to start execution of the program immediately after reading the
tape with the R command, by entering the G command without giving
further consideraton to a starting address.  If the value of the <entry
point address> is zero, the subsequent R command does not alter the
user´s program counter.

For example, assume a hexadecimal formatted tape is puched with the W
command.  An End-of-File record must also be punched to properly
terminate the file.  Also, the user wishes to start execution of the
program represented by the punched paper tape at address 1000H.  The E
command would appear as follows:

        .E1000


NULL LEADER/TRAILER - N

        N


The N command produces a tape leader or trailer by outputting sixty NULL
characters on the PUNCH device.

## Universal PROM Programmer Commands - P, C, T

The Universal PROM Programmer allows the Intellec MDS user to program  a
wide    variety    of    Intel    PROMs.    The  UPP  is  available  in  two
configurations:  one configuration contains one 24-pin  socket  and  one
16-pin  socket, while the other contains two 24-pin sockets.  The 16-pin
socket, if present, must be in the "Socket 1" position on the UPP  front
panel.   The  16-pin  socket is used for programming PROMs having a word
size of 4 bits;  the 24-pin socket is used with PROMs having a word size
of 8 bits.

The three commands  which  are  used  with  the  UPP  each  require  two
alphabetic  parameters, in addition to numeric parameters.  One of these
is referred to below as <socket option>,  which  specifies  whether  the
PROM  being  acted upon is in Socket 1 or Socket 2 and, if the PROM word
size is 4 bits, with which half of an 8-bit MDS  memory  byte  the  PROM
word  corresponds.   This  parameter  may take on the values X, Y, or Z,
which have the following meanings:

> X          Select Socket 2 on the UPP for  this  operation.   Treat
>            all data as 8-bit quantities.
>
> Y          Select Socket 1 on the UPP.  Correlate  the  4  bits  of
>            data  in each PROM word with the 4 most significant bits
>            (bits 7-4) in each byte in MDS memory.
>
> Z          Select Socket 1 on the UPP.  Correlate  the  4  bits  of
>            data in each PROM word with the 4 least significant bits
>            (bits 3-0) in each byte in MDS memory.

The above discussion assumes that Socket 1 contains a 16-pin  PROM.   If
Socket  1  contains  a 24-pin PROM, the specification of <socket option>
must be considered separately for the C, P, and  T  commands.   This  is
dealt with below.

The other alphabetic parameter required by all  three  commands  is  the
<true/false>,  or  <t/f>,  parameter.   This  parameter  establishes the
"sense" of the PROM with respect to MDS RAM, as follows:

> If <t/f> = T, the Monitor assumes that the data in PROM  appears
> in the same sense as it does in MDS RAM;  i.e, a "1" bit in PROM
> corresponds to a  "1"  bit  in  RAM,  and  a  "0"  bit  in  PROM
> corresponds to a "0" bit in RAM.
>
> If <t/f> = F, the Monitor assumes that the data in PROM  is  the
> complement  of  the  corresponding data in MDS RAM;  i.e., a "1"
> bit in PROM corresponds to a "0" bit in RAM, and vice versa.

Each command below requires the UPP to be connected to the Intellec MDS,
with power on, at the time the command is entered.  If the UPP is not in
a READY state, the Monitor will immediately issue an error indicator (*)
as the command is entered.

PROM PROGRAMMING - P

P<t/f><socket option><low address>,<high address>,<PROM address>

The P command programs the PROM in the socket specified by <socket
option> with data taken from MDS memory locations <low address> through
<high address>. Data from <low address> is transferred to the PROM at
<PROM address>, where each PROM is assumed to have a starting address of
0. If <t/f> = F, the data from MDS memory is complemented as it is
transferred to the UPP. The data in MDS memory always remains
unchanged.

The Monitor always transfers 8 bits of data to the UPP for each location
being programmed. If the <socket option> parameter is Y or Z, and
Socket 1 contains a 16-pin PROM, the UPP uses the information provided
by the Y or Z to determine which four bits to use in the programming
operation. If Socket 1 contains a 24-pin PROM, the UPP will use all
eight bits of data in the programming operation. Thus, to program a
24-pin PROM installed in Socket 1 of the UPP, the operator may specify a
<socket option> of either Y or Z. If the 24-pin PROM is installed in
Socket 2, he should use a <socket option> of X.

The UPP reads back and compares each location it programs in PROM with
the original data it received from the MDS. If the two values differ,
it transmits an error indicator to the Monitor, which displays the
current <PROM address> value followed by an error indicator (*), then
terminates the programming operation.

NOTE: The P command in the MDS Monitor cannot be used to program the
2704, 2708, 8704, or 8708 PROMs. Intel distributes a special MDS
program for this purpose.


COMPARE - C

C<t/f><socket option><low address>,<high address>


The C command compares the contents of a PROM located on the UPP, in the
socket specified by the <socket option> input parameter, with the
contents of memory in the area specified by the input parameters <low
address> through <high address>. If the contents of a PROM location are
not equal to the contents of the corresponding memory location, the
memory address, the contents of the memory location, and the contents of
the PROM are printed on the CONSOLE for inspection.

The Monitor always reads 8 bits of data from the UPP. If the Y or Z
socket option is specified, the monitor masks off the appropriate 4 bits
from the contents of the selected memory location so that either the low
or high four bits remain to be compared with the 4 bits of the PROM
location. Assume that the Z socket option is specified. A typical byte
from the UPP may appear as follows:

07

The Monitor retrieves the corresponding byte from memory (let us say,
F7) and masks off the most significant 4 bits to produce a byte with a
value of 07. The comparison is then made. Similarly, if the Y socket
option is selected, the byte from the UPP would be ´F0´ and the
corresponding byte from memory would be masked to produce ´F0´, then the
comparison made. Therefore, to compare a 24-pin PROM installed in
Socket 1 of the UPP with the contents of MDS memory, the operator should
do two C commands: once using a <socket option> of Y, and once using a
<socket option> of Z.

For example, the contents of an 8-bit by 256 word PROM is specified as
true logic, and is to be compared with the contents of memory from
locations 0A00H through 0AFFH. The C command is as follows:

        .CTXA00,AFF

Assume that the contents of memory locations 0A06 and 0A91 are not equal
to the contents of the corresponding PROM locations. The Monitor will
print the following message:

        0A06 AA FF                    (Typical data)
        0A91 00 01

TRANSFER PROM - T

        T<t/f><socket option><low address>,<high address>

The T command transfers the contents of the PROM in the socket specified
by <socket option> to the area of MDS RAM specified by the <low
address>,<high address> pair. If the range of memory locations is
smaller than the contents of the PROM, the excess data in the PROM is
disregarded. If the range is greater than the size of the PROM, the
PROM data will be transferred; the excess memory locations will remain
unchanged; an error prompt (*) will be printed by the Monitor. If
<t/f> = F, the data coming from the UPP is complemented before being
stored in MDS RAM.

The Monitor always receives 8 bits of data from the UPP, and stores the
entire 8 bits in the next consecutive RAM location. Therefore, a
<socket option> of either Y or Z may be used to transfer data from a
24-pin PROM located in Socket 1 of the UPP.

For example, a user wishes to transfer an 8-bit by 256 byte PROM to
memory locations 100 through 1FF. The status of the data is false
logic. A typical T command appears as follows:

        .TFX100,1FF

PROGRAM EXECUTE - G

        G<start address>,<breakpoint 1>,<breakpoint 2>


The G command transfers control of the Intellec MDS from the Monitor
program to the user program starting at the location specified by the
<start address> parameter.

The <start address> parameter is a four-digit value which specifies the
address to be placed into the user's Program Counter.  If this parameter
is omitted, the stored value of the user Program Counter is used as the
starting address.   It is possible to have a previously stored value in
the user Program Counter due to four operations:

        1       When the user program is interrupted with the Interrupt
                0  switch on the MDS front panel, the status of the user
                registers, including the user Program Counter, are
                saved.   This value of the user program counter allows a
                user program to be resumed at the instruction
                immediately following the interrupt.

        2       When a non-zero <entry point address> is included in the
                End-of-File record in a hexadecimal object file.

        3       If the user has manually modified the user program
                counter with a Monitor X command.

        4       When the user program reaches a breakpoint set by a
                previous G command.


The <breakpoint 1> and <breakpoint 2> parameters are two 16-bit values
that specify breakpoint addresses in the user program.  If either is
omitted, no corresponding breakpoint is set.  If either breakpoint
address is encountered while executing the user program, both
breakpoints are reset and control is passed back to the Monitor.

A breakpoint enables the user to temporarily suspend execution of the
user program, examine the state of the program's memory and registers,
make modifications if desired, and then continue the program from the
point of suspension.  When the address where the breakpoint is inserted
in the user program is reached, the user program is terminated, all
pertinent user data is saved, and control is returned to the Monitor
program.  Immediately following a breakpoint, the value of the user
Program Counter points to the memory location in which the previous
breakpoint instruction was held.  The user program may thus be reentered
at a point just beyond the halt.

The Monitor implements breakpoints by saving the contents of the RAM
locations specified as the breakpoint addresses, then substituting RST 0
instructions at these addresses.  When the RST 0 instruction is
encountered during execution, the Intellec MDS hardware branches to
address 0, which contains a branch to the Monitor.  When the Monitor is

entered   through   this   entry point, all status of the 8080 CPU is saved
and the original contents of the breakpoint addresses are restored.

The MDS interrupt system is enabled when the Monitor is entered.    Since
the Monitor cannot determine the previous state of the interrupt system,
just prior to the exit from the user program,   the   assumption   is   made
that   the interrupt system was enabled.  When control is returned to the
user program, the interrupt system remains enabled.   It   is   the   user's
responsibility  to either enable or disable the interrupt system (EI and
DI instructions).

An interrupt from a user program may be performed by pressing the  front
panel INTERRUPT 0 switch.   The suspension of the user program is similar
to that performed by a breakpoint interrupt.   All   user   program   status,
including   the   Program   Counter,   is   saved.    When the user program is
resumed, the saved value of the program counter may be used   to   restart
the   program   at   the   instruction immediately following the interrupted
instruction.

To use the G command, proceed as follows:

> 1          Type a G.   If a starting address other than the  current
>            (stored)  value of the Program Counter is desired, enter
>            it immediately following the G.

> 2          If no breakpoints are  desired,   terminate   the   command
>            with a carriage return.

> 3          To set breakpoints, enter a comma or space.   The Monitor
>            will   type   a   dash   (-) to indicate that it is ready to
>            receive   a   breakpoint.   Enter   the   desired   breakpoint
>            address.

> 4          To set only one breakpoint, enter a carriage return.   To
>            set   another   breakpoint, return to step 3.   At most two
>            breakpoints may be entered.  If the character   following
>            the   second   breakpoint   is   not   a carriage return, the
>            Monitor will issue an error indicator (*) and abort   the
>            command.

For example, a user program starts at location 20 and a breakpoint is to
be   inserted   at   location 2FEH.   To start executing the program, a user
will enter the following command:

    .G20,-2FE

However, if the program was loaded from a hexadecimal tape, with   the   R
command,   and   an   entry  point address was specified in the End-of-File
record on the tape, the starting address need not be included in   the   G
command.   In this case, the user may enter the command as follows:

    .G,-2FE

The delimiter (,) immediately following the letter G indicates that   the

entry point is assumed to be at the location specified by the present
contents of the user program counter.  If desired, breakpoints may be
inserted following the delimiter, in the normal manner.

If the user wishes to start executing a program at location 1FA but does
not wish to use breakpoints, the command will be as follows:

        .G1FA

If a user wishes to re-enter a program after a breakpoint suspension or
a front panel INTERRUPT 0, and new breakpoints are to be specified at
locations FF and 1AB, the command will be as follows:

        .G,-FF,-1AB

If no breakpoints are to be specified, and the user wishes to reenter a
program after a breakpoint or interrupt, the command will simply be:

        .G

If the G command is aborted while it is being entered (e.g., by a
Control/C) or if it contains a syntax error, no breakpoints will be set.


HEXADECIMAL ARITHMETIC - H

        H<number 1>,<number 2>


For the convenience of the user, the Monitor has the capability of
performing simple hexadecimal calculations.  The sum (number 1 + number
2) and difference (number 1 - number 2) of the two numeric parameters
entered are calculated and displayed on the CONSOLE.  Arithmetic is
performed modulo 65,536 (2**16).  The input parameters may be up to four
digits in length.  Negative numbers must be entered in their two's
complement representation:  for example, -2 would be entered as FFFE.
Negative results will be displayed in two's complement form.

For example, a typical hexadecimal arithmetic operation may be performed
as follows:

        .H200,1FE
        03FE 0002


MONITOR ERROR CONDITIONS

The Monitor checks for several error conditions.  Depending on the
particular error, either an error indication is output on the CONSOLE,
or the command is rejected.  The Monitor's response to error conditions
is as follows:

INVALID CHARACTERS.  The Monitor checks the validity of  each  character
as  it  is  entered from the CONSOLE.  As soon as the Monitor determines
that the last character entered is illegal in its context, it aborts the
command and displays a ´*´ to indicate the error.

For example, suppose a character ´G´ is  entered  in  a  parameter  list
where  only  hexadecimal digits (0-9, A-F) and delimiters (comma, space,
carriage return) are valid,  the  output  on  the  console  will  be  as
follows:

        .H100,10G*


Suppose the character ´Y´ is used as a command.  The Monitor will reject
this character and indicate the error as follows:


        .Y*



ADDRESS VALUE ERRORS.  Many commands require  an  address  pair  of  the
form, <low address>,<high address>.  If, in these commands, the value of
<low address> is greater than the value of <high  address>,  the  action
indicated  by the command will be performed on the data at <low address>
only.

Addresses are evaluated modulo 65,536.  Thus, if a  hexadecimal  address
of  more  than FFFFH (four digits) is entered, only the last four digits
are significant.  For example suppose the following address  range  were
entered:


        .M04532AC,945216FCF,0


The above command would be equivalent to M32AC,6FCF,0.

Another type of address error may  occur  when  the  user  specifies  an
address  in memory which does not exist in the Intellec MDS system.  For
example, a user with a 16K system may enter an address above the highest
memory address, as follows:


        .M0,FFF,4000

     or .D6000,60FF


No error  indication  is  generated  by  the  Monitor  for  these  error
addresses.   In  general, if the source address (address from which data
is taken) is nonexistent, the data fetched  is  unpredictable.   If  the
destination  address (address to which the data is to be transmitted) is
nonexistent, the command has no effect.

CHECKSUM ERRORS.  If the Monitor determines that a data record  read  in
an  ´R´  command  contains  a  checksum error, all of the data from that
record is destroyed and an error indication of an ´*´ is output  on  the
CONSOLE.

If no checksum error exists, the Monitor reads the next  record  on  the
tape.


PROM PROGRAMMING ERRORS.  If an error is signalled from the UPP,  during
a  ´P´  command, the command is terminated and a ´*´ message is output on
the CONSOLE.  If the UPP is not connected to the MDS when a P, T,  or  C
command  is  input, an error condition is immediately indicated with the
´*´ message on the CONSOLE.


PERIPHERAL DEVICE ERRORS.  Non-existent peripheral  devices  or  devices
which are not ready cause a condition where the Monitor outputs the data
then waits indefinitely for  the  device  to  become  ready.   No  other
indication is provided.

SECTION 4

MDS TEXT EDITOR

## INTRODUCTION

The Intellec® MDS Text Editor enables the user to create and edit ASCII text files. The Text Editor may be used to edit any ASCII text; in this system it is used primarily to create and edit source programs for the 8080 MDS Macro Assembler.

The Text Editor is character oriented. That is, one or more characters in a line of text can be replaced or deleted, or new characters inserted without disturbing any of the other characters in the same line. Line numbers or other extraneous information need not be appended to the text in order for the Text Editor to operate correctly. All editing can be accomplished by using the commands which are described in the following paragraphs.

## GENERAL INFORMATION

The Intellec® MDS Text Editor occupies approximately 4K bytes of RAM memory. All of the remaining RAM memory is available for use as the Text Editor's working area, which is referred to as the Text Buffer.

The normal editing procedure to create a new file is to start the editor, type in the text, edit the text (to incorporate additions and corrections), and output the text file. Similarly, to edit an already existing file, one would start the editor, input the text file, edit the text, and output the edited version. If the file is already prepared and is being stored on punched paper tape, it may be entered through the system READER device. If a new file is to be created, it may be entered by typing the text on the system CONSOLE device.

### Text Buffer and Buffer Pointer

The text buffer is maintained by the Text Editor as the storage area into which text strings are placed. The size of the Text Buffer is

variable, enlarging as text is entered, and shrinking as text is
deleted.   When  the buffer is empty, the start of buffer and the end of
buffer coincide and the buffer size is zero.

Since the MDS Text Editor is character oriented, the Buffer  Pointer  is
needed  to  locate  the character which is to be acted upon.  The Buffer
Pointer is simply a  movable  position  indicator  which  is  positioned
either  between  two  adjacent characters, before the first character in
the  buffer  (start  of  buffer),  or  immediately  following  the  last
character  in  the  buffer (end of buffer).  The Buffer Pointer is never
positioned directly over a particular character, but may point before it
or  after  it.   Text  is  placed into the buffer at a point immediately
following the buffer pointer.  As each character is entered, the  buffer
pointer is moved ahead one character position.

The user has the facility, through certain of the Text Editor  commands,
to  move  the  buffer  pointer  to  any position inside the buffer.  The
buffer pointer cannot be moved beyond the boundaries of the text buffer;
it  cannot  be moved further back than the start of buffer position, nor
further forward than the end of buffer position.  Any command attempting
to  move  the buffer pointer beyond the boundaries of the Text Buffer is
terminated when the Buffer Pointer reaches the boundary, even though the
number of operations specified has not been completed.

Buffer Pointer movement may be either in terms of characters  or  lines.
The  consideration  of text in terms of lines is convenient because most
text is divided into lines.  A line of text, in the Text  Buffer,  is  a
string  of  characters  having  a  line feed (0AH) character as its last
character.  The next character in the Text Buffer, immediately following
the line feed, is in the next line.  If no line feed characters are used
in the text, the entire text is considered to be one line.


## Commands and Command Strings

Each editor command consists of  a  single  letter  command  designator.
Certain  of the commands take arguments.  Commands may be entered one at
a time or may be combined into command strings.  The Text Editor signals
its readiness to accept commands by printing a prompting asterisk (*) in
the left-most column of the system CONSOLE device.  Command strings must
be  terminated  with  a pair of ALT MODE or ESC characters (depending on
the type of the console device).  Except where  otherwise  noted  below,
individual  commands  in  a command string need not be terminated.  If a
text string is included in a command string, the  text  string  must  be
terminated with a single ALT MODE or ESC character.

Command strings are stored in the same memory area that is reserved  for
the  text  buffer.   In  order  not  to interfere with the text, command
strings are stored at the high end of the  memory,  above  the  'end  of
buffer'.   Command  strings  are stored in reverse order, with the first
character of the command in the  highest  available  location,  and  the
remainder  of  the  characters  in  the command in descending locations.
When the reserved  area  is  full,  the  command  part  may  attempt  to
overwrite  text.   This  condition is trapped.  The procedures to escape

this condition are discussed in the following paragraphs.

## Examples of Commands

The following examples show a typical command, command string, and command string combined with a text string.  In all examples in this section, characters typed by the operator are underlined;  those typed by the Monitor or Text Editor are not.
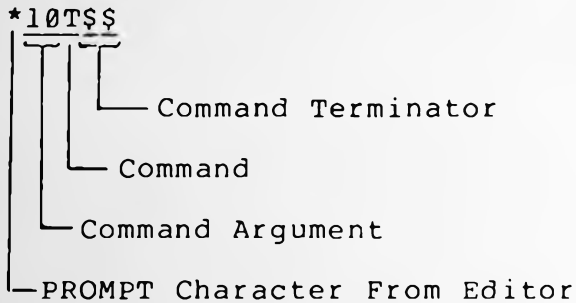
```
*10T$$
```
         └── Command Terminator

      └── Command

    └──Command Argument

└──PROMPT Character From Editor

This command causes ten lines of text to be printed on the system CONSOLE device.

```
*B20K5T$$
```
         └Command Terminator

      └Command with Argument

    └Command with Argument

  └Command

PROMPT Character From Editor
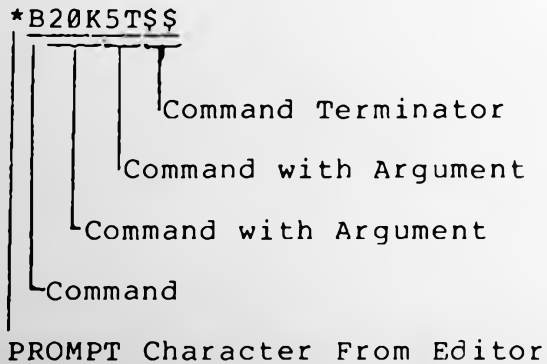
This command string causes the Buffer Pointer to be moved to the start of the buffer, 20 lines of text to be deleted, and the following five lines to be output on the system console device. The command terminator is placed at the end of the command string; the individual commands do not need separate terminators.
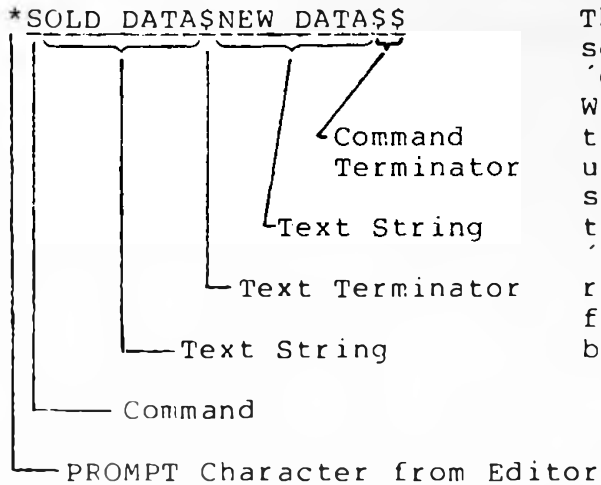
```
*SOLD DATA$NEW DATA$$              This command string causes a
 └┬┘└───┬───┘└───┬───┘└┬┘          search for the text string
  │     │        │     │           'OLD DATA' in the text buffer.
  │     │        │     └─Command   When found, it is deleted and
  │     │        │       Terminator the text string 'NEW DATA' is
  │     │        │                 used as a replacement. The
  │     │        └─Text String     single $ character represents
  │     │                          the text terminator for
  │     └─ Text Terminator         'OLD DATA'; the pair $$
  │                                represents the text terminator
  │                                for 'NEW DATA' in addition to
  └────Text String                 being the command terminator.
  │
  └───── Command

  └────PROMPT Character from Editor
```

## OPERATING FEATURES

User input to the Editor may be either a command, a command string, or a
text string.   The Editor signals the user that it is ready to accept a
new command by printing an asterisk (*) in the left-hand column  of  the
system  CONSOLE  device.  The user may specify text input by issuing the
INSERT command.   User  input  immediately  following  this  command  is
accepted  by  the  Editor  as  text.  As long as the editor is accepting
input characters as text, the PROMPT character (*)  is  not  issued.    A
command  or  command  string is executed immediately following a command
terminator, which consists of two ALT MODE or ESC  characters  typed  in
consecutively.   The Editor echoes these terminator characters as dollar
signs ($$).

## Aborting Commands

A command string may be aborted while it is being  entered,  before  the
command  terminator  is  issued.  If it is necessary to delete an entire
command string, without any consideration to its  contents,  entering  a
Control/C  will  remove  the  entire  command  string and cause the Text
Editor to reprompt for a new command.

Also, after the command terminator is  entered,  and  the  commands  are
being  executed,  the  operation may be aborted by entering a Control/C.
The Text Editor will terminate its operation  and  reprompt  for  a  new
command.

## Deleting Typographic Errors

Any typographic errors in a command string or text string may be removed
by  pressing  the  RUBOUT key once for each character to be removed.  As
each character, starting from the last  one  entered,  is  deleted,  the
Editor  echoes  the  deleted  character.  Corrections of this nature can

only be performed prior to entering the command string terminator.


## Use of TAB Characters

The Editor accepts the horizontal TAB character (09H), which is
generated by a Control/I on the keyboard. This character is stored in
the text buffer as a single character. The Intellec MDS system accepts
this character to generate a sufficient number of spaces to position the
curser to the next tab position. Tab stops are located every eight
character positions across a line of text.


## Carriage Return and Line Feed Characters

Carriage return and line feed characters are saved in the text buffer
the same as any other ASCII character. However, because these two
characters are commonly used to format text listings, they are given
special significance, in the following respects:

> During an INSERT command (to be described later) entering a
> carriage return on the system console causes a line feed
> character to be generated by the Editor and appended to the
> carriage return character. Thus, the entry of a single carriage
> return character causes a pair of characters to be stored in the
> text buffer.

> During an APPEND command (to be described later), entering a
> carriage return character via punched paper tape, causes only
> the carriage return to be placed in the text buffer. Text may
> be prepared off-line, using both the carriage return and line
> feed characters to format the off line listing. The punched
> paper tape, containing both the carriage return and line feed
> characters, may then be input to the Editor, without causing
> additional line feed characters to be generated. The carriage
> return and line feed characters are accepted and stored as
> single characters.


## LOADING THE TEXT EDITOR

The Text Editor program is supplied on punched paper tape and is loaded
into the system using the loading facilities of the MDS Monitor.
Because the Editor uses the I/O facilities of the Monitor, all
peripheral devices must be defined before starting execution of the
Editor.

The following start-up procedures can be used to load and start the
Intellec MDS Text Editor:

> • If it is not already operating, start the system Monitor
> and assign the CONSOLE and READER devices to the

peripherals to be used during the loading of the Editor
program.

*   Place the tape containing the Text Editor into the
    peripheral assigned as the system READER.

*   Type in the READ HEXADECIMAL FILE command (R) on the
    system CONSOLE.

    .R0

    The paper tape will be read by the reader until an
    end-of-file record is read.  The MONITOR will prompt for
    a new command by printing a period (.) in the left-most
    column of the CONSOLE device.  If required, new device
    assignments may be made at this time to assign the
    system PUNCH and LIST devices, or to change the system
    CONSOLE or READER devices.

*   To start executing the Editor, type in the following
    command:

    .G20

    The Text Editor will assume control and print the
    following message:

    INTELLEC MDS TEXT EDITOR, - VERSION x.x
    *

    The Text Editor is now ready to accept commands on the
    system CONSOLE device.

## EDITOR COMMANDS

Editor commands are provided to perform four groups of operations,
including:  Text Input/Output, Buffer Pointer manipulation, Text
modification, and String Searches.  In the following discussion, the
order in which the commands are discussed allows the user to initially
operate the Editor with a minimum of commands.  Then, as more commands
are learned, the user can perform more complex editing operations.

## Commands B, Z, I, T

B - BEGINNING OF TEXT BUFFER

The B command is a pointer manipulation command used to move the buffer
pointer to the beginning of the text buffer to the position defined as
'Start of Buffer'.  This command is useful in several respects, for
example:

- Setting a reference point for counting lines of text;

- Defining a starting point when the whole text buffer contents are to be typed out;

- Moving the Buffer Pointer to the start of buffer prior to starting a search for a selected text string;

- Inserting text at the beginning of the text buffer, before text already in the buffer.


## Z - END OF WORKSPACE

The Z command is a pointer manipulation command used to move the buffer pointer to the end of buffer position, immediately following the last character in the buffer. This command is used mainly to position the pointer so that new text will be inserted at the end of old text and be appended to it.


## I - INSERT TEXT

The I command is a text input command used to enter text into the text buffer from the system CONSOLE device. Placement of the new text in the text buffer is dependent on the position of the Buffer Pointer.

If the buffer is empty, the buffer pointer will be positioned at the start of buffer. Thus, any text input with the I command would be placed into the buffer starting at the beginning. If the buffer already contains text and the buffer pointer is at some intermediate position in the text, the new text will be inserted immediately following the buffer pointer position, splitting the old text. If the text buffer already contains text and the buffer pointer is at the end of buffer, the new text will be inserted at the position immediately following the old text. After insertion, the pointer will be positioned after the last character in the new text.

Entering a carriage return character causes a line feed character to be generated by the Editor and appended to the carriage return character. Thus, the entry of a carriage return character causes a pair of characters to be stored in the text buffer. Note that this only occurs during the I command.

After recognizing the letter 'I' as a command, the Editor accepts all subsequent input as text (including the carriage return and appended line feed characters) until an ALT MODE, ESC, or Control/C is input. The ALT MODE or ESC character specifies the termination of the text string; the Control/C character aborts the command.


## T - TYPE OUT TEXT

The T command is an output command, used to type out lines of text.

This command uses an argument which is placed in front of the command as
follows:

        *nnnT$$                        nnn represents any decimal   number   from
                                            -65,535 to +65,534.

If the argument is positive, typing starts at the   current   location   of
the buffer pointer;   the argument value specifies the number of lines to
be typed.   If the argument is negative, typing begins   at   the   location
defined   by   the current location of the buffer pointer minus the number
of lines specified by the argument value. Typing   continues   until   the
location   of   the   buffer   pointer is reached.   If the argument value is
zero,   typing   starts   at   the   beginning   of   the   current   line.   All
characters   up   to   the   buffer   pointer   are   typed.   If   no   value is
specified, the Editor assumes a default value of 1.

If the argument value is greater   than   the   number   of   lines of   text
between   the buffer pointer and the appropriate buffer boundary, all the
specified text in the buffer will be typed.   However, the   command   will
be terminated automatically when the buffer boundary is reached.


Examples Of Editing Using B, Z, I, and T Commands

Suppose a user has entered   data   into   the   text   buffer   using   the   I
command,   and   now   wishes to type out the entire buffer.   The following
command string may be used:

        *B500T$$                      The B command moves the   buffer   pointer
                                      to   the   beginning   of   the text buffer.
                                      The 500T command types out 500 lines   of
                                      text.   The argument 500 is assumed to be
                                      larger than the number of lines of   text
                                      present   in   the buffer.   This being the
                                      case, the T command is   terminated   when
                                    the   end   of   buffer   is   reached,   even
                                    though the full count is not reached.

Suppose the user is entering a source program, using the I command,   and
has already entered a large number of text lines.   For some reason the I
command is terminated and the buffer pointer   is   moved   to   some   other
location   in   the   buffer.   When the user wishes to resume entering the
source file, the buffer pointer is simply moved to the end of buffer and
the   I   command   is   initiated.   A   typical   command   string will be as
follows:

        *ZITEXT STRING------------$$
                                      The new text will be inserted   following
                                      the old text.

If the user is entering text and wishes to see the previous   five   lines
entered, without moving the buffer pointer, the following command may be
used:

       *-5T$$                         The five lines before the current line
                                    (the one in which the buffer pointer is
                                    located) are printed on the system
                                    CONSOLE device.  The current line is not
                                    printed.

The following command may be used to print the current line of text:

       *0TT$$                         The ´0T´ part of the command prints from
                                    the beginning of the line up to the
                                    buffer pointer.  The following ´T´
                                    command prints from the buffer pointer
                                    to the end of the line.

## Commands L, K

### L - LINE

The L command is a line-oriented pointer manipulation command.  This
command uses an argument which is placed in front of the command as
follows:

       *nnnL$$                       nnn represents any decimal number from
                                    -65,535 to +65,534.

The pointer cannot be moved outside the boundaries of the text buffer.
If the user issues an L command to move the buffer pointer forward
beyond the end of buffer, or backward beyond the start of buffer, the
buffer pointer will be moved to the respective buffer boundary; the L
command will then be terminated.

The line feed character (0AH) serves as the delimiter between lines.  A
line of text is defined as having a line feed character as its last
character.

When the argument value is 1 or no argument used (default value of 1
assumed), the buffer pointer is advanced to the start of the next line.
A positive argument value advances the buffer pointer to the beginning
of the nth line following the current line (n = positive argument
value).  A negative argument value moves the buffer pointer back to the
beginning of the nth line preceding the current line (n = negative
argument value).  When the argument value is -1 or just -, the buffer
pointer is moved back to the beginning of the line preceding the current
line.  Finally, if the argument is 0, the buffer pointer is moved back
to the beginning of the current line.

### K - KILL

The K command is a line-oriented deletion command used to delete lines
of text.  This command uses an argument which is placed in front of the
command as follows:

       *nnnK$$                          nnn represents any decimal   number   from
                                           -65,535 to +65,534.

The numeric argument used with this command specifies a number and sign.
The  number  represents  the  number  of  lines to be deleted;  the sign
indicates the direction.  A negative argument deletes lines prior to the
line  containing  the buffer pointer.  A positive argument deletes lines
following the line containing the buffer pointer.  If  the  argument  is
zero, the characters from the start of the current line up to the buffer
pointer are deleted.  If the argument is 1,  the  characters  from  the
buffer  pointer,  up  to  and including the line feed character which is
used to terminate the line, are deleted.  When no argument is  included,
a default value of 1 is implied.

If the argument value is greater  than  the  number  of  lines  of  text
between  the  buffer  pointer  and  the  text buffer boundary, the lines
between the buffer pointer and boundary are  deleted.   The  command  is
terminated when the buffer pointer reaches the boundary.


Examples of Editing Using the B, Z, I, T, L, and K Commands

A user wishes to move the pointer back 5 lines of text and have the line
where the pointer is positioned typed out.  The following command string
may be used:

       *-5LT$$                       The buffer pointer is moved back to  the
                                start  of  the  fifth  line  before  the
                                current line.  This new line becomes the
                                current  line.  The T command causes the
                                line to be typed out.

If the pointer is at some intermediate position in a line and  the  user
wishes  to  return  the  pointer to the start of the line, the following
command may be used:

       *0L$$                         0L causes the buffer pointer to move  to
                                the start of the current line.

The following text is present in the text buffer:

       THIS IS LINE 1
       THIS IS LINE 2
       THIS IS LINE 3
       THIS IS LINE 4
       THIS IS LINE 5
       THIS IS LINE 6
       THIS IS LINE 7
       THIS IS LINE 8
       THIS IS LINE 9
       THIS IS LINE 10


Assume that the buffer pointer is in line 6, positioned  between  the  I

and   the   S   in   the   word   IS.    Each   of the commands in the following
examples will produce the specified type out:

       *0T$$                          Types out from the start of the  current
                                     line (line 6) up to the buffer pointer.

       THIS I*

       *T$$                            Types from the buffer pointer to the end
                                     of line.

       S LINE 6
       *

       *0TT$$                         Types the whole line without moving  the
                                     buffer pointer.

       THIS IS LINE 6
       *

       *-5T$$                         Types 5 lines preceeding the
                                     current pointer line. Types
       THIS IS LINE 1          the current line from its start
       THIS IS LINE 2          to the pointer position.
       THIS IS LINE 3
       THIS IS LINE 4
       THIS IS LINE 5
       THIS I*

       *5T$$                          Types five lines including part
                                     of current line from position
       S LINE 6                of buffer pointer. In this
       THIS IS LINE 7          case, the five lines were
       THIS IS LINE 8          typed. However, if the
       THIS IS LINE 9          command were ´6T´, the
       THIS IS LINE 10        sixth line would not be
       *                              typed because the sixth line
                                     after line 6 does not exist
                                     inside the text buffer
                                   boundaries. The five lines
                                   would be printed as in the
                                   example to the left; then the
                                   command would be terminated.

       *-5T5T$$                     Types all ten lines of the
                                     buffer. Includes the five
       THIS IS LINE 1          lines preceeding the buffer
       THIS IS LINE 2          pointer, the line containing
       THIS IS LINE 3          the buffer pointer (from the
       THIS IS LINE 4          beginning of the line up to the
       THIS IS LINE 5          pointer), the remainder of the
       THIS IS LINE 6          current line (from the pointer
       THIS IS LINE 7          to the end), and the four
       THIS IS LINE 8          remaining lines.

```
            THIS IS LINE 9
            THIS IS LINE 10
            *
```

The buffer pointer is located in line 6, between the characters I and  S
in the word IS.  The user wishes to delete lines 3,4,5,and 6.

```
            THIS IS LINE 1
            THIS IS LINE 2
            THIS IS LINE 3
            THIS IS LINE 4
            THIS IS LINE 5
            THIS IS LINE 6
            THIS IS LINE 7
            THIS IS LINE 8
            THIS IS LINE 9
            THIS IS LINE 10
```

Before this operation can be  started,  the  buffer  pointer  should  be
positioned  either  before  or  after the lines which are to be deleted.
First, let us move the pointer in front of the lines and use a  positive
argument  K command.  Because the pointer is in line 6, it must be moved
in front of line 3, then the four lines 3,4,5,and 6 can be deleted.  The
command is as follows:

```
     *-3L4K$$                    The -3L moves the pointer to
                                 the start of line 3. The
                                 4K deletes the next four lines.
```

The pointer may be moved to the line following the lines to  be  removed
and a negative argument K command used, as follows:

```
     *L-4K$$                     The L moves the pointer to
                                 the start of line 7. The
                                 -4K deletes lines 6,5,4,and 3.
```

## Commands E, W, N

The E (End), W (Write), and N (Null) commands  generate  output  on  the
system PUNCH device, as described below.  The operation of each of these
commands differs slightly, depending on which device is assigned as  the
system  PUNCH  device.   If the Teletype is assigned as the system PUNCH
device when an E, W, or N command is  executed,  the  Editor  recognizes
this and starts the punch operation with the message:

```
          START PUNCH, TYPE CHAR
```

The Editor is requesting the user to turn on the  Teletype  punch,  then
type in any character on the keyboard, to resume Editor operation.


Punching starts and continues until the specified number  of  lines  are

punched out.  The  operation then stops, allowing the operator to turn
the punch off.  To return the Editor to  a  prompt  mode,  type  in  any
character;  the Editor will reply by prompting with an asterisk.

If the high speed punch is assigned as the PUNCH device and the Teletype
or  CRT  is  assigned  as  the  CONSOLE  device,  paper tape punching is
initiated immediately, without any requests for operator intervention.


## E - EXIT

The E command is used to punch out  the  entire  contents  of  the  text
buffer  at  the  completion  of a work session.  Output is on the system
PUNCH device.  The text buffer is cleared after  punching;  the  Editor
program  is  effectively  restarted and will accept new text immediately
following the E command execution.

No leader is generated by the E command.  However,  60  NULL  characters
are  generated  at  the end of the tape to act as trailer.  If leader is
needed, the N command (described below) can be  used.

After the buffer contents are  punched  on  tape,  the  text  buffer  is
cleared,  and  the remaining input (on the READER device) is read in and
copied directly to  the  PUNCH  device.   The  Editor  then  punches  an
End-of-File  mark  (Control/Z)  into  the  tape.   If  this  tape  is
subsequently input by  an  A  command  (to  be  described  later),  this
character  will be recognized by the Editor as the end of the tape.  The
editor is reinitialized, and the following message is printed:

           INTELLEC MDS TEXT EDITOR, - VERSION x.x

If the text buffer is empty, the E command may be used to  copy  punched
paper  tape.   The  procedure  to  perform tape copying operations is as
follows:

  •          While operating the Monitor, assign the READER and PUNCH
             devices required to perform the tape copying operation.

  •          Start the Text Editor as described in this section.

  •          Place the tape to be copied into the READER.

  •          Make sure that both the READER and PUNCH are turned  on,
             then type in the following command:

             *E$$


## W - WRITE

The W command is an output command used to punch out a specified  number
of lines from the text buffer onto the system PUNCH device.  The text is
always taken from the beginning of the text buffer,  regardless  of  the
current  position  of  the buffer pointer.  As text is punched, the line

punched is deleted from the text buffer, and the remaining text is compacted up to the start of text. This command uses an argument which is placed in front of the command as follows:

   *nnnW$$       nnn represents any decimal number from
                 -65,535 to +65,534. Both positive and
                 negative arguments are treated as
                 positive values. An argument value of
                 zero causes no punching to occur.

## N - PUNCH 60 NULL CHARACTERS

The N command is an output command used to punch leader and trailer into punched paper tape. Each ´N´ entered into the command punches 60 NULL characters.

## Examples Of Editing Using The E, W, And N Commands

Assume a text buffer with 100 lines of text. The first 25 lines are satisfactory and require no further editing. The user selects to output these lines of text to punched paper tape before continuing with the editing. The command will appear as follows:

   *25W$$

The first 25 lines in the text buffer are punched on the system PUNCH device. The pointer is moved to the start of buffer; line 26 becomes line 1 and all of the subsequent lines of text are moved up a corresponding amount.

If the user requires leader or trailer on the punched paper tape, it should be produced with the ´N´ command or the manual facilities of the PUNCH device. No leader or trailer is generated with the ´W´ command.

If a text file is being inserted and, for some reason, the user finds it necessary to terminate the editing session before the file is complete, the integrity of the text file can be maintained by punching out the contents of the text buffer and reloading it when editing is resumed. The ´E´ command and ´N´ command can be used, as follows:

   *NNE$$       Punches out 120 NULL characters, then
                punches the contents of the whole text
                buffer.

This procedure is handy when the user has to stop an editing job temporarily and wishes to save the current contents of the text buffer for subsequent continuation of the job.

A condition may occur where the text buffer part of reserved memory is

almost  full  and the command string is attempting to overwrite the last
character in the text string.  This condition is trapped  and  the  last
character  of  the  command  string  is  rejected.   When this condition
occurs, the only character that will be accepted is the RUBOUT.

The user must type in a sufficient number of RUBOUT characters  so  that
enough  buffer is made available to input a command terminator.  At this
point, the user must clear out part of  the  text  buffer  so  that  the
remainder  of  the  input  text, or part of it, may be input to the text
buffer.  The most expedient  manner  to  empty  the  text  buffer  while
maintaining  the  integrity of the text is to punch out the text that is
already edited.  The 'W' command is best suited to this  task.   Suppose
that  the  first  500  lines  of text are edited and can be output.  The
following command string will output 500 lines of text:

        *NN500WNN$$             This command string will punch 120  NULL
                                      characters, punch out 500 lines of text,
                                      relocate the remaining text to the front
                                      of  the  text buffer, and then punch 120
                                      NULL characters as trailer.

## Command A

A - APPEND

The A command is used to enter text into the text buffer from the system
READER  device.   The  input text is appended to the text already in the
buffer, with the new text being stored at the end of the  buffer.   Once
initiated,  the  A  command  continues  reading  text  until  one of the
following conditions is satisfied:

- The end of tape is reached.

- An  End-of-File  character  is  read  (Control/Z).   The
  Control/Z is not placed into the text buffer.

- The workspace is full.

- A Form Feed character is  read  (Control/L).   The  Form
  Feed is placed into the text buffer.

- 50 lines of text are read.

If needed to enter a large text file, the A command may be  repeated  as
many  times  as  needed  to  enter  the  entire file.  Each A command is
terminated when one of the above conditions is satisfied.   If the READER
is  turned off during an A command, the system recognizes this as an End
Of Tape and terminates the command.

Examples Of Editing Using The A Command

Suppose that, in the last example, the tape was to be reentered and the
editing job continued. Also, suppose that 150 lines of text are saved
(punched). The following procedure will reenter the saved text into the
text buffer:

- Start the Editor

- Place the tape into the tape READER.

- Type in the following command,

    *AAA$$

The tape READER will start and 150 lines of text will be read. The text
may now be edited, or new text appended.


Commands C, D

C - CHARACTER

The C command is a pointer manipulation command used to move the pointer
a specified number of character positions. An argument is used with
this command and is placed in front of the command as follows:

    *nnnC$$                          nnn represents any decimal number from
                                     -65,535 to +65,534.

The pointer cannot be moved beyond the boundaries of the text buffer.
Thus, if the buffer contains 5,000 characters, the command "7000C" will
move the buffer pointer 5,000 characters, until the buffer pointer
reaches the boundary, then the command will be terminated. Similarly, a
command to move the buffer pointer in a negative direction, past the
start of buffer, will be terminated once the start of buffer boundary is
reached.

Normally, the C command is not the best way to move the pointer over
large distances. This command is best utilized when pointer movement,
on a character basis, is restricted to one line of text. Larger pointer
movements are more easily performed by using the ´L´ or ´F´ (to be
described later) commands.


D - DELETE CHARACTER

The D command is used to delete a specified number of characters from
the text. The numeric argument used with this command specifies a
number and sign. The number represents the number of characters to be
deleted; the sign indicates in which direction, moving from the buffer
pointer position, the deletion is to occur. A negative argument deletes
characters in front of the buffer pointer; a positive argument deletes
characters following the buffer pointer. The format of the D command is

as follows:

       *nnnD$$                               nnn represents any decimal number from
                                            -65,535 to +65,534.

An argument with a value larger than the number of characters between
the buffer pointer and either boundary of the buffer cannot be used to
move the delete operation out of the text buffer. The delete operation
will continue until the end of buffer or beginning of buffer (when the
argument is negative), at which time the command will be terminated.


Examples Of Editing Using The C and D Commands

A user wishes to delete a character from a line of text. The buffer
pointer is at the beginning of the line. Consider the following line of
text where the word MULTUIPLY can be corrected by simply deleting the
extraneous letter U.

      ; THIS ROUTINE WILL MULTUIPLY TWO 16-BIT NUMBERS

Because the buffer pointer is at the beginning of the text line, it must
be moved 23 character positions to the point immediately before the
letter U; whereupon the letter U may be simply deleted with the D
command. The command to perform these operations is as follows:

     *23CD$$

This is a clumsy way to position the buffer pointer and delete a
character. It is included here as an example to show use of the C
command. A better way to perform the same operation would be initiated
with the SUBSTITUTE ´S´ command (to be described later).

If no argument is used with the C or D commands, the default value of 1
is assumed. If a value of 0 is used, the pointer does not move and the
command has no effect.


Commands F, S

F - FIND TEXT STRING

The F command is a search command used to find a text string of up to
sixteen characters. The characters ALT MODE, ESC, and Control/C are not
considered to be text because of their control functions, and cannot be
included in the set of text characters.

The F command causes the Editor to search for the first occurrence of a
character string matching the character string specified in the command
All characters must match, including printing and non-printing
characters. The search is started at the current location of the buffer
pointer and continues until either the end of buffer is reached or a
successful match is made. If a successful match is made, the Editor
terminates the command leaving the buffer pointer at a location

immediately following the last character in the search string.  A prompt
character is output, requesting the next command.  If no match is  found
when the end of the buffer is reached, the Editor prints the message:

         CANNOT FIND "xxxxxxxxxx"    where xxxxxxxxxx represents
              *BREAK*                the specified string.

The buffer pointer, in this case, is left at the end of the buffer.   If
the  specified  string is larger than sixteen characters, only the first
sixteen will be used.

The format of the ´F´ command is as follows:

         *FTHIS IS THE STRING$$

If the command is to be part of a  command  string,  a  single  $  (ESC)
terminator character will terminate the text string, allowing additional
commands to be appended.  If no other commands are to be  included,  the
text  string  and  command string can both be terminated with the double
$$.   It is important to remember to terminate  the  text  string  before
additional  commands  are  appended.  Otherwise, the additional commands
will be treated as part of the text string.  For example, the string:

         *FDIVIDE0LT$$

initiates a search for the string ´DIVIDE0LT´, instead of  the  intended
string  ´DIVIDE´.   The appended command 0LT is part of the text string.
The correct format for this command is as follows:

         *FDIVIDE$0LT$$

Note that the text string DIVIDE is itself terminated with a single $.

It is wise to verify the match of the requested  text  string  with  the
text  string  found.   In  some  cases  a  string  may appear in several
unexpected places prior to the line being searched for.  For example, if
the  label  DIV:   is  being searched for and several occurrences of the
string DIV are present (i.e.  DIVIDE, DIV1, DIV2, DIV3, etc)  specifying
DIV  as  the  search  string  will  produce  spurious results. A unique
combination of characters are required;  in  this  case,  it  would  be
better  to  search for DIV: (include the colon), and then verify search
results by typing out the line.


S - SUBSTITUTE TEXT STRING

The S command is a search command used to find a text string and replace
it  with  another  text  string.   The  search  part of the S command is
similar to the F command.  The substitute part of this command occurs if
the  search  is successful.  Any number of characters may be substituted
for the characters in the search string.  At the  completion  of  the  S
command,  the  buffer pointer will be placed after the last character in
the replacement string.  The format of the S command is as follows:

          *SSTRING 1$STRING 2$$

Each of the strings must be terminated with a $. The first string
STRING 1 is the search string, which must be limited to sixteen
characters. STRING 2 is the substitution string and may contain any
number of characters (excluding the characters ALT MODE, ESC, and
Control/C). The substitute string must be terminated with an ESC or ALT
MODE.

If no substitution string is included, the search string will be found
and deleted. The S command is used in this manner to selectively delete
strings up to 16 characters long.


Examples Of Editing Using The F And S Commands

There are three extraneous characters which are to be deleted with the D
command. The line of text appears as follows:

          PARAM:  CALL    BACKOFF ;  BACK IS THE RETN

The user would like to delete the three characters OFF from the word
BACKOFF. First the pointer must be positioned adjacent to the string
OFF, and then the deletion performed. The command will appear as
follows:

          *BFBACK$3D$0TT$$          This command string moves the pointer to
                                    the start of buffer, then commences a
                                    search for the string BACK. At the
                                    first occurrence of this string, the
                                    pointer is positioned following the K in
                                    BACK. The next three characters are
                                    deleted (OFF), and finally the line is
                                    typed out. The $$ terminates the
                                    command string.

When completed, the line of text will appear as follows:

          PARAM:  CALL    BACK ;  BACK IS THE RETN

If for some reason the pointer is positioned immeditaely after the
string of charcters which are to be deleted, the D command may be used
with a negative argument. Consider the following example. A user
wishes to delete the label PARAM:

          PARAM:  CALL    SUB1

The command would appear as follows:

          *FPARAM:$-6D$$           The string PARAM: would be searched
                                    for, and when found, the pointer will be
                                    positioned following the colon (:).  By
                                    deleting the preceding six characters

the six character string PARAM: is
deleted.

An error to be corrected consists of a misspelled word. The following
command is used to search for the incorrect word and replace it with the
correct one. Once corrected, a typeout is specified to verify the
operation.

        *SINITAIL$INITIAL$0TT$$

The Editor responds by performing the substitution and typing out the
corrected line. At the termination of the operation, the buffer pointer
is positioned at the location between the L in INITIAL and the following
blank character. The corrected line is typed out as follows:

            LXI     H,0      ;   INITIAL VALUE FOR REMAINDER

If a carriage return occurs in the search string of an F or S command,
or in the replacement string of an S command, the Text Editor will
automatically generate a line feed following it. Thus, the command

        *FEND.
        NEXT$$

will search for the characters "END.<cr><lf>NEXT".


Command Iterations

A command or command string may be repeated any number of times by
enclosing the string in angle brackets "<" and ">", preceded by a number
which specifies the number of times the iteration is to be performed.
The format of the command is as follows:

        n<command or command string>$$
                            where n specifies the number of times
                            the command enclosed between the symbols
                            ´<´ and ´>´ is executed.


For example, if a user program is written using a label ´DIVID´ which is
included fewer than ten times in the source file and wishes to shorten
the label to DIV, an iterative substitute (S command) would be used as
follows:

        *B10<SDIVID$DIV$>$$        The B command moves the pointer to the
                                   beginning of the text buffer. The
                                   iterative command is repeated ten times;
                                   it searches for the text string DIVID,
                                   and each time it finds the string,
                                   replaces it with the new string DIV.

Command iterations may be nested up to eight deep. Any attempt to nest

command  iterations more than eight deep is trapped and an error message
is printed on the console device:

        ITERATION STACK FAULT

For example, suppose the text buffer contains the SINEs of angles from 0
degrees  to  90  degrees  in increments of one degree.  Assume these are
arranged one per line and each is listed to 15  digits  of  accuracy  as
follows:

        0.000000000000000
        0.017452406437284
        0.034899496702501
        0.052335956242944
                "
                "
        0.999390827019096
        0.999847695156391
        1.000000000000000

To improve the readability, it would be helpful to break each number  up
into  groups  of five digits by inserting spaces, so that each one would
appear as follows:

        0.00000 00000 00000
        0.01745 24064 37284
        0.03489 94967 02501
              "
              "
        0.99939 08270 19096
        0.99984 76951 56391
        1.00000 00000 00000

This can be done by nesting  two  iterated  commands:  One  level  will
insert  a  space  at  every  fifth character on each line, and the other
level will advance through the 91 lines in the file.
The command is:

        *B91<2C2<5CI $>L>$$


TEXT EDITOR MESSAGES

The Intellec® MDS Text Editor  prints  messages  on  the  system  CONSOLE
device to notify the user of various status conditions.

        There are three error messages as follows:

        "n" ILLEGAL IN THIS CONTEXT

                The "n" represents the  illegal  alphanumeric  character
                that was incorrectly typed in.

        CANNOT FIND "xxxxxxx"

The xxxxx represents the text string which the editor
could not find during an F or S command. This message
further prints *BREAK* to signal that the command is
aborted.

ITERATION STACK FAULT

This message is a notification to the user that iterated
commands were nested more than eight levels deep. The
command is aborted.


A start-up message:

INTELLEC MDS TEXT EDITOR, - VERSION x.x


A device status message:

START PUNCH, TYPE CHARACTER

This message is device sensitive, appearing only when a
punched paper tape is output with an 'E', 'W', or 'N'
command, while the Teletype is being used as the system
PUNCH device. The system waits for the requested
operation.


## USE OF EDITOR TO CORRECT SAMPLE PROGRAM

The following consists of a sample program that needs correction, the
Editor commands and responses while making corrections, and a final
output showing the corrected listing.

```
;                                    ; BC = DIVIDEND/QUOTIENT
; REENTRANT DIVIDE ROUTINE           ; HL = TEMPORARY
;                                    ; DE = DIVISOR/REMAINDER
DIV:                                 ;
        MOV     A,D                  ; NEGATE THE DIVISOR
        CMA
        MOV     D,A
        MOV     A,E
        CMA
        MOV     E,A
        INX     D
        LXI     H,0                  ; INITAIL VALUE FOR REMAINDER      sp.
        MVI     A,17                 ; INITIALIZE LOOP COUNTER
DIV0:
        PUSH    H                    ; SAVE REMAINDER
        DAD     D                    ; SUBTRACT DIVISOR (ADD NEGATIVE)
        JNC     DIV1                 ; UNDER FLOW, RESTORE HL
        XTHL
DIV1:
        POP     H
        PUSH    PCW                  ; SAVE LOOP COUNTER    PSW
        MOV     A,C                  ; 4 REGISTER LEFT SHIFT
        RAL                          ; WITH CARRY
        MOV     C,A                  ; CY -> C -> B -> L -> H
        MOV     A,B
        RAL
        MOV     B,A
        MOV     A,L
        RAL
        MOV     L,A
        MOV     A,H
        RAL
        MOV     H,A
        POP     PCW                  ; RESTORE LOOP COUNTER
        DCR     A                    ; DECREMENT IT
        JNZ     DIV0                 ; KEEP LOOPING
DIV2:                                ; POST-DIVIDE CLEAN UP
        ORA     A                    ; SHIFT REMAINDER RIGHT AND RETURN IN DE
        MOV     A,H
        MOV     D,A
        MOV     A,L
        MOV     E,A
        RET
        END
```

Handwritten annotations:
- Change to "DV0" and "DV1"
- INITAIL — sp.
- PCW → PSW
- PCW (circled)
- RAR → (MOV A,H)
- RAR → (MOV A,L)
- Move over and put blank lines around them

Figure 4-1: Sample Program Before Editing

The three comments may be entered into the text with the following command string:

```
*3LI; BC = DIVIDEND/QUOTIENT
; HL = TEMPORARY
; DE = DIVISOR/REMAINDER
;
$$
```

The next error to be corrected consists of a misspelled word. The following command is used to search for the incorrect word and replace it with the correct one. Once corrected, a typeout is specified to verify the operation

```
*SINITAIL$INITIAL$0TT$$
```

The Editor responds by performing the substitution and typing out the corrected line. At the termination of the operation, the buffer pointer is positioned at the location between the L in INITIAL and the following blank character. The corrected line is typed out as follows:

```
         LXI      H,0      ;   INITIAL VALUE FOR REMAINDER
```

The next task is to find all occurrences of the label DIV0 and replace it with the string DV0. The number 10 is selected because it is known that there are fewer than 10 occurrences of the label DIV0. As each occurrence is matched during search, it is replaced and the corrected line is typed out. The command appears as follows:

```
*10<SDIV0$DV0$0TT>$$
```

The editor will reply with the following:

```
DV0:
         JNZ      DV0      ; KEEP LOOPING
```

```
CANNOT FIND "DIV0"
```

```
*BREAK*
```

Similarly, we can replace ´DIV1´ with ´DV1´ throughout.

Next, the register named PSW is incorrectly called PCW in the source coding. A search will be made for the text string PCW, and each time it is found it will be replaced with the text string PSW. The command is as follows:

```
*B10<SPCW$PSW$0TT>$$
```

The operation is similar to that described above. The Editor output is as follows:

```
         PUSH     PSW      ; SAVE LOOP COUNTER
         POP      PSW      ; RESTORE LOOP COUNTER
```

            CANNOT FIND "PCW"

            *BREAK*


We have found all occurrences of DIV0, DIV1, and PCW and corrected them.
The buffer pointer is at the end of the buffer. The next task is to
delete the label DIV2 (which is not referenced in any of the statements)
and move the comments over to the left-hand margin. Also, comment lines
will be generated to precede and follow the "; POST DIVIDE CLEAN UP"
message.   In  addition, the word "POST DIVIDE" will be hyphenated.  The
command string will appear as follows:

            *BFKEEP LOOPING$LI;
            $9DFPOST$DI-$LI; SHIFT REMAINDER RIGHT AND RETURN IN DE
            ;
            $-4T$$


The first command, B, returns the pointer to the start  of  the  buffer.
Although  text  string  "KEEP  LOOPING"  is  on the previous line, it is
convenient to use F to locate the required line.

The next command, L, moves the buffer pointer to the next line where the
semicolon  and  carriage return/line feed are inserted. Nine characters
are deleted, removing the label and  four  TAB  characters,  moving  the
comment  to  the  left-hand margin.  The string POST is searched for and
quickly found because it is immediately  following  the  pointer.   This
positions  the  pointer  after  the  letter T in POST.  One character is
deleted and replaced with the dash (-) character.  The L  command  moves
the  pointer  to  the  next line, where another comment line, a carriage
return/line feed, a semicolon, and  another  carriage  return/line  feed
combination  are  inserted.   The  -4T command  is  used  to verify the
operation.  The Editor prints the following:


            ;
            ; POST-DIVIDE CLEAN UP
            ; SHIFT REMAINDER RIGHT AND RETURN IN DE
            ;


The next task is to insert two "RAR" commands.  The  command  to  locate
the line and insert the first RAR is as follows:

            *2<FMOV$>0LI       RAR
            $-2T$$

From the current pointer  position,  following  the  last  operation,  a
search  is  made  for two occurrences of the text string MOV. Note that
this command would produce unpredictable results  if  started  when  the
position  of  the  pointer  is  unknown.   However, remembering the last
command string and determining that the pointer is left  at  a  position
between  the  line  feed  from the line containing the ";" and the <tab>
character preceeding the O in ORA, the position  of  the  next  two  MOV

strings is known.

After the second MOV is located, the pointer is positioned following the
V in the second MOV. The pointer is then moved to the start of the
current line and a line is inserted, containing <tab>, RAR, <cr,lf>.
The -2T command is used to look at the previous line and the current
line to verify the insertion. The Editor prints out the following:

```
            MOV       A,H
            RAR
```

In the last command string, the pointer is moved forward three lines,
then the string <tab>, RAR, <cr,lf> is inserted. The current line and
the previous line are output. The command appears as follows:

```
      *3LI      RAR
      $-2T$$
```

The Editor replies by printing out:

```
            MOV       A,L
            RAR
```

The program is now complete. A corrected version of the text appears in
Figure 4-2.

```
;
; REENTRANT DIVIDE ROUTINE
;
; BC = DIVIDEND/QUOTIENT
; HL = TEMPORARY
; DE = DIVISOR/REMAINDER
;
DIV:
        MOV     A,D                     ; NEGATE THE DIVISOR
        CMA
        MOV     D,A
        MOV     A,E
        CMA
        MOV     E,A
        INX     D
        LXI     H,0                     ; INITIAL VALUE FOR REMAINDER
        MVI     A,17                    ; INITIALIZE LOOP COUNTER
DV0:
        PUSH    H                       ; SAVE REMAINDER
        DAD     D                       ; SUBTRACT DIVISOR (ADD NEGATIVE)
        JNC     DV1                     ; UNDER FLOW, RESTORE HL
        XTHL
DV1:
        POP     H
        PUSH    PSW                     ; SAVE LOOP COUNTER
        MOV     A,C                     ; 4 REGISTER LEFT SHIFT
        RAL                             ; WITH CARRY
        MOV     C,A                     ; CY -> C -> B -> L -> H
        MOV     A,B
        RAL
        MOV     B,A
        MOV     A,L
        RAL
        MOV     L,A
        MOV     A,H
        RAL
        MOV     H,A
        POP     PSW                     ; RESTORE LOOP COUNTER
        DCR     A                       ; DECREMENT IT
        JNZ     DV0                     ; KEEP LOOPING
;
; POST-DIVIDE CLEAN UP
; SHIFT REMAINDER RIGHT AND RETURN IN DE
;
        ORA     A
        MOV     A,H
        RAR
        MOV     D,A
        MOV     A,L
        RAR
        MOV     E,A
        RET
        END
```

Figure 4-2: Sample Program After Editing

SECTION 5

MDS ASSEMBLER

INTRODUCTION

The Intellec MDS Assembler accepts 8080 source language statements and, in two or three passes, depending on the peripheral devices available, generates a listing (including symbol table) and a hexadecimal object file on punched paper tape. A full description of assembly language syntax and semantics is included in the 8080 Assembly Language Programming Manual, Intel publication # 98-004.

In use, the assembler is loaded into the Intellec MDS system using the hexadecimal loading facilities of the system Monitor. When initially loaded and started, the Assembler prints an introductory header message. The operator now specifies which pass of the Assembler is to be executed. The source tape of the program being assembled is read during each pass. After each pass the tape must be rewound and prepared for the next pass. The assembler outputs a list file and a hexadecimal object file.

The user's reply to the prompt question depends on the system I/O format, specifically on the peripheral devices available for system I/O. This is discussed below.

LOADING THE ASSEMBLER

The assembler is supplied on punched paper tape and is loaded into the system using the tape loading facilities of the system Monitor. Because the assembler uses the I/O facilities of the monitor, all peripheral devices must be defined before starting execution of the assembler. The following procedures load and start execution of the Intellec MDS Assembler:

1        If the Monitor is not already operating, start the system as described in the "START UP" procedures.

2        With the Monitor running, assign (using the Monitor    "A"

command)    the  peripherals  required  for  loading  the
Assembler, as the CONSOLE and READER devices.

3          Place the tape containing the  MDS  Assembler  into  the
           peripheral assigned as the system READER device.

4          Type in the R command as follows:

                .RØ

           The paper tape will be  read  by  the  READER  until  an
           End-Of-File record is read.  The Monitor will prompt for
           a new command by printing a period (.) in the  left-most
           column  of  the system CONSOLE device.  If required, new
           device assignments may be made at this time,  to  assign
           the  system  PUNCH  and  LIST  devices, or to change the
           system CONSOLE or READER devices.

5          To start executing the Assembler, type in the  following
           command:

                .G2Ø

           The  Assembler  will  assume  control  and  print  the
           following introductory message:

                8Ø8Ø MDS MACRO ASSEMBLER VERSION x.x
                P=

The Assembler is now ready to read a source program from  punched  paper
tape.   If  the user has a single LIST/PUNCH device, such as a Teletype,
the assembler requires three passes to produce a listing and hexadecimal
object  code  output.   A  user with separate punch and list devices may
generate the list and object file outputs in two passes.  The  user  has
control  of  assembler operations by replying to the "P=" prompt with an
appropriate reply (by typing in the numerals 1, 2, 3, or 4).

It should be noted that a user with separate list and punch  devices  is
not limited to assembling a source program in two passes.   However, the
user with a single list/punch device will  produce  an  unusable  object
output file if an assembly operation is performed in two passes.

The significance of the user's replies to "P=" are as follows:

     P=1       Reads punched paper tape source file and sets up  symbol
               table for the subsequent passes.  This reply is normally
               used for the first pass of an assembly.  It must be  run
               before any of the other passes can be run.

     P=2       Reads source file again  (user  must  first  rewind  the
               source tape after pass 1) and generates a listing on the
               LIST device.

> P=3      Reads source file again (user must first rewind the
>          source tape after pass 1 or pass 2) and generates a
>          hexadecimal object file on the PUNCH device.
>
> P=4      Reads the source file again (user must first rewind the
>          source tape after pass 1) and generates a listing on the
>          LIST device concurrently with a hexadecimal object file
>          on the PUNCH device.  If the LIST and PUNCH devices are
>          combined into a common unit, an unusable object file
>          will be generated.

Once the operator has run pass 1, he may run passes 2, 3, or 4 in any
order, executing each pass any number of times.  This is useful when
multiple listings and/or object tapes are needed.

Twelve inches of tape leader and trailer are generated automatically
each time a hexadecimal object file is punched on paper tape.
Additional leader and trailer may be produced by using the manual
facilities of the PUNCH device.  When using a Teletype, the procedure is
as follows:

> 1        Switch Teletype to LOCAL mode.
>
> 2        Turn on the PUNCH.
>
> 3        Press HERE IS.  Approximately 15 feed characters are
>          generated each time the HERE IS key is pressed.
>
> 4        Turn off the PUNCH.
>
> 5        Switch Teletype to LINE mode and continue.

If using a tape PUNCH device, press the FEED (or equivalent) switch to
generate tape with sprocket holes only.

Facilities are provided with both the MDS Monitor and MDS Editor to
punch NULL characters as leader or trailer.  Refer to the appropriate
paragraphs discussing the "N" command.  Leader prepared in this way must
be done prior to entering the assembly program execution.  Trailer may
be prepared after returning control to the Monitor.


ASSEMBLING A PROGRAM

The following procedure is suggested:

> 1        Prepare a source file using the Intellec® MDS Text
>          Editor.  This file should be on punched paper tape.  The
>          file format required is described below.
>
> 2        Place the paper tape into the system READER device.
>
> 3        In reply to the prompt, "P=", type in the number 1.  The

tape will be read in and a symbol table generated internally.

4       Rewind the tape.

5       Depending on the peripherals available and on the job requirements, type in 2, 3, or 4, in reply to the P= prompt. The tape will be read in a second time. As the tape is read, the listing is generated (P=2) or an object file is generated (P=3), or both a listing and object file are generated concurrently (P=4), with separate LIST and PUNCH devices.

6       Rewind the source tape and return to step 5 until all of the required operations are performed.

7       To terminate the assembly, press the INTERRUPT 0 switch on the front panel.


## Input File Format

The input file consists of lines of assembly language statements. A line must be terminated with a carriage return/line feed character combination, or a carriage return/form feed character combination; each line can be at most 72 characters long. A complete description of the 8080 Assembly Language may be found in the 8080 Assembly Language Programming Manual.


## List Output

The list file is a formatted file created by the assembler. The data is designed to be output to a printer. The significance of each column in the listing is discussed in Table 5-1 below, with a sample listing shown in Figure 5-2.

COLUMNS  DESCRIPTION

1        ASSEMBLER ERROR CODE.  If the assembler encountered a syntax
         error in this source line, the appropriate error code (see
         Appendix C) will appear in this column.  Otherwise, this column
         will be blank.

2        Blank.

3-6      CURRENT VALUE OF THE PROGRAM LOCATION COUNTER.  The address
         assigned to the first byte of the object code shown in columns
         8-9 of this line is printed in hexadecimal.  In addition, the
         result of the value-generating pseudo-ops ORG, EQU, and SET will
         appear in this field.

7        Blank.

8-9      FIRST BYTE OF OBJECT CODE.  The first byte of object code
         produced by the assembler for this source line is printed here
         in hexadecimal.  If this source statement produces no object
         code (e.g., is a comment, or a pseudo-op), this field will be
         blank.

10-11    SECOND BYTE OF OBJECT CODE.  This field will be blank if the
         source statement generates no object code (comments and
         pseudo-ops), or generates only one byte of object code.  Again,
         this field is printed in hexadecimal.

12-13    THIRD BYTE OF OBJECT CODE, if this statement produced three
         bytes of code;  otherwise blank.

14-15    FOURTH BYTE OF OBJECT CODE, if generated;  otherwise blank.

16       MACRO EXPANSION FLAG.  A "+" in this column indicates that this
         source line was produced as a result of a macro expansion.
         Otherwise, this column will be blank.

17-...   LISTING OF THE ASSEMBLER SOURCE TEXT.  This field terminates at
         column 72 for all output devices other than the line printer,
         and at column 120 for the line printer.


                    Table 5-1: Assembler List File Format

8080 MDS MACRO ASSEMBLER VERSION 1.0                                PAGE 1


```
                          ;
                          ; REENTRANT DIVIDE ROUTINE
                          ;
                          ; BC = DIVIDEND/QUOTIENT
                          ; HL = TEMPORARY
                          ; DE = DIVISOR/REMAINDER
                          ;
   0000          DIV:
   0000 7A               MOV    A,D          ; NEGATE THE DIVISOR
   0001 2F               CMA
   0002 57               MOV    D,A
   0003 7B               MOV    A,E
   0004 2F               CMA
   0005 5F               MOV    E,A
   0006 13               INX    D
   0007 210000           LXI    H,0          ; INITIAL VALUE FOR REMA
   000A 3E11             MVI    A,17         ; INITIALIZE LOOP COUNTE
   000C          DV0:
   000C E5               PUSH   H            ; SAVE REMAINDER
   000D 19               DAD    D            ; SUBTRACT DIVISOR (ADD
   000E D21200           JNC    DV1          ; UNDER FLOW, RESTORE HL
   0011 E3               XTHL
   0012          DV1:
   0012 E1               POP    H
   0013 F5               PUSH   PSW          ; SAVE LOOP COUNTER
   0014 79               MOV    A,C          ; 4 REGISTER LEFT SHIFT
   0015 17               RAL                 ; WITH CARRY
   0016 4F               MOV    C,A          ; CY -> C -> B -> L -> H
   0017 78               MOV    A,B
   0018 17               RAL
   0019 47               MOV    B,A
   001A 7D               MOV    A,L
   001B 17               RAL
   001C 6F               MOV    L,A
   001D 7C               MOV    A,H
   001E 17               RAL
   001F 67               MOV    H,A
   0020 F1               POP    PSW          ; RESTORE LOOP COUNTER
   0021 3D               DCR    A            ; DECREMENT IT
   0022 C20C00           JNZ    DV0          ; KEEP LOOPING

                          ;
                          ; POST-DIVIDE CLEAN UP
                          ; SHIFT REMAINDER RIGHT AND RETURN IN DE
                          ;
   0025 B7               ORA    A
   0026 7C               MOV    A,H
   0027 1F               RAR
   0028 57               MOV    D,A
   0029 7D               MOV    A,L
   002A 1F               RAR
   002B 5F               MOV    E,A
   002C C9               RET
                         END
```

DIV     0000        DV0     000C        DV1     0012

Figure 5-2: Sample Assembler List Output

## Object Code Output Format

The object code output tape contains the contents of program memory which result from loading the assembled source program. The code is formatted in hexadecimal records. The punched paper tape contains the ASCII representation of the hexadecimal bytes of data.

The format of the object code is a series of records, each containing its record length, type, memory load address, checksum, and data. Figure 5-3 shows a typical output file in hexadecimal format. A complete description of hexadecimal object file format is given in Appendix D.

Twelve inches of paper tape leader, consisting of feed holes only (NULL characters) is generated in front of the hexadecimal object file. Similarly, twelve inches of trailer are generated immediately following the last hexadecimal record in the output file.

```
    ┌─ Record Mark
    │ ┌──── Record Length
    │ │ ┌────── Starting Load Address
    │ │ │ ┌────── Record Type
    │ │ │ │                    ┌─ Data
    │ │ │ │                    │                     ┌─Checksum
    │ │ │ │                    │                     │
:10000000 7A2F577B2F5F132100003E11E519D21282
:10001000 00E3E1F579174F7817477D176F7C176775
:0D002000 F13DC20C00B77C1F577D1F5FC96A
:00000001
```
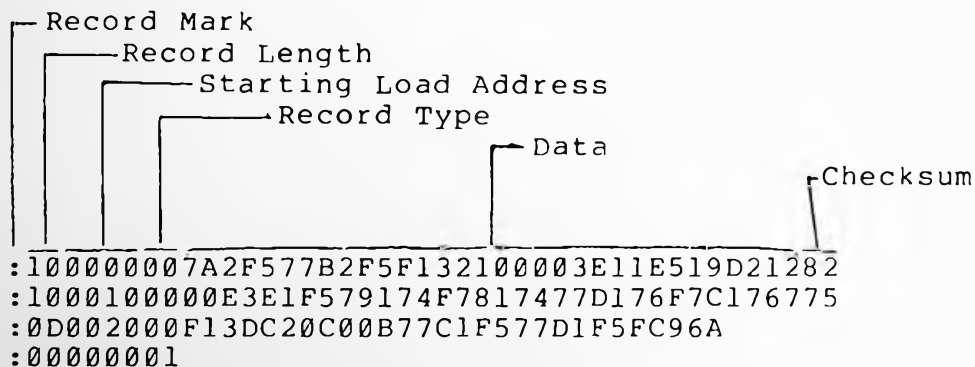
        Because Record Length = 0 and Record Type = 01,
        this record specifies End-of-File.


                    Figure 5-3: Sample Object File


## Object Code Output For PROM Programming

No special output code is generated with the Intellec MDS Assembler to program PROMs or ROMs. Control of the operations of the Universal PROM Programmer is performed with the Intellec MDS Monitor, which accepts hexadecimal object files as input.

The Monitor is also able to load a hexadecimal file and dump the memory contents in BNPF format. This facility is useful when a BNPF punched paper tape is needed to produce ROMs on another system which uses BNPF tapes.

## ASSEMBLER ERROR MESSAGES

Errors detected by the Assembler are indicated by single letter codes in column 1 on the output listing. When multiple errors are detected in a single source statement, the first error determines the error code listed. The following is a list of the Assembler error codes. A full discussion is included in Appendix C.

| CODE | NAME |
|------|------|
| B | Balance Error |
| E | Expression Error |
| F | Format Error |
| I | Illegal Character |
| M | Multiple Definition |
| N | Nesting Error |
| P | Phase Error |
| Q | Questionable Syntax |
| R | Register Error |
| S | Stack Overflow |
| T | Table Overflow |
| U | Undefined Identifier |
| V | Illegal Value |

Table 5-4: Assembler Error Codes

SECTION 6

USE OF MONITOR´S I/O SYSTEM

## INTRODUCTION

This section describes the use of the Intellec® MDS Monitor I/O facilities. This information will enable a programmer to use the I/O drivers already included in the Monitor coding to perform I/O operations in the user routines. Also, information is included to allow a programmer to write special device-specific drivers and link these to the system I/O facilities so that the monitor may have access to these drivers.

In some cases it may be better to write a user driver for a special peripheral and access the driver directly from the user program. However, by providing linkages from the monitor to the user driver and accessing the driver through the monitor, additional versatility is achieved because of several factors which include:

- Common access point to driver for all user and monitor calls.

- Availability of driver to other programs such as MDS Editor and Assembler.

- Device selection can be controlled with the Monitor ´A´ (ASSIGN) command.

## USING THE I/O SYSTEM

The Intellec® MDS I/O system contains drivers for several peripheral devices, including a Teletype console/reader/punch, a high speed paper tape reader, a high speed paper tape punch, a CRT console, and a line printer. Access to any of these devices can be made through the Monitor I/O system. A user can program his routine to access any of the four logical devices, to perform the I/O function, and, before executing the routine, assign (using the Monitor ´A´ command) the required physical device. The user may also assign a device dynamically using the Monitor

´IOSET´ call (described in the following paragraphs).


## Accessible I/O Routines

The user may access the Monitor I/O system from his programs (as do other Intellec MDS system programs) by calling the routine provided by the Monitor to perform the desired function.

The calling sequence for each system I/O routine consists of a subroutine jump to a reserved address. Each system I/O routine has a unique starting address. These fixed addresses are listed in Appendix E. Samples of driver access are included in the examples below.

Parameter passing from the user routine to the system I/O routine consists of placing the parameter in the C-Register (if it is a byte value), or in the C- and D-Registers (if it is an address value) then calling the I/O routine. When address values are used, the least significant byte must be placed in the C-Register; the most significant byte must be placed in the D-Register.

Parameter passing from the system I/O routine to the user routine consists of calling the I/O routine first, then retrieving the data from the A-Register (if it is a byte value), or from the A- and B-Registers (if the data is an address value). The least significant byte is retrieved from the A-Register and the most significant byte is retrieved from the B-Register.

This mechanism of parameter passing is the same as that performed by PL/M.[T.M.] It is equivalent to the operation produced by declaring and calling a byte procedure with parameters of type address. A user program written in PL/M has the mechanism to pass data to and from the Monitor´s drivers.

Accessible I/O can be considered from two general points of view: A user can pass data to and from the Monitor´s drivers, and can call a driver; a user can request system status information, including device status, memory size, and I/O system status.


## I/O DRIVER ROUTINES

The following paragraphs contain descriptions of the use of the I/O driver routines. The following functions are included:

- CI        Console Input

- CO        Console Output

- RI        Reader Input

- PO        Punch Output

- LO        List Output

CI - Console Input

The CI routine is an Intellec/MDS Monitor driver that returns a
character received from the selected system console device and places it
in the A-Register.  Once started, the CI routine loops until a character
is input.  No timeout facilities are included.  The character read is
not echoed.  The A-Register and the CPU condition code are affected by
this operation.

Examples:

        The basic assembly language calling sequence is as follows:

```
            CI        EQU       0F803H
                      .
                      .
                      .

                      CALL      CI
                      LXI       H,DATA
                      MOV       M,A
```

        The following routine is a sample of assembly language coding in
        which  the Console Input facilities are used.  This routine will
        input a  string  of  characters  from  the  console  device  and
        terminate  its  operation  when  either  a  carriage  return  is
        detected as an input  character  or  the  number  of  characters
        specified  in BUFSIZ have already been read.  The two exits DONE
        and  OVFL  correspond  to  the  CR  detected  and  buffer   full
        conditions respectively.

```
            ; CONSOLE INPUT ROUTINE
            ; INPUT UP TO 72 CHARACTERS FROM CONSOLE DEVICE
            ; A CARRIAGE RETURN IN INPUT CAUSES EXIT TO ´FULL´
            ; INPUT OF 72 CHARACTERS CAUSES EXIT TO ´OVFL´
            ;
                      ORG       20H
            CI        EQU       0F803H   ; RESERVED ADDRESS OF CI ROUTN
            CR        EQU       0DH      ; CARRIAGE RETN
            BUFSIZ    EQU       72       ; BUFFER SIZE 72 CHAR
            ;
            START:
                      LXI       H,BUFFR ; SET UP BUFFER POINTER
                      MVI       D,BUFSIZ; SET UP BUFFER SIZE
            BPC:
                      CALL      CI       ; GET CHARACTER
                      ANI       7FH      ; STRIP OFF PARITY BIT
                      MOV       M,A      ; SAVE CHARACTER
                      CPI       CR       ; IS IT CARRIAGE RETN
                      JZ        DONE     ; YES, TAKE DONE EXIT
                      INX       H        ; NO, MOVE BUFFER POINTER
                      DCR       D        ; DECR CHARACTER COUNT
                      JZ        OVFL     ; IF BUFFER FULL, TAKE ´OVFL´
```

```
                                     ; EXIT
                      JMP     BPC
            ;
            DONE:     xxxx    xxxx
                      ,
                      ,
            OVFL:     xxxx    xxxx
                      ,
                      ,
            BUFFR:    DS      BUFSIZ  ; RESERVE 72 LOCATIONS FOR
                                      ; BYTES RETURNED BY CI
```

CO - Console Output

The CO routine is an Intellec MDS Monitor driver that takes a character
from the C-Register and transmits it to the system CONSOLE for output.
The A-Register, C-Register, and CPU Condition Codes are affected by this
operation.

Examples:

The basic assembly language calling sequence is as follows:

```
            CO        EQU     0F809H
                      ,
                      ,
                      MOV     C,M
                      CALL    CO
                      ,
                      ,
                      ,
```

The following routine is a sample of assembly language
programming in which the Console Output facilities are used.
This routine allows a user to output a string of characters on
the CONSOLE.  The operation terminates when a carriage return is
detected in the text string.  The carriage return character is
output, then the operation is terminated.

```
            CR        EQU     ODH
            CO        EQU     0F809H  ; RESERVED ADDRESS
            OTPT:
                      LXI     H,MSGBUF; GET BASE ADDRESS OF MESSAGE
                                      ; BUFFER
                      MOV     C,M     ; GET CHAR FROM BUFFER
                      CALL    CO      ; OUTPUT CHARACTER
                      MVI     A,CR    ; IS CHAR CR?
                      CMP     M
                      JZ      RTN     ; YES, EXIT THROUGH ´RTN´
                      INX     H       ; INCREMENT BUFFER POINTER
```

```
                    JMP       OTPT
            RTN:
                    xxxx      xxxx
            MSGBUF: ; MSGBUF IS THE BASE ADDRESS OF
                    ; THE STRING OF CHARACTERS WHICH
                    ; ARE OUTPUT TO THE CONSOLE DEVICE
                    ·
                    ·
                    ·
```

RI – Reader Input

The RI routine is an Intellec MDS Monitor driver that returns a
character from the system READER and places it in the A-Register.  If no
character is read within a waiting period of 250 milliseconds, due to an
end of file condition, the A-Register is zeroed and the Carry condition
bit in the 8080 is set to one.  Thus, immediately following a call to
this driver, a check can be made to determine if the carry bit is set or
reset.  If the carry bit is one, the character in the A-Register is
invalid or is zero.  If the carry bit is zero, the character in the
A-Register is a valid character.  Once the end of file condition is
sensed, control is returned to the calling program.

The A-Register and the CPU condition codes are affected by this
operation.

Examples:

        The basic assembly language calling sequence is as follows:

```
            RI        EQU       0F806H
                      ·
                      ·


                      CALL      RI
                      JC        LAST
                      LXI       H,DATA
                      MOV       M,A
                      ·
                      ·
                      ·
```

        The following routine is a sample of assembly language
programming in which the reader input facilities are used.  This
routine reads a string of characters from paper tape and  stores
them in an expanded buffer area.  When the reader either runs
out of punched tape, or a Control/Z (1A Hexadecimal) character
is read (it is used in this context as an EOF character), the
operation is terminated and control is returned to the calling
program.

```
                    ORG     20H
            RI      EQU     0F806H
            EOF     EQU     1AH       ; USED AS EOF CHAR
            ;
            START:
                    LXI     H,BUFF    ; BASE OF ADDR BUFFR
            LOOP:
                    CALL    RI        ; GET CHARACTER
                    JC      EFEX      ; IF CY NOT ZERO TAKE
                                      ; ´EFEX´ EXIT
                                      ; NO CHAR FOR 250 MS
                    ANI     7FH       ; STRIP OFF PARITY BIT
                    MOV     M,A       ; STORE CHAR IN BUFFR
                    CPI     EOF       ; IS CHAR EOF ?
                    JZ      EFEX      ; YES, TAKE EFEX EXIT
                    INX     H         ; NO, ENLARGE BUFFER
                    JMP     LOOP
            EFEX:   xxxx    xxxx      ;; EOF EXIT
                    ;
                    ,
            BUFF:
                    DS      1
                    ,
                    ,
```

PO - Punch Output

The PO routine is an Intellec MDS Monitor driver that takes a character
from tne C-Register and transmits it to the device selected as the
system punch device.

The A-Register, C-Register, and CPU condition codes are affected by this
operation.

Examples:

        The basic assembly language calling sequence is as follows:

```
            PO      EQU     0F80CH
                    ´
                    ,
                    ,
                    MOV     C,M
                    CALL    PO
                    ,
                    ,
```

LO - List Output

The LO routine is an Intellec MDS Monitor driver that takes a character
from the C-Register and transmits it to the system LIST device.

The A-Register, C-Register, and CPU condition codes are affected by this operation

Examples:

The basic assembly language calling sequence is as follows:

```
LO       EQU       0F80FH
         ,
         ,
         ,

         MOV       C,M
         CALL      PO
         ,
         ,
```

The following routine is a sample of assembly language programming in which the LIST output device is used. This routine reads characters from a buffer area in memory and continues outputting contiguous characters until an ETX (03H) character is encountered in the text string.

```
         ORG       20H
LO       EQU       0F80FH    ; LO ROUTINE FIXED
                             ; ADDRESS
ETX      EQU       03H       ; TERMINAL CHAR
START:
         LXI       H,BUFFR   ; SET UP POINTER
PRNT:
         MOV       A,M       ; CHAR INTO ACCUM
         CPI       ETX       ; IS IT ´ETX´?
         JZ        XIT       ; YES, EXIT
         MOV       C,M       ; NO, SET UP CHAR
         CALL      LO        ; PRINT CHAR
         JMP       PRNT      ; RETURN FOR NEXT CHAR
BUFFR:   DB        ´THIS IS THE MESSAGE´
         DB        ETX       ; TERMINAL CHARACTER AT
                             ; END OF BUFFER
XIT:     xxxx      xxxx
                             ; EXIT
         END
```

## SYSTEM STATUS ROUTINES

The system status information routines include the following:

- CSTS    Console Input Status

- IOCHK   Check I/O System Configuration

- • IOSET Set I/O System Configuration

- • MEMCHK Determine Size Of RAM Memory

CSTS - Console Input Status

In many applications there is a need to poll the console device to see
if the operator wishes to terminate the current operation. The CSTS
routine allows the caller to test the console to see if a character is
ready for input. In other words, it checks to see if a console keyboard
key was pressed since the last CI operation. If no key was pressed, the
value of 00 is returned in the A-Register. If a key was pressed, a
value of 0FFH is returned in the A-Register. A CI operation may then be
initiated to retrieve the character.

Examples:

The basic assembly language calling sequence is as follows:

```
        CSTS    EQU     0F812H
                ´
                ´
                ´

        CALL    CSTS
                RRC
                JNC     NOCHAR  ; NO KEY WAS PRESSED
        KYPRS:  xxxx            ; KEY WAS PRESSED
                ´
                ´

        NOCHAR:
                xxxx
```

The following routine is an example of use of the CSTS routine.
During a print (console output) operation, it is necessary to
monitor the keyboard so that the operator has facilities to
signal that the operation is to be terminated.

```
                ORG     20H
        CSTS    EQU     0F812H  ; FIXED STARTING ADDR
        CTLC    EQU     03H     ; CONTROL/C
        CI      EQU     0F803H
                ´
                ´

                CALL    CSTS    ; GET STATUS
                RRC             ; ROTATE ACC INTO CY
                JNC     CONT    ; NO CHAR, CONTINUE
                CALL    CI      ; GET CHAR
                CPI     CTLC    ; IS IT CONTROL/C?
                JZ      STP     ; YES, JMP TO INTERRUPT
                ´
```

```
                              ,
                              ,
               CONT:
                         xxxx      xxxx
                              ,
                              ,

               STP:
                         xxxx      xxxx      ; BREAK ROUTINE
                              ,
                              ,
```

IOCHK - Check I/O System Configuration

The IOCHK routine returns an 8-bit value in the A-Register which
describes the current assignment of physical devices to logical devices.
This routine allows the user to programmatically select I/O devices and
configure the I/O system. For example, if the system punch device can
be assigned to either a free standing punch device or a combination
print and punch device, the manner in which data is output may be
different for each device. The free standing punch device may be left
in an active mode at all times and be used by the user calling program
at any time. However, a combination punch/printer device usually
requires that the punch be turned off while printing. Otherwise,
massive amounts of tape will be wasted. Before using this type of punch
device, the user must be notified to turn it on, and similarly to turn
it off. Using the IOCHK routine, a user may determine which device is
being used, and tailor the user coding to the device.

The status byte that is returned contains the current mapping of logical
I/O devices to physical I/O devices. The 8-bit byte is divided into
four 2-bit fields as shown below:

BIT

```
     7   6   5   4   3   2   1   0       STATUS WORD
         |       |       |       |
         |       |       |       |
         |       |       |       +——————————— CONSOLE
         |       |       +——————————————————— READER
         |       +——————————————————————————— PUNCH
         +——————————————————————————————————— LIST
```
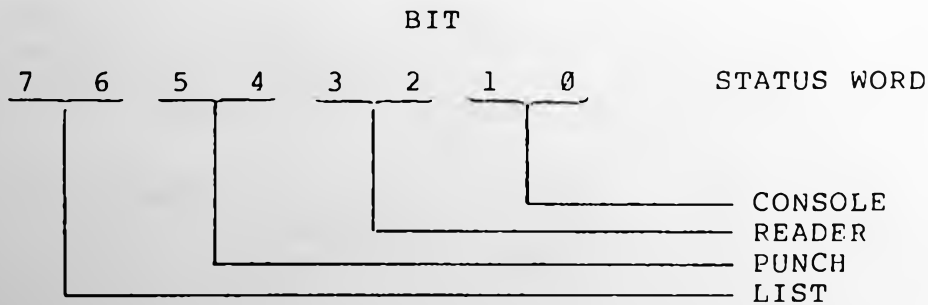
Each field may contain a value of 0 through 3 which represents the
physical device currently assigned to it. Each of the fields is
described as follows:

```
      CONSOLE          0          Assigned to TTY
```

|        |   |                                                  |
|--------|---|--------------------------------------------------|
|        | 1 | Assigned to CRT                                  |
|        | 2 | BATCH: Special assignment where CONSOLE input function is reassigned to system READER device, and CONSOLE output function is reassigned to system LIST device. |
|        | 3 | Assigned to user-defined I/O device (to be discussed later). |
| READER | 0 | Assigned to TTY                                  |
|        | 1 | Assigned to high speed paper tape reader         |
|        | 2 | Assigned to user-defined I/O device 1 (to be discussed later). |
|        | 3 | Assigned to user-defined I/O device 2 (to be discussed later). |
| PUNCH  | 0 | Assigned to TTY                                  |
|        | 1 | Assigned to high speed paper tape punch          |
|        | 2 | Assigned to user-defined I/O device 1 (to be discussed later) |
|        | 3 | Assigned to user-defined I/O device 2 (to be discussed later) |
| LIST   | 0 | Assigned to TTY                                  |
|        | 1 | Assigned to CRT                                  |
|        | 2 | Assigned to line printer                         |
|        | 3 | Assigned to user-defined I/O device 1 (to be discussed later) |

The Monitor is initially configured at cold start time to assign the CONSOLE to the device that is first operated during initialization, and the TTY to all other devices.

Example:

The basic assembly language calling sequence is as follows:

```
IOCHK   EQU     0F815H
```

```
                              .
                              .
                              .
                  CALL      IOCHK
                  ANI       DEVMSK
                  CPI       DEVICE
                  JZ
                              .
                              .
                              .
```

Values for DEVMSK are as follows:

```
        CONSOLE  03H       00000011B
        READER   0CH       00001100B
        PUNCH    30H       00110000B
        LIST     C0H       11000000B
```

Values for the peripheral devices for DEVICE are as follows:

```
        CONSOLE  TTY       00       00000000B
                 CRT       01       00000001B
                 BATCH     02       00000010B
                 User      03       00000011B

        READER   TTY       00       00000000B
                 PTR       04       00000100B
                 User 1    08       00001000B
                 User 2    0CH      00001100B

        PUNCH    TTY       00       00000000B
                 PTP       10H      00010000B
                 User 1    20H      00100000B
                 User 2    30H      00110000B

        LIST     TTY       00       00000000B
                 CRT       40H      01000000B
                 LPTR      80H      10000000B
                 User      C0H      11000000B
```

In the following example, a check is made to determine if the
CONSOLE device is a CRT.  If it is not, the routine exit is
´CONTINUE´.  If it is a CRT, the routine exit is ´OUTCRT´.

```
        IOCHK     EQU       0F815H
        DEVMSK    EQU       03H
        DEVICE    EQU       1
                    .
                    .
                  CALL      IOCHK
                  ANI       DEVMSK  ; MASK ALL BUT CONSOLE
```

```
                              CPI      DEVICE   ; IS IT A CRT?
                              JZ       OUTCRT   ; YES,
                     CONTINUE:                  ; NO
                              xxxx     xxxx
                              xxxx     xxxx
                              xxxx     xxxx
                     OUTCRT:
                              xxxx     xxxx
```

IOSET - Set I/O System Configuration

The IOSET routine allows the user to modify the system status word.  The
new  value of the status byte is placed in the C-Register before calling
the IOSET routine.

The A-Register, C-Register, and the  CPU  flags  are  modified  by  this
routine.

Examples:

     The basic assembly language calling sequence is as follows:

```
             IOSET    EQU      0F818H
             IOCHK    EQU      0F815H
                      ;
                      ;
                      ;

             CALL     IOCHK    ; GET STATUS WORD
             ANI      NOT DEVMSK
                               ; CLEAR CURRENT DEV
             ORI      NEWDEV   ; ENTER NEW DEV
             MOV      C,A
             CALL     IOSET    ; REPLACE STATUS BYTE
                      ;
                      ;
```

     Suppose a user wishes to change the I/O Status  byte  to  change
     the CONSOLE device from CRT to TTY.  The following is an example
     of the coding:

```
             IOSET    EQU      0F818H
             IOCHK    EQU      0F815H
             DEVMSK   EQU      03H
             NEWDEV   EQU      0
                      ;
                      ;
                      ;

             CALL     IOCHK    ; GET STATUS BYTE
             ANI      NOT DEVMSK
                               ; CLEAR CURRENT CONSOLE
                               ; DEVICE ASSIGNMENT
```

```
                        ORI     NEWDEV  ; ASSIGN TTY TO
                                        ; CONSOLE
                        MOV     C,A
                        CALL    IOSET   ; STORE MODIFIED BYTE
```

MEMCHK - Determine Size of RAM Memory

The MEMCHK routine provides the user the ability to determine the
highest  RAM address which is currently available in the system.  MEMCHK
returns the highest address available to the user after the Monitor  has
allocated its own storage at the top of contiguous RAM memory.

If RAM memory is installed in segments,  instead  of  contiguously  from
address 0000 upward, the Monitor will assign its own storage area at the
top of the first segment of RAM memory.  In this case, additional memory
will  exist  above  the memory address determined by the MEMCHK routine.
However, the location and extent of this memory must  be  determined  by
the user.

Examples:

        The basic assembly language calling sequence is as follows:

```
            MEMCHK  EQU     0F81BH
                    .
                    .

                    CALL    MEMCHK
                    LXI     H,MEXMEM        ; LOCATION ASSIGNED
                                            ; TO HOLD ADDRESS
                    MOV     M,A             ; LSB IN A
                    INX     H
                    MOV     M,B             ; MSB IN B
                    .
                    .
```

        A practical example of the use of MEMCHK is the  setting  up  of
        the  Stack  Pointer.  A full description of the Stack Pointer is
        included in  the  8080  Assembly  Language  Programming  Manual.
        Generally,  it  is advisable to move the Stack Pointer into high
        memory, away from the user coding.  A suitable location to start
        the Stack Pointer is at the high memory location determined from
        MEMCHK.  The coding to set up the Stack Pointer is as follows:

```
            MEMCHK  EQU     0F81BH
                    .
                    .

                    CALL    MEMCHK          ; FIND HIGH MEM ADDR
                    MOV     H,B             ; MSB IN B
                    MOV     L,A             ; LSB IN A
```

                      SPHL                            ; LOAD ADDRESS INTO
                                                      ; STACK POINTER

                        .
                        .
                        .


EXTENDING THE I/O SYSTEM

The user may write special drivers for non-standard system  devices  and
install  them  into  the  Intellec MDS system.  By having them installed
into the I/O system, all system programs and user  programs  may  access
them.


IODEF - Define User-Written I/O Routines

The Monitor program defines a set of absolute  addresses  that  it  will
branch  to in the event that the I/O status byte contains a selection of
user-defined I/O devices.  The system routine IODEF  allows  a  user  to
define  the  logical  device  category of the device in question and the
starting address of the  routine.   IODEF  requires  two  parameters  as
follows:

        (1) A byte value from the  following  table,  defining  the  logical
            device  category.   This  value must be placed in the C-Register
            prior to calling IODEF.

                        0          User Defined Console Input
                        1          User Defined Console Output
                        2          User Defined Reader 1
                        3          User Defined Reader 2
                        4          User Defined Punch 1
                        5          User Defined Punch 2
                        6          User Defined List Device
                        7          User Defined Console Status

        (2) The starting address of the driver routine, written by the user.
            This  address  value  must  be  placed in the D- and E-Registers
            prior to calling IODEF, with the most significant  byte  in  the
            D-Register and the least significant byte in the E-Register.


The driver routines written by the user should save and restore any  CPU
registers that are not specifically used to pass parameters.


Examples:

                IODEF     EQU       0F81EH

```
                        .
                        .
                        .
             MVI        C,LOGDEV           ; SET UP LOGIC
                                           ; DEVICE SELECTION
             LXI        D,DRVADD           ; GET DRIVER
                                           ; STARTING ADDRESS
             CALL       IODEF
                        .
                        .
                        .
```

For example, if a user wishes to write a driver for  a  magnetic
tape  cassette  and  define  it  as  a  system PUNCH device, the
following coding can be used to  install  the  driver  into  the
Monitor  I/O system.  Assume that the driver starting address is
labeled, 'MAGDRVR'.

```
         IODEF      EQU      0F81EH
         P1LOC      EQU      4
                    .
                    .
                    .
             MVI        C,P1LOC            ; SET UP PUNCH1
             LXI        D,MAGDRVR          ; ADDR OF DRVR
             CALL       IODEF
                    .
                    .
                    .
```

APPENDIX A

INTELLEC® MDS MONITOR COMMANDS


The following brief descriptions are intended as a quick reference to the function and syntax of the Intellec MDS Monitor commands. More complete descriptions of the commands are given in Section 3 of the manual.


A          ASSIGN I/O DEVICE

           A<logical dev>=<physical dev>


           Valid values for <logical dev> and <physical dev> are:

                   <logical dev>              <physical dev>

                   C or CONSOLE               B or BATCH
                                              C or CRT (Default)*
                                              T or TTY (Default)*
                                              1 (user-defined device)

                   L or LIST                  C or CRT
                                              L or LPT
                                              T or TTY (Default)
                                              1 (user-defined device)

                   P or PUNCH                 P or PTP
                                              T or TTY (Default)
                                              1 (user-defined device)
                                              2 (user-defined device)

                   R or READER                P or PTR
                                              T or TTY (Default)
                                              1 (user-defined device)
                                              2 (user-defined device)


*One assigned during Cold Start procedure.  See Section 2.

B       PUNCH BNPF FILE

        B<low address>,<high address>


C       COMPARE PROM TO RAM

        C<t/f><socket option><low address>,<high address>

        where <t/f> = T for positive TRUE, F for negative TRUE; and
        <socket option> = X for Socket 2 (24 pin), Y for Socket 1 (high
        4 bits), and Z for Socket 1 (low 4 bits).


D       DISPLAY CONTENTS OF MEMORY

        D<low address>,<high address>


E       PUNCH HEXADECIMAL END-OF-FILE

        E<start address>


F       FILL RAM WITH CONSTANT

        F<low address>,<high address>,<constant>


G       EXECUTE PROGRAM (GO)

        G<start address>,<breakpoint 1>,<breakpoint 2>

        where <breakpoint 1> and <breakpoint 2> are always optional.  If
        <start address> is absent, execution resumes at stored value of
        user's Program Counter.


H       HEXADECIMAL ARITHMETIC

        H<number 1>,<number 2>

        The two results printed are $((<number\ 1>+<number\ 2>)\ MOD\ 2^{16})$
        and $((<number\ 1>-<number\ 2>)\ MOD\ 2^{16})$, respectively.


L       LOAD BNPF FILE

        L<low address>,<high address>

M           MOVE MEMORY

            M<low address>,<high address>,<destination address>


N           PUNCH NULL CHARACTERS

            N

            Each N will punch 60 NULL characters.


P           PROGRAM A PROM

            P<t/f><socket option><low address>,<high address>,<PROM address>

            For values for <t/f> and <socket option>, see the C command.


Q           I/O STATUS QUERY

            Q

            For meaning of values displayed, see the A command.


R           READ HEXADECIMAL FILE

            R<bias>

            <bias> will be added (MOD 2**16) to indicated load addresses.


S           SUBSTITUTE MEMORY

            S<address>

            Continue command with  ´ ´  or  ´,´;   terminate  with  carriage
            return.


T           TRANSFER PROM TO RAM

            T<t/f><socket option><low address>,<high address>

            For values for <t/f> and <socket option>, see the C command.


W           WRITE HEXADECIMAL FILE

            W<low address>,<high address>

X        EXAMINE AND MODIFY CPU REGISTERS

         X<register identifier>

         where <register identifier> has the following meanings:

         A,B,C,D,E,H,L    Corresponding 8080 CPU register.

         M                H and L together.
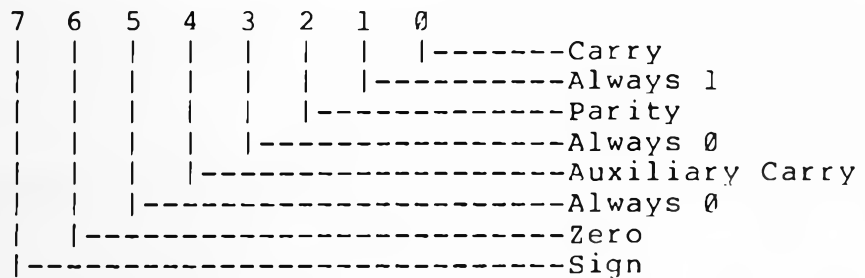
         P                Program Counter.

         S                Stack Pointer

         I                Intellec MDS Interrupt Mask Register.

         F                8080 CPU flags, packed as follows:

```
7  6  5  4  3  2  1  0
|  |  |  |  |  |  |  |-------Carry
|  |  |  |  |  |  |----------Always 1
|  |  |  |  |  |-------------Parity
|  |  |  |  |----------------Always 0
|  |  |  |-------------------Auxiliary Carry
|  |  |----------------------Always 0
|  |-------------------------Zero
|----------------------------Sign
```

         If <register identifier> is absent, the Monitor will display
         each register in alphabetical order.  Type a ´ ´ or ´,´ to
         continue command, or a carriage return to terminate.

APPENDIX B

TEXT EDITOR COMMANDS


A          Append text from READER to Text Buffer.


B          Move buffer pointer to beginning of Text Buffer.


nC         Move buffer pointer "n" character positions to the right   (if   n
           is positive) or to the left (if n is negative).


nD         Delete the "n" characters to the right (if n is positive) or   to
           the left (if n is negative) of the buffer pointer.


E          Empty Text Buffer to PUNCH;  copy remaining input in   READER   to
           PUNCH;  reinitialize Text Editor.


Fstring$
           Find the first  occurrence  of  "string"  following   the   buffer
           pointer.


Istring$
           Insert "string" into  Text  Buffer  at  current  buffer   pointer
           position.


nK         Delete ("Kill") the "n" lines following (if n  is   positive)   or
           preceding (if n is negative) the buffer pointer.


nL         Move the buffer pointer "n" lines forward (if n is positive)   or
           backward (if n is negative).

N          Punch 60 NULL characters.


Sstring 1$string 2$
           Substitute "string 2" for the  first  occurrence  of  "string 1"
           following the buffer pointer.


nT         Type the "n" lines following (if n is positive) or preceding (if
           n is negative) the buffer pointer.


nW         Write the first "n" lines of  the  Text  Buffer  to  the  PUNCH;
           delete  these  lines  from  the  Text  Buffer.   Note:   "n"  is
           interpreted as a positive number, regardless of the sign used in
           the command.


Z          Move buffer pointer to end of Text Buffer.


n< ... >
           Repeat the commands enclosed in "< ... >" "n" times.

# APPENDIX C

## ASSEMBLER ERROR MESSAGES

Errors detected by the assembler are indicated by single-letter codes on the output listing. When multiple errors are detected in a single line of code, only the first error is flagged.

The error flags and their meanings are:


B       Balance Error

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

Example:

ORG $/256+1)*256-$
DB 'A''


E       Expression Error

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

Example:

ORG ($/256+1)256-$


F       Format Error

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

Example:

```
MOV A,
MOV A,B,C
```

I       Illegal Character

This message 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 A,02B
        ADI A,79Q


M       Multiple Definition

The M error message indicates that a symbol or a macro is
defined more than once.  The M error will occur on all
definitions of, and all references to, the multiply-defined
symbol.  Recall that symbols must be unique in the first five
characters;  therefore, long symbol names which differ only at
the end may cause an M error.

    Example:

        LOCATION1:  NOP
        LOCATION2:  NOP

        will cause an M error.


N       Nesting Error

This message indicates that an ENDIF, ENDM, or END statement  is
improperly nested.

    Example:

        ENDIF

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

P       Phase Error

This message indicates that the value of an element being
defined changed between pass one and pass two of 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 encountered.  The assembler will assume it
to be at location zero and begin assembling the  program
at  that  location.  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

        This message is usually caused by  omitting  or  misspelling  an
        opcode.

            Example:

                ...  34H,B3H


R       Register Error

        This  message  indicates  that  a  register  specified  for  an
        operation is invalid for the operation.

            Example:

                INR 9


S       Stack Overflow

        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 <EXPRESSION>
          IF <EXPRESSION>
          IF <EXPRESSION>
          .
          .
          .
          ENDIF
          ENDIF
          ENDIF
```

may cause an S error.


T       Table Overflow

This message indicates that the assembler's symbol table space
has been exhausted.  This is caused by using too many symbols in
one assembly, or by accumulating more macro text than the
assembler can store in the memory available.


U       Undefined Identifier

This message indicates that a symbol used in an operand field
has never been defined by appearing in the label field of
another instruction.

    Example:

        If the statement

        JMP LAB1

        is in the program, but LAB1 does not appear in the label
        field of any other statement, it will cause a "U" error.


V       Illegal Value

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

    Example:

        The statement

        MVI A,257

        will cause a "V" error because the second operand of an
        "MVI" instruction must be in the number range 0 to 255.

# APPENDIX D

## OBJECT CODE FORMATS

### Hexadecimal Object File Format

Hexadecimal object code format is an ASCII representation of program memory, expressed as a series of hexadecimal digits. These are blocked into records, each of which contains the record length, type, memory load address, and checksum, in addition to data. The description below applies to paper tape on a frame-by-frame basis.


Frame 0:   RECORD MARK

> A colon (3A in base 16) is used to signal the start of a record.


Frames 1, 2:   RECORD LENGTH

> This is the count of the actual data bytes in the record. Frame 1 contains the high-order digit of the count, and frame 2 contains the low-order digit. A record length of zero indicates end of file.


Frames 3-6:   LOAD ADDRESS

> The four-character starting address at which the following data will be loaded. The high-order digit of the load address is in frame 3, and the low-order digit is in frame 6. The first data byte is stored in the location indicated by the load address. Successive data bytes are stored in successive memory locations.


Frames 7, 8:   RECORD TYPE

> A two-digit code in this field specifies the type of this record. The high-order digit of this code is located in frame 7. Currently, all data records are type 0. End-of-file records may be type 0 or type 1; in either case, they are distinguished by a zero RECORD LENGTH field (see above). Other possible values for this field are reserved for future expansion.

Frames 9 to 9+2*(record length)-1:  DATA

>   Each 8-bit memory word is represented by two  frames  containing
>   ASCII  characters  0-9,A-F,  which  represent a hexadecimal value
>   between 0 and FF (0 and 255 decimal).  The high-order  digit  of
>   each byte is located in the first frame of each pair.

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, beginning with the RECORD LENGTH and ending with the
>   last DATA byte, evaluated modulo 256.  The sum of all  bytes  in
>   the record (including the checksum) should be zero.

BNPF Object File Format

BNPF format is an ASCII representation of a byte in pure binary form.  A B  is  punched  to indicate the beginning of a byte.  Following the B, a string of P´s and N´s will be punched, with a  ´P´  representing  a  ´1´ bit,  and an ´N´ representing a ´0´ bit.  An ´F´ is used to indicate the end of a byte.  All characters following the F are ignored until another B is encountered.  This allows comments (not containing the letter B) to appear between bytes of data in BNPF format.

Bits of data in a BNPF byte appear  in  left-to-right  order  from  most significant to least significant.

Example:  The two bytes ´3AF0´ would be represented in BNPF format as

        BNNPPPNPNF BPPPPNNNNF

# APPENDIX E

## MDS MONITOR I/O SYSTEM


Entry Point Addresses for MDS Monitor I/O Routines

| Routine | Address | Function |
|---------|---------|----------|
| CI | 0F803H | Console Input |
| RI | 0F806H | Reader Input |
| CO | 0F809H | Console Output |
| PO | 0F80CH | Punch Output |
| LO | 0F80FH | List Output |
| CSTS | 0F812H | Console Status |
| IOCHK | 0F815H | Check I/O System Configuration |
| IOSET | 0F818H | Set I/O System Configuration |
| MEMCHK | 0F81BH | Determine Size of Available RAM |
| IODEF | 0F81EH | Incorporate User-Written I/O Drivers |


Device Selection Codes for Use with IODEF

| Driver Function | Code |
|-----------------|------|
| Console Input | 0 |
| Console Output | 1 |
| Reader (1) | 2 |
| Reader (2) | 3 |
| Punch (1) | 4 |
| Punch (2) | 5 |
| List | 6 |
| Console Status | 7 |


Description of the I/O status byte

The I/O system status byte contains the current mapping of logical I/O devices to physical I/O devices. The 8-bit byte is subdivided into four 2-bit fields, each field corresponding to a logical device as follows:

| | | |
|--|--|--|
| bits 0,1 | — | CONSOLE |
| bits 2,3 | — | READER |
| bits 4,5 | — | PUNCH |
| bits 6,7 | — | LIST |

Each field can contain a value 0-3 which represents the physical  device
currently  assigned  to it.  The following paragraphs discuss each field
in detail.

CONSOLE field, bits 0,1

```
00        - CONSOLE is assigned to the TTY
01        - CONSOLE is assigned to the CRT
10        - BATCH mode: use the READER as CONSOLE input,
             the LIST device as CONSOLE output.
11        - User-defined CONSOLE
```

READER field, bits 2,3

```
00        - READER = TTY
01        - READER = high speed reader, PTR
10        - User-defined READER (1)
11        - User-defined READER (2)
```

PUNCH field, bits 4,5

```
00        - PUNCH = TTY
01        - PUNCH = high speed punch, PTP
10        - User-defined PUNCH (1)
11        - User-defined PUNCH (2)
```

LIST field, bits 6,7

```
00        - LIST = TTY
01        - LIST = CRT
10        - LIST = LPT
11        - User-defined LIST device (1)
```

APPENDIX F

INTERRUPT PROCESSING ON THE INTELLEC® MDS

## Introduction

Interrupt processing on the Intellec MDS is controlled by logic on the
8080 CPU Module.  This module provides an eight-level priority interrupt
structure, using an Interrupt Mask Register and a "current operating
level" indicator, which keeps track of the level of interrupt (if any)
currently being serviced.  The Interrupt Mask Register, which can be set
by a program or from the CONSOLE, permits the user to select which
interrupts will be acknowledged at any time.

## Priority of Interrupts

The Intellec MDS bus provides eight interrupt lines, numbered 0 through
7, corresponding to the eight Interrupt switches and lights on the MDS
Front Panel.  Interrupt 0 is the highest priority, and Interrupt 7 is
the lowest.  Thus, for example, an interrupt of level 4 which is
currently being serviced can itself be interrupted to service an
interrupt of level 3, 2, 1, or 0, but cannot be interrupted to service
one of level 5, 6, or 7, nor by another interrupt of level 4.

## The Interrupt Mask Register

The Interrupt Mask Register on the 8080 CPU Module determines which
interrupts will be accepted by the MDS.  The Interrupt Mask Register
contains 8 bits, numbered 0 (least significant) through 7 (most
significant).  Each bit controls the recognition of interrupts at the
corresponding level. A "1" bit in the Interrupt Mask Register prevents
the corresponding interrupt from being serviced; a "0" bit allows the
interrupt to be serviced.  For example, the MDS Monitor sets the
Interrupt Mask Register to 0FEH = 11111110, thereby blocking all
interrupts but Interrupt 0.

The Interrupt Mask Register can be set programmatically by writing the
desired value to Port 0FCH;  thus

```
        MVI     A,0F0H
        OUT     0FCH
```

sets the Interrupt Mask Register to 11110000, blocking interrupts 4 - 7 and permitting interrupts 0 - 3.

A program can also read the current value of the Interrupt Mask Register from Port 0FCH:

        IN        0FCH

places the current value of the Interrupt Mask Register into the A-register.

There are three phases to interrupt processing on the MDS:

- Initialization

- Acceptance

- Removal

These are discussed in detail below.


## Initialization

The interrupt logic on the 8080 CPU Module must be initialized whenever the RESET switch on the front panel is used. The following steps must be done in the order indicated:

- A value of 12H must be written to Port 0FDH.

- A value of 00H must be written to Port 0FCH.

- The Interrupt Mask Register must be set to the desired value.

This can be accomplished as follows:

```
        .
        .
        .
    MVI    A,12H
    OUT    0FDH
    MVI    A,00H
    OUT    0FCH
    MVI    A,MASK
    OUT    0FCH
        .
        .
        .
```

where MASK has been set by an EQU or SET statement.

## Interrupt Acceptance

When an interrupt occurs, the 8080 CPU Module checks the Interrupt Mask Register to see if an interrupt at that level is permitted. If it is not, no further action is taken; in particular, the interrupt is not cleared, and remains pending on the Intellec bus. If the interrupt is permitted, the CPU Module checks the "current operating level" to see if another interrupt of equal or higher priority is being serviced. If so, the new interrupt remains pending until the value of the "current operating level" is less than the priority of the new interrupt.

If this interrupt can be serviced now, the following actions take place:

- The CPU Module transmits a signal to the Interrupt Request line of the 8080, then locks out all interrupts coming from the Intellec bus.

- When the 8080 responds with an Interrupt Acknowledge, the CPU Module generates an RST instruction to one of eight addresses (see table below), stacks the current operating level to reflect the new interrupt, and resets the 8080 Interrupt Request Line.

- The lockout placed on interrupts from the Intellec bus is removed.

The addresses called when an interrupt is accepted are:

| Interrupt Level | Address |
|-----------------|---------|
| 0 | 0000H |
| 1 | 0008H |
| 2 | 0010H |
| 3 | 0018H |
| 4 | 0020H |
| 5 | 0028H |
| 6 | 0030H |
| 7 | 0038H |

## Interrupt Removal

The program servicing the interrupt must do two things: it must transmit a signal to the interrupting device, telling it to remove the interrupt signal it generated initially; and it must restore the current operating level maintained by the CPU Module. The former action is device-dependent; the latter is accomplished by writing a value of 20H to Port 0FDH. This must be done with interrupts disabled. If the code permits another interrupt to be serviced while this is being done, a stack overflow could result. A sample sequence for doing this is:

```
          .
          .
          .
        <remove interrupt signal from external device>
        DI
        MVI       A,20H
        OUT       0FDH
        POP       PSW       ; RESTORE A REGISTER AND FLAGS
        EI
          .
          .
          .
```

The example below shows the code necessary to service  an  interrupt  at
level 1.

```
        ORG       8         ; RST ADDRESS FOR INTERRUPT 1
        JMP       INT1
        ORG       40H       ; ABOVE AREA USED BY MONITOR AND RST´S
INT1:
        EI                  ; INTERRUPT ROUTINES CAN BE INTERRUPTED
        PUSH      PSW       ; SAVE REGISTERS
        PUSH      B
        PUSH      D
        PUSH      H
          .
          .
          .
        <code to service interrupt and remove signal>
          .
          .
          .
        POP       H         ; RESTORE REGISTERS
        POP       D
        POP       B
        DI                  ; CRITICAL SECTION: DISABLE INTERRUPTS
        MVI       A,20H     ; RESTORE CURRENT OPERATING LEVEL
        OUT       0FDH
        POP       PSW       ; RESTORE A REGISTER AND FLAGS
        EI                  ; PERMIT INTERRUPTS AFTER NEXT INSTRUCTION
        RET                 ; MUST IMMEDIATELY FOLLOW ´EI´ TO
                            ; MAKE SURE IT´S EXECUTED BEFORE ANOTHER
                            ; INTERRUPT OCCURS
```