# MITS Programming System II
## (Rev 3.0)

## TABLE OF CONTENTS

*Copyright © MITS, Inc. 1976*

# I. Glossary

Machine instruction - Binary byte(s) that execute to perform a defined computer function.

Assembly (source) code - Symbolic labels, opcodes, and operands that are ordered in succession to define a logical procedure which can be assembled to produce executable machine instructions.

Opcodes - Defined symbols that assemble directly as 1 to 3 bytes of machine instruction. The symbols are meaningful descriptors of the machine function to be performed during program execution.

Label - A user defined symbol that corresponds to the storage address of the following opcode. Labels are used to define points for transfer of program execution which normally proceeds in a sequential manner.

Operands - Symbolic references to registers, labels or constants that are used to completely define the function specified by the opcode.

Pseudo-opcodes - Mnemonics that direct the assembly of the source code. They can allocate memory, define constants or affect control of the assembly procedure.

Execution storage - The physical memory space where an assembled program can execute. Absolute assembly generates machine instructions that will only execute correctly in memory space that was defined during assembly(defined by ORG pseudo op).

Program storage - The physical memory where program machine code is stored during assembly. The program will not necessarily execute correctly at this location unless assembly defined the program storage to be the same as execution storage(no ORR psuedo op given).

Debug - The process of testing a program to remove logic errors(bugs) by analyzing its execution(performance).

Patch - Correcting a program by changing machine instructions during debugging.

### Symbolism used in Manual

<CR> - Type a carriage return.
<LF> - Type a line feed.
<Control Z> - Type a Control Z not the individual characters.
<Escape> - Type an escape(TTY key usually has ESC on it).
<Tab> - Type a tab(Control I).

Anything enclosed in square brackets(i.e. [ ] ) is optional.
A Q following a number indicates the number is octal.

## II.   System Monitor

The system monitor has been designed to load and execute absolute programs, have a flexible I/O system for supporting all MITS peripherals, allow handlers for non-standard peripherals to be added, and has limited debugging capability.

Their are three tables within the monitor that make this flexibility possible. The names of all programs that have been read in is stored in the program name table(PTL), all open I/O symbolic names, flags, and other information are contained in the I/O table, and the hardware table which contains addresses of the device handlers.

### Hardware Table

This table has room for 5 entries, 4 of which are defined for standard MITS I/O devices. The four default device names and uses are:
    TY - Console terminal
    AC - Audio cassette
    EB - Edit buffer read
    TR - Mits high speed paper tape reader
The exact structure of this table and instructions on how to modify it is given in Appendix D.

### I/O Table

The I/O table has room for 7 symbolic device names to be open simultanously. The five names used by PKG II programs are automatically open, leaving room for 2 names to be set up for user programs if needed.


        TTY - Monitor, Editor and Assembler command I/O device name
                Open for echoing,tabbing,and ASCII mode
        LST - Editor, Debug, and Assembler list on LST
                Open for tabbing and ASCII mode.
        FIL - file I/O for Editor and Assembler
                Open for echoing and ASCII mode.
        ABS - program loading and file searching
                No options
        ALT - used by Editor's alter command
                Open for tabbing

```
I/O DRIVERS              **********   **********   **********   **********        U
These drivers            *        *   *        *   *        *   *        *
perform all monitor      *  TTY   *   *  ACR   *   *  EDIT  *   *  HSR   *
input/output             * DRIVER *   * DRIVER *   * BUFFER *   * DRIVER *
(CHECK STATUS AND        *        *   *        *   * DRIVER *   *        *
INPUT OR OUTPUT A        *        *   *        *   *        *   *        *
CHARACTER ).             **********   **********   **********   **********
                             A            A            A            A
                             A            A            A            A
                             A            A            A            A
                             A            A      >>>>>>>>>A            A
                             A<<<<<<<<     A      A                     A
HARDWARE TABLE               A       A    A      >>>>>>>>>>>>>>>>>A
                             A       A    A      A
                             A       A    A      A   *************
This table contains the hardware  A  A    A      A   *           *
driver names and pointers to the  A  A    A      A<<<<*   "TR"    *
proper device drivers.       A       A    A          *_____*
                             A       A    A          *           *
                             A       A    A          *           *
                             A       A    A<<<<<<<<<<<*   "EB"    *
                             A       A               *_____*
                             A       A               *           *
                             A       A               *           *
                             A       A<<<<<<<<<<<<<<<<*   "AC"    *
                             A                       *_____*
                             A                       *           *
                             A                       *           *
                             A<<<<<<<<<<<<<<<<<<<<<<<<*   "TY"    *
                                                     *           *
                                                     *************
                                                          A
                                                 >>>>>>>>>>A
I/O TABLE                                        A
                                                 A   ****************
This  table  contains  symbolic                  A   *              *
device  names  along  with  their                A<<<<*  TTY  - E,A,T *
flags  and hardware table pointer,               A   *              *
set by an  OPN  command.   In  the               A   *_____*
table  shown  at  left the default               A   *              *
flags and pointers are shown.                    A<<<<*  LST - T,A    *
                                                 A   *              *
                                                 A   *_____*
                                                 A   *              *
                                                 A<<<<*  FIL - E,A    *
                                                 A   *              *
                                                 A   *_____*
                                                 A   *              *
                                                 A<<<<*  ABS -        *
                                                 A   *              *
                                                 A   *_____*
                                                 A   *              *
                                                 A<<<<*  ALT - T      *
                                                     *              *
                                                     ****************
```

The block diagram of the I/O table shows all pointers from the symbolic device names going to the "TY" entry of the Hardware table. This is because all are open to "TY" when the monitor first comes up and can be changed at any time by giving an OPN command.

In order to support a non-standard I/O device, a driver would need to be written and the address of it patched into the hardware table as explained in appendix D. The device could then be used by all PKG II as well as user programs.

## Monitor Command Format

1. Execute a program

> The Monitor signals that it is ready for a command by printing 2 blanks and a question mark. A program is executed by typing its 3 character name followed by a carriage return. If the program has already been loaded, it will start execution immediately. If it hasn't, it will be searched for and loaded from symbolic device ABS. When the program finishes loading it will automatically start executing.
> Example:
> If you want to load the Editor from your TTY type:
>   ?OPN ABS,TY<CR>
>   ?EDT<CR>
> The OPN command would only be needed if ABS was not open to the TY.
> To load the Editor from the ACR make sure the last OPN ABS command was as follows:
>   ?OPN ABS,AC<CR>

2. Execution time options

> User programs can be passed execution options by the monitor if enclosed in parenthesis.
>   ?EDT(R)<CR>
> When the program requested is branched to the D&E register pair contain the address of the first character following the monitor command. In the above example D&E would point at the open parenthesis. The B&C register pair contains the address of the monitor status word, which contains the length of the command line. In the above example the status word contains the number 6.

3.  File searching

By typing a file name followed by a comma  and  a  type
designator, the monitor will search for the file on the
device ABS is open to, and return  after  the  file  is
skipped.
        S - ASCII file
        A - Absolute file
The main use of this command is to search for the  last
file  on  an audio cassette in order to write out a new
one.  All files on the the cassette should  be  of  the
same type as the file you are searching for.
   ?OPN ABS,AC<CR>
   ?AM2,A<CR>
This example would search for the absolute  file  "AM2"
on  the  ACR.   After  finding  the  file,  it would be
skipped and control returned to the monitor.

## Utility Programs

The monitor includes 9 utility programs:
```
        CLR - Deletes a program name from the PTL.
        OPN - Opens a symbolic device name to a physical device.
        CLS - Removes a symbolic device name from the I/O table.
        DEP - Used to change contents of memory locations.
        EXM - Used to print in octal the contents of memory.
        JMP - Causes the machine to start executing at any location.
        DMP - Dumps out memory in the checksum format.
        NUL - Sets the number of nulls to output after a <CR>.
        CNS - Console command
```
All numbers typed as parameters to these programs should be octal.

1.   OPN

This program is used to assign different physical devices to a symbolic name making programs device independent.
   ?OPN ABS,TY<CR>
This command opens the symbolic name "ABS" to the teletype.   In this type of OPN the mode default would be assumed.   These are:
        No echo - Don't echo input.
        Absolute - All 8 bits of every read are transmitted.
        Tabs - A Control I is output.
The options are:
        E - Echo all input on TTY
        A - ASCII mode, high order bit of characters masked to 0 and the line format described in Appendix F is recognized.   Even parity is generated on output.
        T - Tab control.  Spaces are printed to force cursor into a column that is an even multiple of 8 from the left margin.  The tab character is Control I(11Q).
The options are specified at the end of the command seperated by commas.   The order that they occur is irrelevent.  If the symbolic name is already open when another command to open it is given, the symbolic name is reopened according to the new command.

2.   CLS

Closing a symbolic name removes the name from the I/O table making the name unavailable until it is reopened.
   ?CLS ABS<CR>
In this example "ABS" is now closed meaning that no program should be executed that reads or writes on "ABS" .  If an I/O operation is attempted the program will abort, causing the monitor to print an error message.

3.  CLR

This command deletes a program name from the PTL.
    ?CLR EDT<CR>
After giving the above command
    ?EDT<CR>
will cause the monitor to try to reload the EDITOR.

4.  DEP

This command lets the programmer modify memory
locations directly from the monitor. The format is:
    ?DEP ADDRESS<CR>
All octal numbers typed after this command will be
deposited starting at the address given. Typing a
Control Z returns control to the Monitor.
Example:
    ?DEP 5000<CR>
25<CR>
50<CR>
175<CR>
<CONTROL Z>
    ?

Location 5000 would now contain an octal 25,5001 a 50
and 5002 a 175.

5.  EXM

The format of the examine command is as follows:
    ?EXM ADDR1[,ADDR2]<CR>
The contents of memory locations are dumped out in
octal starting at the location specified by ADDR1,
through ADDR2. If ADDR2 is not given or is = ADDR1
only the location at ADDR1 is printed.
Example:
    ?EXM 100<CR>
303
    ?EXM 100,101<CR>
303
153
    ?EXM 100,0<CR>
303
    ?

6. JMP

The format of the jump command is as follows:
?JMP ADDRESS<CR>
The jump command causes the machine to start executing
at the address given in the command. All registers are
loaded from the save area prior to jumping.
Caution: Make sure SP register will be loaded with
address of memory that can be used for a stack. For a
further explanation of the save area see section 8 of
the Program Monitor Calls.

7. DMP

The format of the dump command is as follows:
?DMP [NAME,]ADDR1,ADDR2[,START ADDRESS]<CR>
The name field of the command should contain the 3
character name of the program followed by any comments.
The only restriction is this field should not contain a
comma. ADDR1 is the address of the first location to
be dumped and ADDR2 is the address of the last to be
dumped. If a start address is not specified the
monitor will automatically be returned to when the tape
is loaded.
Example:
You want to dump a program located at 5100Q - 5352Q,
name it TST, put in a comment indicating it is the
first draft of the program and have it start executing
at location 5100Q.
?DMP TST REVISION 0,5100,5352,5100<CR>

8. NUL

The format of the NUL command is as follows:
?NUL NUMBER<CR>
This causes the number of nulls specified to be output
after every carriage return written in the ASCII mode.
It is adviseable to use this command before saving long
program files on cassette or paper tape as this insures
no characters will be missed during assembly.
Example:
Before saving a program on audio cassette give the
following command:
?NUL 4<CR>

9. CNS

The console command is used to switch between terminals
connected to your Altair. Before giving a console
command, set the sense switches to indicate the type of
I/O board to switch to( See Appendix E ). If the I/O
port address is not the same as the default listed in
the terminal options chart, flip sense switch 8 up and
deposit the I/O port address at location 572Q.

Program Monitor Calls

     The following section decribes how a user written program  can  use
features  of  the  Monitor to free him of the need to write I/O handlers
for each program he writes.
     Before any call to the monitor is preformed the B&C  register  pair
must  contain the address of a monitor control block.  A monitor control
block is used to specify the operation to be performed, symbolic  device
to  use,  etc.  The  monitor  is  called  by  executing a CALL IO.  All
registers are restored before returning to the calling program, and  the
monitor control block is left unchanged.  The example program at the end
of the assembler programming section uses monitor I/O.

```
    1.  READ
                    LXI     B,RDPKT ;LOAD B&C WITH ADDRESS OF THE
                                    ;MONITOR CONTROL BLOCK
                    CALL    IO      ;CALL THE MONITOR
        - DONT PUT MONITOR CONTROL BLOCK NEXT IN YOUR PROGRAM -
             RDPKT: DB      20Q     ;OPERATION CODE FOR READ
                    DB      "SDN"   ;SYMBOLIC DEVICE NAME
                    DW      INBUF   ;ADDRESS OF THE BEGINNING OF
                                    ;THE INPUT BUFFER
                    DW      80      ;MAXIMUM NUMBER OF CHARACTERS
                                    ;TO BE READ IN
                    DW      STAT    ;ADDRESS OF STATUS WORD
                    DW      END     ;END OF FILE RETURN ADDRESS
        After the read has been completed the status word
        contains the number of bytes read in.

    2.  WRITE
                    LXI     B,WRPKT ;LOAD B&C WITH ADDRESS OF THE
                                    ;MONITOR CONTROL BLOCK
                    CALL    IO      ;CALL THE MONITOR
             WRPKT: DB      22Q     ;OPERATION CODE FOR WRITE
                    DB      "SDN"   ;SYMBOLIC DEVIE NAME
                    DW      OUTBUF  ;ADDRESS OF THE OUTPUT BUFFER
                    DW      80      ;NUMBER OF BYTES TO WRITE OUT
                    DW      STAT    ;ADDRESS OF STATUS WORD
        When the monitor returns the status word  contains  the
        number of bytes output.

    3.  OPEN
                    LXI     B,OPNPKT ;LOAD B&C WITH ADDRESS OF THE
                                     ;MONITOR CONTROL BLOCK
                    CALL    IO       ;CALL THE MONITOR
            OPNPKT: DB      63Q      ;OPERATION CODE FOR OPEN
                    DB      "SDN"    ;SYMBOLIC DEVICE NAMED TO BE
                                     ;OPENED
                    DB      "TY"     ;DEVICE TO BE OPENED TO
                    DB      XXX      ;HARDWARE CONTROL BYTE
        The hardware control byte specifies  echo  control  and
        and ASCII or absolute read mode.
        Bit 1 - 1 for ASCII read mode, 0 for absolute
        Bit 2 - 1 for input echo, 0 for no echo
        Bit 3 - 1 for tabs to be expanded, 0 for no expansion
```

4. CLOSE

```
            LXI     B,CLSPKT ;LOAD B&C WITH THE ADDRESS OF
                             ;THE MONITOR CONTROL BLOCK
            CALL    IO       ;CALL THE MONITOR
   CLSPKT:  DB      62Q      ;OPERATION CODE FOR CLOSE
            DB      "SDN"    ;SYMBOLIC NAME TO BE CLOSED
```

5. ERROR

```
            LXI     B,ERRPKT ;LOAD B&C WITH ADDRESS OF THE
                             ;MONITOR CONTROL BLOCK
            CALL    IO       ;CALL THE MONITOR
   ERRPKT:  DB      60Q      ;OPERATION CODE FOR ERROR
                             ;HANDLING ROUTINE
            DB      "X"      ;ONE CHARACTER TO BE OUTPUT AS
                             ;ERROR MESSAGE
```
The character specified followed by a # sign will be echoed by the monitor instead of the next 2 characters that would normally be echoed.

6. PASS PROGRAM NAME

```
            LXI     B,PASPKT ;LOAD B&C WITH ADDRESS OF THE
                             ;MONITOR CONTROL BLOCK
            CALL    IO       ;CALL THE MONITOR
   PASPKT:  DB      61Q      ;OPERATION CODE FOR PASS NAME
                             ;NAME ROUTINE
            DB      "PRG"    ;3 CHARACTER PROGRAM NAME
            DW      PRG      ;START ADDRESS OF PROGRAM
```
The 5 bytes of name and address are copied into the PTL and the program jumped to.

7. FIND ASCII FILE

```
            LXI     B,FFPKT  ;LOAD B&C WITH ADDRESS OF THE
                             ;MONITOR CONTROL BLOCK
            CALL    IO       ;CALL THE MONITOR
   FFPKT:   DB      65Q      ;OPERATION CODE FOR A FIND FILE
            DB      "SDN"    ;SYMBOLIC DEVICE TO SEARCH ON
            DB      "FIL"    ;FILE TO BE SEARCHED FOR
```
This call causes the monitor to search for the named program on the physical device the symbolic device name is open to and return to the calling program as soon as it is found.

8. Returning to the Monitor

When a program has finished its job and wishes to ruturn to the Monitor, a JMP MON will be needed. All registers are saved in the register save area and the stack pointer it reloaded to delete anything left on the stack. The address of the monitor is on the stack when a program is executed by the monitor, so if your program has left nothing on the stack, a return instruction can be used to return to the monitor.

The symbols MON and IO are permenant equates within the assembler. The actual locations are - MON-100Q, IO-103Q

When a program returns to the monitor either by jumping to location 100Q, examining location 100Q and pressing RUN , or by typing a Control C all registers are stored in the register save area. The order they are saved in is given below.

```
571     FLAGS
572     A - REGISTER
573     C - REGISTER
574     B - REGISTER
575     E - REGISTER
576     D - REGISTER
577     L - REGISTER
600     H - REGISTER
601     SP - REGISTER (LOW BYTE)
602     SP - REGISTER (HIGH BYTE)
603     PC - REGISTER (LOW BYTE)
604     PC - REGISTER (HIGH BYTE)
```

If a program is waiting for input from the TY and a Control C is typed, the PC stored in the save area is the address of the instruction following the monitor call.


Note - 30 octal bytes of stack have been allocated for user program use. If more stack space is needed, a LXI SP instruction will be needed in your program to set up its own stack.

## III. Assembler Operation

### Introduction

Typical assemblers process source code by reading the same source code 2,3, or 4 times to produce a load tape that must then be loaded before executing the program. An assembler that requires that type of procedure is extremely cumbersome for users with paper tape or cassette magnetic tape input. Off-line storage is always required for assemblies of this type. Further, high speed storage is desireable due to the extensive I/O required during processing.

The Mits loading assembler was designed to process source code directly into memory for immediate execution or to produce an absolute load tape for later execution in the space occupied by the assembler. The source code is processed only once, thereby producing executable code in a minimum amount of time. Significant improvement in program development time is achieved, especially for users with program input rates under 100 characters/second. Furthermore, the assembler is still resident in memory with the user program, so it can be used to patch program errors during debugging. Since the patches can be entered in symbolic source code and labels can still be assigned to correct execution sequences, fewer errors are introduced in the debug process. thus, complete programs can be input and developed with input from only an ASCII keyboard and a minimal amount of memory(the Monitor and Assembler require approximately 5k bytes).

The loading assembler allows direct assembly of programs to any unused memory space or indirect assembly generating a load tape for programs that need to reside in memory being used during the assembling process. Program modules(parts) can be developed and debugged seperately, then assembled with all source errors corrected for off-line loading to other program space. The modules can be linked during assembly using the preserved symbols from previous assemblies or by defining names for referenced locations in other modules. Source programs are input from a Monitor-defined device called "TTY" and all selected output is to device "LST". Input can be selected from a source file that was created by the Editor. The files can be assembled in any sequence selected.

## Assembler Options

The assembler was designed as a module of the Mits operating system to be loaded by the system monitor. The assembler is loaded from the device ABS is open to when the assembler is executed for the first time. The following execution options are possible when ever starting the assembler.

                              ASM(A,S,P)

          Where:
                    P=Preserve symbols entered during previous assembly(s).
                    Used for symbolic patches and program additions

                    S=Symbol table listing wanted at end.  All defined
                    names and label symbols with corresponding program
                    addresses and the next program address($) are output

                    A=Absolute tape dump wanted at end.  Binary output  for
                    Monitor loading is output to "ABS"

All output begins at the first storage address that was defined  by the  first ORG( or ORR when used ) and continues to the current address. Program addresses are used for the absolute tape dump and defined symbol listings.
Warning*** If assembly is begun without the P option the symbol table is cleared and is not recoverable.


## Assembler Psuedo Op's

FILE Psuedo Op

The file input psuedo op  forces  source  input  to  be  read  from symbolic device "FIL".   If a file name is given as an operand, a file with that header is searched for before processing  any  input.   The source file  must end with a Control Z or EOA psuedo op so that control is restored to the  Monitor  at  the  end  of  file.   Assembly  can  be continued by entering the Assembler with the P-option.
Example:
          FILE     TWO        ;INPUT FILE "TWO" FROM CASSETTE

END Psuedo Op

All of the entry options in the group(S,A) are performed each  time an END pseudo-psuedo  op  is encountered.  The END statement will also produce a listing of all undefined symbol  names  with  program  storage locations  that reference the symbol.  If the A-option was selected, the first 3-letters of the operand define the program name  when  loaded  by the  Monitor.  Up to 77 characters following can be used to document the program(Revision, Date, etc.)
Example:
          END      PRG        ;IF THE A OPTION WAS SPECIFIED
                              ;THE PROGRAM WOULD BE DUMPED

;WITH NAME PRG

EOA Psuedo Op

The Assembler will return control to the Monitor when an End Of Assembly(EOA) pseudo-psuedo op is encountered. The Monitor prints a prompt to indicate it is in control.

Memory Allocation

The user must understand the way that memory is used during the assembly process to avoid errors and to use available memory in an efficient way. The diagram in Appendix B illustrates the relative storage used during assembly.

The user must estimate the symbol space needed for each assembly before defining the first storage location. It should be apparent that short symbols and few labels or names will increase the space available for user program storage. A rule of thumb for estimating symbol table space is to reserve 1 byte of symbol table space for each statement in the program.

ORG Psuedo Op

This psuedo op is required to be the first statement of all programs. It defines the memory your program will run in and and where the Assembler should store it while the Assembler is running.

ORR Psuedo Op

If the address of the start of the program given by the ORG statement does not allow enough space for the symbol table, an ORR statement will be needed to set the address the program should be stored at during assembly. Since the symbol table is built from the end of the Assembler to this address, its maximum size can be set with this psuedo op.
Example:
  ?ASM<CR>


*ASM*

```
        ORG     5100Q   ;WANT PROG TO LOAD AT 5100Q
        ORR     17000Q  ;SINCE ASSEMBLER IS AT 5100Q
                        ;A PLACE TO SAVE THE PROGRAM
                        ;DURING ASSEMBLY MUST BE SET UP
```

DS PSUEDO OP

STORAGE IS ALLOCATED WITH THE DS PSUEDO OP. THE VALUES OF BYTES IN THE SPACE ARE NOT CHANGED AND SHOULD NORMALLY BE PRESET DURING EXECUTION PRIOR TO USE. THE OPERAND MUST BE A DEFINED SYMBOL(EQU OR SET TO A CONSTANT) OR A CONSTANT VALUE. A LABEL SYMBOL DEFINES AN ADDRESS WHICH SHOULD NOT GENERALLY BE USED FOR A STORAGE OPERAND. A DS PSUEDO OP IS GENERALLY PRECEDED BY A LABEL USED TO REFERENCE THE STORAGE ALLOCATED DURING PROGRAM EXECUTION.
EXAMPLE:
LABEL:  DS      20      ;THIS RESERVES 20 BYTES OF MEMORY

DW Psuedo Op

An address word or two byte quantity is preset(assigned during assembly) by using the DW psuedo op. The 2-byte value is stored with the least significant byte in the first memory address and the most significant byte in the next higher memory location. This feature is the same as all 2-byte operands for machine opcodes(i.e. JMP, LXI, etc). This arrangement is convenient because it allows byte references to the least significant byte using the same label as a word reference. Multiple operands are allowed and must be separated with a space or comma.
Example:
LABEL:  DW      LOC     ;THIS STORES THE 2 BYTE ADDRESS OF LOC

DB Psuedo Op

All bytes constants are defined by using the DB psuedo op Multiple operands can be used. All operands define one byte of storage except string or literal constants which are stored as one ASCII character per byte. Each operand must be separated by a space or comma.
Example:
        DB      0       ;STORES THE CONSTANT 0 IN 1 LOCATION
        DB      "THIS IS A STRING CONSTANT"
                        ;THE ABOVE STRING WOULD BE STORED 1 CHARACTER
                        ;TO A BYTE, SO IT WOULD TAKE 25 MEMORY
                        ;LOCATIONS

DC Psuedo Op

The define character psuedo op is used to define literal constants of determinable length. All characters except the last have their high order bit masked to zero, but the last character has it set to one. The last character can then be found by searching for a character with its high order bit on.
Example:
        DC      "AB"    ;STORES THE CHARACTERS IN 2 CONSECUTIVE
                        ;MEMORY LOCATIONS WITH THE HIGH ORDER BIT OF
                        ;THE LAST CHARACTER TURNED ON.

EQU Psuedo Op

A symbol can be defined prior to use by assigning it a value equal to a specified constant or another label that has already been defined. A symbolic name(not a label) is defined by using the EQU psuedo op and a defined operand. The EQU psuedo op cannot be used to change the value assigned to a name. Refer to the SET psuedo op.
Example:

```
ONE       EQU     1       ;THIS SETS THE VALUE OF ONE = TO 1
          MVI     A,ONE   ;THIS WOULD NOW BE THE SAME
                          ;AS      MVI     A,1
```

SET Psuedo Op

A name can be changed or reassigned by using the SET psuedo op in the same manner as the EQU psuedo op. It can also be used to assign a value the first time a name is defined.

BEG Psuedo Op

The operand of the BEG psuedo op in a program sets the begin execution address, output by the Assembler during an absolute dump. If the BEG psuedo op is not found, a start address of 100Q will be assumed, causing a return to the Monitor after the program loads.

RUN Psuedo Op

The operand of the RUN psuedo op is returned as a program name along with the address from the latest BEG statement to be entered into the PTL. The address of the BEG statement is then branched to. This psuedo op should only be used during on-line assembly(no ORR statement).

## IV. Assembler Programming

Assembly programs include symbolic names,constants, opcodes and comments in sequential statements that are converted by the Assembler to produce executable machine instructions. Each line or program statement of source code must follow certain rules that govern the acceptable structure of the program. If they are not observed, assembly or execution errors will occur. This section will define the form that is acceptable to the Mits Loading Assembler.

Character Set

The entire 128-character ASCII character set is acceptable but all opcodes are defined in capital letters. Any combination of characters beginning with a non-numeric character can be used for statement labels or symbolic names. The maximum length for these symbols is 255 characters, but to minimize symbol table length, they should be kept as short as possible.

Constants

Constants can be used whenever an operand is required. All constants begin with a numeric character and can end with an alphabetic character that defines the radix of conversion. If the last character is numeric, the conversion defaults to decimal. Legal conversions are as follows:

```
12340    OCTAL
5678Q    OCTAL(8 CONVERTS AS 0)
12345D   DECIMAL
0ABCD    DECIMAL(A CONVERTS AS 1,ETC.)
057EFH   HEXIDECIMAL
```

NOTE: Values are first masked leaving only the significant binary quantities, thus alphabetic conversions are legal. Byte values are set equal to the converted value using modulus 256 arithmetic. Similarly, overflow of 16-bit constants(words) during conversion is ignored.

String or literal constants are defined by enclosing all characters in " symbols. The " symbol cannot be defined in a string constant. Example:

```
DB       "THIS IS A MESSAGE"      ;THIS IS A CONVENIENT
                         ;WAY TO STORE A MESSAGE FOR OUTPUT
                         ;DURING PROGRAM EXECUTION
```

WARNING****** Only one character should be used if a single byte operand is required(i.e. MVI A,"ABC" will store 4 bytes).

## Statement Structure

The assembly source statements may include any of the following in the order given:

1. Symbolic label of any length terminated by a colon(:). The symbol can include any ASCII character except delimiters(Space, TAB, or Comma) in any combination including instruction opcodes. The following symbols are predefined values.

```
         $=Next program byte address
following are only valid byte(not word)values.
         B,C,D,E,H,L,M,A=0,1,2,3,4,5,6,7 respectively
         SP and PSW=6
```

2. A name is the same as a label except that a terminating colon is not used. A name is used in place of a label and remains undefined until a defining pseudo opcode is encountered (i.e. Equ, SET).

3. Opcode(s) or pseudo opcode(s) with required operands. All opcodes that are defined in the Altair 8800 Operators Manual and All pseudo-op's defined in section III are acceptable to the Assembler.

4. Comments are used to document the source code but are not required by any statement. Comments begin with a semi-colon(;) which terminates assembly of all following ASCII characters on the line. Lines that begin with a semi-colon contain only comments.

## Statement Options

All register pair instruction operands can reference either of the two 8-bit registers in the pair.
Thus--
POP A is the same as POP PSW
LXI L is the same as LXI H
DCX C is the same as DCX B


Multiple instructions can appear on the same line of source code. This feature can be used to minimize the number of characters on a source tape and in some cases improves the program readability.
        MOV B,H MOV C,L
        RAR,RAR


The delimiters SPACE, TAB, or COMMA can be used anywhere in the line to improve readability.

## Statement Formats

```
*********************************************************************
*                                                                   *
*[ LABEL: ]     MNEMONIC         [ OPERAND FIELD ]       [ ;COMMENTS ]*
*       ( THIS FORMAT IS USED FOR ALL STATEMENTS EXCEPT EQU AND SET )  *
*                                                                   *
*NAME           MNEMONIC         OPERAND FIELD           [ ;COMMENTS ]*
*         ( THIS FORMAT IS USED FOR EQU AND SET STATEMENTS )          *
*                                                                   *
*********************************************************************
```

## Programming Tricks

The choice of opcode(s) to use to achieve a specific result is generally based on the generally accepted criteria that a program should use a minimal amount of space and should execute as rapidly as possible. Often one consideration is sacrificed in favor of the other but certain practices should always be avoided in favor of another which produces the same result at less cost in time or storage. The following practices are recommended:

1. Avoid the instruction sequence
   CALL Subroutine
   RET
in favor of
   JMP Subroutine
   The JMP statement will return to the same place without need for
the RET thus saving one byte of program storage.


2. Avoid CPI 0
              which requires two bytes of storage
   in favor of
ORA A
              which requires one byte of storage
   All flag bits are affected in the same manner without changing
the contents of the A-register.


3. Avoid PUSH B,POP H
              or similar register contents transfer
   in favor of
MOV H,B MOV L,C
              which executes in less time.


4. If a series of MVI 2-byte instructions are used followed by a
jump to the same address,less storage can be used by LXI B(DB 1)
replacing the jumps saving two bytes.

For Example:

```
AERR:    MVI     A,"A"                    AERR:    MVI     A,"A"
         DB 1                                      JMP ERR
BERR:    MVI     A,"B"                    BERR:    MVI     A,"B"
         DB 1                                      JMP ERR
```

   etc.

ERR:

A JMP to any MVI to set the "ERR" code will load the A-reg with the
character to be output and will skip over the rest by executing the DB 1
as LXI B,XXXX where XXXX is the two-byte MVI instructions.  the contents
of B&C will change.  Other register pairs can be used similarly.

## Example Program

A sample program and assembly are given to illustrate the operation of the assembler and use of the monitor calls for output.

The sample program will dump out any section of memory in octal as shown later in the example. This type of memory dump can be very useful in debugging programs. In order to use this program, change the addresses at locations FIRST and LAST to the address of the first and last memory location you want dumped.

```
    ?ASM(A,S)


*ASM*
          ORG     20000Q    ;SET LOCATION COUNTER
                            ;WILL NEED TO BE CHANGED IF ONLY 8K MACHINE
DUMP:     LHLD    FIRST     ;GET ADDRESS OF FIRST BYTE TO BE DUMPED
          XCHG              ;PUT ADDRESS IN D&E
NEWLN:    LXI     H,BUF     ;GET ADDDRESS OF OUTPUT BUFFER
          PUSH    H         ;SAVE ADDRESS
          LHLD    LAST      ;LOAD ADDRESS OF LAST BYTE TO BE DUMPED
          MOV     A,L       ;SUBTRACT LOW ORDER BYTES
          SUB     E
          MOV     A,H       ;SUBTRACT HIGH ORDER BYTE
          SBB     D
          POP     H         ;RESTORE H&L
          JC      MON       ;JUMPS OUT IF NO MORE BYTES TO BE DUMPED
          MOV     A,D       ;START CONVERSION OF ADDRESS TO ASCII
          RAL               ;ROTATE HIGH BIT INTO C
          MVI     A,0       ;ZERO OUT REST OF A BUT DONT CHANGE FLAGS
          RAL               ;ROTATE HIGH BIT INTO LOW ORDER POSITION
          ORI     60Q       ;OR IN ASCII 0
          MOV     M,A       ;STORE IN OUTPUT BUFFER
          INX     H         ;INCREMENT BUFFER POINTER
          MOV     A,D       ;PICK UP HIGH ORDER BITE AGAIN
          RAR               ;ROTATE BITS 4,5,6 INTO LOW ORDER POSITIONS
          RAR
          RAR
          RAR
          ANI     7         ;MASK OFF ALL BITS EXCEPT LOW THREE
          ORI     60Q       ;OR IN ASCII 0
          MOV     M,A       ;STORE IN OUTPUT BUFFER
          INX     H         ;INCREMENT POINTER INTO OUTPUT BUFFER
          MOV     A,D       ;PICK UP HIGH BYTE OF ADDRESS
          RAR               ;ROTATE BITS 1,2,3 INTO LOW ORDER POSITION
          ANI     7         ;MASK OFF ALL BITS EXCEPT LOW THREE
          ORI     60Q       ;OR IN ASCII 0
          MOV     M,A       ;STORE IN OUTPUT BUFFER
          INX     H         ;INCREMENT POINTER INTO OUTPUT BUFFER
          MOV     A,D       ;PICK UP HIGH BYTE OF ADDRESS
          RAR               ;SAVE LOW BIT IN THE CARRY FLAG
          MOV     A,E       ;PICK UP LOW BYTE OF ADDRESS
          CALL    LAST3     ;CALL SUBROUTINE THAT CONVERTS 3 DIGITS
```

```
            MVI     B,8       ;LOAD B WITH NO OF BYTES TO DUMP PER LINE
NXTNUM:     PUSH    H         ;SAVE H&L
            LHLD    LAST      ;LOAD ADDRESS OF LAST BYTE TO DUMP
            MOV     A,L       ;DO THE DOUBLE WORD COMPARE AGAIN
            SUB     E
            MOV     A,H
            SBB     D
            POP     H         ;RESTORE THE H&L REGISTERS
            JC      LNDN      ;JUMP TO ROUTINE TO FINISH UP IF DONE
            MVI     C,5       ;IF ANOTHER TO COME SEPERATE THEM BY 5 BLANKS
            CALL    BLANK
            XRA     A         ;SET THE CARRY FLAG TO 0
            LDAX    D         ;GET BYTE TO DUMP
            CALL    LAST3     ;CALL ROUTINE TO CONVERT 3 DIGITS
            INX     D         ;INCREMENT POINTER TO DUMP NEXT BYTE
CHKLN:      DCR     B         ;DECREMENT LINE BYTE COUNTER
            JNZ     NXTNUM    ;CONVERT NEXT NUMBER IF IT WILL FIT ON LINE
            LXI     B,OUT     ;GET ADDRESS OF MONITOR CONTROL BLOCK
            CALL    IO        ;WRITE OUT LINE
            JMP     NEWLN     ;JUMP TO WRITE OUT NEXT LINE
LNDN:       MVI     C,8       ;PAD LINE WITH 8 BLANKS FOR EACH NUMBER THAT
            CALL    BLANK     ;WOULD FIT
            DCR     B         ;DECREMENT NUMBERS THAT COULD FIT IN LINE
            JNZ     LNDN      ;LOOP UNTIL LINE FILLED WITH BLANKS
            LXI     B,OUT     ;GET ADDRESS OF MONITOR CONTROL BLOCK
            JMP     IO        ;WRITE OUT LINE
BLANK:      MVI     A,40Q     ;PUT A ASCII BLANK IN A
BL:         MOV     M,A       ;STORE IT IN THE OUTPUT BUFFER
            INX     H         ;INCREMENT THE OUTPUT BUFFER POINTER
            DCR     C         ;DECREMENT THE NUMBER OF BLANKS TO STORE
            JNZ     BL        ;LOOP UNTIL ALL STORED
            RET               ;RETURN TO CALLER
LAST3:      PUSH    PSW       ;SAVE BYTE TO CONVERT
            RAL               ;ROTATE CARRY AND HIGH 2 BITS TO LOW ORDER
            RAL               ;POSITION
            RAL
            ANI     7         ;MASK OFF ALL BUT LOW ORDER THREE BITS
            ORI     60Q       ;OR IN A ASCII 0
            MOV     M,A       ;STORE DIGIT IN OUTPUT BUFFER
            INX     H         ;INCREMENT THE OUTPUT BUFFER POINTER
            POP     PSW       ;POP BYTE TO CONVERT
            PUSH    PSW       ;SAVE FOR LATER ALSO
            RAR               ;ROTATE BITS 3,4,5 INTO LOW ORDER POSITION
            RAR
            RAR
            ANI     7         ;MASK OFF ALL BUT LOW THREE BITS
            ORI     60Q       ;OR IN AN ASCII 0
            MOV     M,A       ;STORE DIGIT IN OUTPUT BUFFER
            INX     H         ;INCREMENT OUTPUT BUFFER POINTER
            POP     PSW       ;POP BYTE TO CONVERT
            ANI     7         ;MASK OFF ALL BUT LOW THREE BYTES
            ORI     60Q       ;OR IN ASCII 0
            MOV     M,A       ;STORE DIGIT IN OUTPUT BUFFER
            INX     H         ;INCREMENT OUTPUT BUFFER POINTER
            RET               ;RETURN TO CALLER
```

```
OUT:     DB      22Q         ;MONITOR WRITE OPERATION CODE
         DB      "LST"       ;SYMBOLIC DEVICE TO WRITE ON
         DW      BUF         ;ADDRESS OF OUTPUT BUFFER
         DW      72          ;WRITE OUT 72 CHARATERS
         DW      STAT        ;ADDRESS OF STATUS WORD
STAT:    DW      0
FIRST:   DW      15100Q
LAST:    DW      15272Q
BUF:     DS      70          ;RESERVE 70 MEMORY LOCATIONS
         DB      15Q         ;ASCII CARRIAGE RETURN
         DB      12Q         ;ASCII LINE FEED
         BEG     DUMP        ;SETS ADDRESS OF PLACE TO START EXECUTING
                             ;PROGRAM
         END     DMP
```

```
 SENSE SWITCH 15 FOR DUMP
NOTE: AT THIS POINT THE PUNCH OR OUTPUT TAPE IS READIED FOR
OUTPUT OF THE PROGRAM IN ABSOLUTE BINARY FORMAT(APPENDIX A).
OUTPUT DONE


UNDEFINED SYMBOLS

SYMBOL TABLE
DUMP 020000
FIRST 020234
NEWLN 020004
BUF 020240
LAST 020236
LAST3 020165
NXTNUM 020067
LNDN 020133
BLANK 020154
CHKLN 020116
OUT 020222
BL 020156
STAT 020232
RUN DMP
015100    104    040    002    007    001    105    040    003
015110    007    001    110    040    004    007    001    114
015120    040    005    007    001    115    040    006    007
015130    002    123    120    040    006    007    003    120
015140    123    127    040    006    007    001    044    040
015150    000    000    000    000    000    000    000    000
015160    377    377    377    377    377    377    377    377
015170    377    377    377    377    377    377    377    377
015200    000    000    000    000    000    000    000    000
015210    000    022    114    123    124    272    032    016
015220    000    073    033    022    115    101    107    000
015230    000    016    000    073    033    061    104    115
015240    120    000    040    020    115    101    107    272
015250    032    110    000    073    033    335    017    065
015260    115    101    107    377    377    377    060    000
015270    043    007    122
```

## V.  Text Editor


The editor is used to create and modify source program files  using  the
four  editing  commands.   These  now include the alter command, used to
make corrections within a line, eliminating  the  need  to  replace  all
mistyped  lines.   The  insert,  delete,  and  replace command are still
included and have been improved to ease the job of modifying a program.


### Symbolic device names used by the Editor

Before running the editor, all symbloic device names used by it are open
to  the  TY,  and  need to be changed only if the device is not correct.
They are listed below along with  mode  information  needed  for  proper
operation.

         FIL - File I/O device name
                 A - ASCII read mode should be specified
                 T - Tabs should not be specified
                 E - Can be speified if the user wants a listing
                     Tabs will not be expanded
         ALT - Alter command I/O
                 A - ASCII mode should not be scecified
                 T - Tabs should be specified
                 E - Echoing should not be specified
         LST - Write and print command I/O
                 A - ASCII mode should be specified
                 T - Tabs should be specified
                 E - Echoing is not used during writing


### Buffer Area


     The first 2 K of memory following the  editor  is  allocated  as  a
buffer  to store the lines of text that you are editing.  If the size or
location of this buffer area need to be changed,  two  addresses  within
the  editor  must be changed.  Starting at location 5124Q is the address
of the beginning of the buffer and the address of the end of the  buffer
starts at location 5530Q.


### Loading the Editor

     Open symbloic device ABS to the AC or TY depending on whether  your
copy of the editor is on audio cassette or paper tape.  The editors file
name is EDT and is loaded by typing EDT<CR> .
Example:
To load the Editor from paper tape type:
  ?OPN ABS,TY<CR>
  ?EDT<CR>
(TURN ON PAPER TAPE READER)
START INPUT

*          (The astrisk it printed whenever the editor is ready for a
command)

     If after completing an edit and returning to the monitor  you  want
to use the editor again, type:
  ?EDT<CR>
START INPUT
*
If you would like to continue editing lines left in the  editors  buffer
area when you last exited the editor, use the R execution option.
Example.
  ?EDT(R)
*
Start input is not printed in this case and the buffer is not
reinitialized.  This feature is  especially  useful  when  assembling
directly from the editors buffer.


                Range and Line Number Specifications

     When a range is called for by an instruction, the following  syntax
is required.
Line Number,[ Line Number ]


                          Line Numbers

     Three types of line numbers are now recognized by the Editor.  They
are as follows:
 NUMBER(N)              THE N'TH LINE IN THE BUFFER.
 .[ + OR - NUMBER ]     RELATIVE ABOUT THE CURRENT LINE.
 *[ - NUMBER ]          RELATIVE ABOUT THE LAST LINE IN THE BUFFER
EXAMPLE.
*P*
Prints the last line in the buffer.
*P10
Prints the tenth line in the buffer.
*P.+10
Assuming this command was executed after the P10 command, line 20  would
be printed.


                        Editor Commands


I [Line Number]   Insert Command

The insert command causes the editor to enter the  insert  mode  at  the
line  specified.   After all lines to be entered have been typed, type a
Control Z to return to the command level of  the  editor.   If  no  line
number is typed all lines are inserted before the first line.

D Range                          Delete Command

Deletes all lines in the specified range.

R Range                          Replace Command

Deletes the lines in the range and enters the insert mode.


P [ Range ]                      Print Command

Prints all lines within the range or all lines in the buffer if no range
is given.  Line numbers are printed to the left side of the lines.

W [ Range ]                      Write Command

Same as print command except line numbers are not printed.

                          String Search Command

F[ String ] [ <ESCAPE> [ Line Number ] ]

The find command searches the buffer area starting  at  the  given  line
number, printing the first line it finds the string in.  If no string is
given, the string from the last find command issued is searched for.  If
no line number is typed, the editor starts searching at the current line
plus 1 (ie .+1).  The escape is optional when not typing a line number.

S                        Save File Command

Save command prints
FILENAME=
An optional 3 character file  name  is  typed  followed  by  a  carriage
return.  The editor responds by typing CHANGE SENSE SWITCH 15 as soon as
this message has finished printing turn on the device the file is to  be
dumped  on.   Change  the  position  of  sense switch 15 to indicate the
device is ready.  When all lines have been dumped, the editor returns to
the  monitor.   When  a  file  name  is given, a header block is written
containing the file name.  If no file name was typed, no header block is
output.

L                        Load File Command

The load command prints
FILENAME=
An optional 3 character file name  is  typed,  followed  by  a  carriage
return.   If  a file name is typed, a header block containing the proper
file name is searched for and the file following it is loaded  into  the
buffer.   If no file name is typed, all lines are loaded until an end of
file is read.  This command reads files from symbolic device FIL.

<ESCAPE>                    Backup Command

If an escape is typed to the editor a dollar sign is echoed and and the
current line minus 1(ie .-1) is typed.

<LF>                        Next Line Command

If a line feed is typed to the editor the current line plus 1(ie.+1)  is
printed.

E                           Exit Command

Causes the editor buffer read pointer to be reset to  the  beginning  of
the buffer, and returns to the monitor.

A Line Number               Alter Command

The alter  command  puts  the  Editor  into  Alter  mode,  allowing  the
programmer  to  change  lines  without  replacing  them.   The following
command characters are recognized but not echoed, and all  commands  can
be prefixed by a repetition factor of up to six digits.  This repetition
factor is refered to as "N" in the following description, and is assumed
to be one(1) if not given.


                       Alter Mode Commands

        D - Deletes the next n characters in line.  A slash is  output
        followed by all characters deleted and a closing slash.
        Example:
        The current line is
        BLAB:   MOV A,B
        *A.             Give Alter Command
        You type 3D and the editor responds by typing /BLA/  indicating
        that BLA has been deleted from the line.

        I - Inserts all characters typed after the I into the  line  at
        the  current  current  place  in  the line.  All characters are
        echoed and typing an <ESCAPE> will get you back  to  the  alter
        mode.

        R - Deletes the next N characters in the line  and  enters  the
        insert mode.

        S - Typing a S followed by any character will  cause  a  search
        for the N'th occurance of that character.

        Blank - Typing a blank will cause the next N characters in  the
        old line to be copied into the new line and be printed out.

        <CR> - Typing a carriage return will print out the rest of  the
        old  line,  inserting  the characters at the same time into the
        new line.  The old line line will the be replaced  by  the  new
        one and control is returned to the command level of the editor.

Q - Causes control to return to the command level of the editor without replacing the old line. This command is used to abort an alter during which you made a bad mistake.


Sample Edit

In the following example characters typed as alter mode commands that would not be echoed are inclosed in parenthses.
```
  ?EDT<CR>
START INPUT
*I<CR>
THIS IS A DEMONSTRATON
OF THE EDITOR.
<CONTROL Z>
*A1<CR>
(2ST)THIS IS A DEMONSTRA(<SPACE>)T(I)I(<ESCAPE>)<CR>
*W<CR>
THIS IS A DEMONSTRATION
OF THE EDITOR.
*A2<CR>
(3<SPACE>)OF (2D)/TH/<CR>
W2<CR>
OF E EDITOR.
A.<CR>
(SE)OF (R)/E/THE(<ESCAPE>)<CR>
*W<CR>
THIS IS A DEMONSTRATION
OF THE EDITOR.
*E<CR>
  ?
```

# DBG DOCUMENTATION VER 1.0

Package Summary:

The DEBUG package provides the user with the following capabilities:

1) Display memory locations, registers, or flags in any of several output I/O modes (including a symbolic instruction mode).

2) Modify memory locations, registers, or flags using corresponding input modes.

3) Set (or display or remove) breakpoints in the code to be debugged.

4) Enter and execute user code either
   A) at a specified location or
   B) automatically in such a way as to proceed properly from the most recently encountered breakpoint.

The commands accepted by DBG are 1-character commands or combinations of 1-character commands and data. These commands will be described in the remainder of this document.

Note: In the examples that follow, <CR> represents a carriage return character, <LF> a line feed, <RUBOUT> a delete character, <TAB> a tab (Control-I), <UPARROW> a ^.

Numbers may be typed in either as octal (the default) or as decimal by preceding the number with a number sign (#). Therefore #255 is equal to 377. It a single byte value is expected and a value greater than 377 is input, only the low order eight bits (byte) of the value is used.

RUBOUT:

If in the course of entering commands or data an error is made, a rubout character can be typed at any time to abort user input. DBG will type a question mark (?) and begin accepting commands on a new line.

I/O Modes:

Information is usually displayed and then re-entered in accordance with the current I/O mode. The I/O mode can be set by typing an ESCAPE or dollar sign (echoed $) followed by a character that specifies the I/O mode:

    $O  Specifies octal mode.
    $D  Specifies decimal mode.
    $W  Specifies double byte octal mode.
    $A  Specifies ASCII mode.
    $S  Specifies symbolic instruction mode.

### $O Mode

In octal mode, each location is typed as an octal value between 0 and 377. The line feed or up arrow characters always advance or back up the location counter by 1. Input is expected to be a one byte value between 0 and 377. Example:

```
10/ 0 55<CR>
10/ 55 #48<CR>
10/ 60
```

### $D Mode

Decimal I/O mode is identical to $O (octal) mode except that output is decimal and input is always assumed to be decimal (no number sign should precede input).

### $W Mode

In double byte octal mode ($W) the location and location plus one are interpreted as a double byte (16 bit) quantity. Assuming location 10 contains 0 and location 11 contains 1, then:

```
$W 10/ 400 200
10/ 200 <LF>
12/ 0
```

Line feed and up arrow always add or subtract two from the location counter.

The value re-entered in $W mode (the 200 in the above example) is interpreted as being a 16 bit (double byte) value and is stored in memory low order byte first, high order byte second.

### $A Mode

ASCII mode is used to type out or input ASCII information. When a location is opened in ASCII mode, the ASCII representaition of the byte store in there is typed:

```
$O 10/ 0 101<CR>
$A/
10/ A
```

When ASCII information in input, DBG expects the user to type a delimiter, a string of ASCII characters, and then a terminating delimiter which is the same as the initial delimiter.
Example:

10/ A "B"

This enters the character B into location 10. The delimiters(") are not stored in memory. Multi character strings may be entered:

10/ B "FOO"

(Note: One should not try to use the special characters <CR>, LF>, <TAB>, `"`, <ESC OR $>, <RUBOUT>, =, ;, !, ., +, -, / as delimiters as these characters have special meanings for DBG. Double quote and single quote should suffice for most string entry.) Typing slash (/) after a string has been entered will reopen the first location in the string:

10/ A "FOO"/
10/ F

Typing <LF> after a string has been entered will open the location after the last location stored in:

10/ A "FOO"<LF>
13/ Z

The only character which may not be entered in an ASCII string is <RUBOUT>. <RUBOUT> may be used to terminate the entry of an ASCII string. However, any characters that had been entered prior to the typing of the slash are still there.
10/ A "FO<RUBOUT>
?
10/ F <LF>
11/ O

Special note: The high order bit (D7) of data entered via $A mode will always be set to zero.


### $S Mode

Symbolic (instruction) mode is used to type out locations as if they were instructions, and to enter instructions into memory using their mnemonics. Example (suposing locations 7-16 octal contained 0):

$S 5/ NOP LXI H, 8192<LF>
10/ NOP MVI B,100<LF>
12/ NOP MVI M,0LF>
14/ NOP DCR B<LF>
15/ NOP JNZ 12<LF>
16/ NOP JMP 100<CR>

In this example, a short program has been entered to set the 64 decimal bytes starting at location 8192 decimal to zero. After it finishes, the program returns to the monitor by jumping to location 100 octal. Decimal numbers may be used in the address or immediate fields of an instruction by preceding them with a number sign (#). DBG may be used in the fashion demonstrated above to 'improvise' programs. After they have been written and debugged, the monitor DMP command may be used to

store them on cassette or paper tape. The symbolic I/O mode is often very useful in patching or changing instructions in existing programs to fix bugs temporarily before the source code is re-assembled.

<LF> in symbolic mode opens the location which is the current location plus the number of bytes of the instruction typed out (or just entered) -1.

<UPARROW> opens the current location minus the number of bytes of the instuction typed out ( or just entered). This may or may not be meaningful, as the previous intruction may not be the same number of bytes as the one just typed in or displayed.

The default mode (when DBG is first entered) is octal.

Slash:

A memory location can be displayed by typing its octal address followed by a "/". This address may be octal or decimal and is independent of the I/O mode. Thus

30/

or

#24/

Will cause the contents of octal location 30 to be displayed in the current I/O mode. In the case of symbolic I/O mode, up to 3 bytes (e. g. a JMP) may be displayed depending on the type of instruction found in the first byte. Registers can also be displayed by typing a "/" after their 1-character names. For example

L/

Will cause the contents of the 1 register to be displayed in the current I/O mode. (The value actually displayed is not actually the L register but a memory location used to maintain the user's L register while DBG executes.)

The flag register (condition codes) is displayed similarly by typing

F/

Since the contents of the flag register is usually interpreted as settings of the carry (C) zero (Z) sign (S), parity (P) and half carry (H) flags, a special type out mode has been provided so the user can display the flags in a meaningful fashion without having to interpret the octal value of the flags:

F/ 106 !ZP

In this example, the flag register was opened in octal mode.  In order to display which flags were set, the character ! was typed and the debug package typed back 'ZP' which meant that the zero and parity flags were set.  The exclamation character may be used to type out any loctaion in 'CONDITION CODE' format.  This can prove useful when examining condition codes that have been pushed on the stack by the 'PUSH PSW' instruction.

There is no corresponding method to enter condition codes symbolically. The user must re-enter any change he wishes to make in the current I/O mode (octal is suggested).  See the table below under the F register for the bit positions of the different flags.

The stack pointer may be displayed by typing:

S/

This will display the lower 8 bits of the stack pointer in the current I/O mode.  To display the high 8 bits, type linefeed.  Typing TAB (Control I) after opening the low eight bits of a register or location will open the location pointed to by the register pair or double byte memory pointer.  i.e.  to look at the byte pointed to by the H and L registers, type:

L/ 4 <TAB>
5004/ <The contents of this location>

In this example, H would have contained 12 octal.  A TAB is also useful when the I/O mode is symbolic and a 3-byte instruction has just been displayed.  The current location pointer will be set to the address part of the displayed instruction, and the contents at that new address will be displayed.  This feature permits simplified program tracing when jumps and calls are encountered.  If the last value displayed was not a three byte instruction displayed in symbolic mode, then TAB will open and display the location pointed to by the double byte (low byte first, high order byte second) which reside in memory starting at the current location (as in the previous example).

70/ JMP 5000 <TAB>
5000/ SHLD 4750

The user program registers are stored inside DBG when a breakpoint is encountered during the execution of a user program.  The order of the registers in memory is as follows:

```
REGISTER
NAME
-
F         (CONDITION CODES)
          BIT     MEANING (IF = 1)
          ---     ----------------
          0       CARRY
          2       EVEN PARITY (NUMBER OF ONES IN RESULT WAS EVEN)
          4       HALF CARRY FOR BCD OPERATIONS
          6       ZERO
          7       SIGN (ONE MEANS MSB OF RESULT WAS 1)
A
C
B
E
D
L
H
S         (STACK LOW EIGHT BITS)
          (STACK HIGH EIGHT BITS)
```

Thus, once a particular register is opened, linefeeds, and/or uparrows may be used to display the register above or below the one currently opened.


          DOT:

The address of the most recently displayed location is saved in a "CURRENT LOCATION POINTER". The location at that address can be redisplayed at any time (even after changing I/O modes) by typing

./

or simply

/

The dot (which can be optionally omitted) can be thought of as a symbol for the address of the current display location. This pointer can be offset in either direction using a + or - and a number. Thus:

.+5/

displays the contents 5 locations after the current one, and

-33/

displays the contents of the location 33 octal locations before the current one. (As before, typing of the "." is optional.)

Multiple subtractions and additions may be performed to calculate addresses or other sixteen bit (two byte) values:

```
100+20-30/
70/ 25
```

        The equal sign (=)

The equal sign may be used to type out the current value of calculation:

```
100+20-30=70 -#8=60
```


        Semicolon:

In general, whereas a slash can be typed at any time to display the current location in accordance with the current I/O mode, a semicolon can be typed at any time to display the current location in octal independent of current I/O mode. The I/O mode is not changed by a semicolon, but if location modifying information immediately follows the octal display, the input information will be accepted in octal. Thus if the current I/O mode were symbolic the octal equivalent of a symbolic instruction at location 100 could be examined easily with a semicolon as follows:

```
100/ MOV A,C;   171
```

        Line Feed and Up-Arrow:

A line feed (usually typed after some location has been displayed) causes the current location pointer to be advanced to the next location, and that location will be displayed. This permits rapid display of successive memory locations. If the current I/O mode is symbolic the current location pointer advances by the number of bytes in the instruction just displayed. Thus a rapid symbolic display of program segments is possible.

The up-arrow command acts similarly to a line feed except that it decrements rather than increments the location pointer.

        Location Modification:

Immediately after a location has been displayed it is subject to modification. (Susceptability to accidental modification at this point can be removed by typing a carriage return.) Input for modifying the location must conform to the current I/O mode. (Exception: After a semicolon as described above.) A failure to conform to the current I/O mode, or entry of uninter- uninterpretable data will result in rejection of the input data. (A question mark will be typed by DBG.) In general, spaces are always ignored on input and can be typed or omitted with no effect in any I/O mode.

The following special characters:

<CR>, <LF>, /, ;, <TAB>, +, -, <ESCAPE>, !, =, <UPARROW>

Will always cause termination of data input strings as they have special meaning to DBG.

A "/" can be used as a terminator to get an automatic feedback of the typed input data. In the following example, the I/O mode is symbolic.

./JMP 204 LXIB, 12 3/
200/ LXI B,123

Three bytes starting at location 200 are set by the above commands. The second line was typed entirely by DBG in response to the "/" terminator. This sequence checks both the correctness of the entered data (which at first looks questionable) and the previously uncertain value for the current location pointer.

If input is purposely terminated by a line feed, up-arrow, slash, or tab, the input will be processed and the appropriate new location will be displayed. Thus, for example, the following sequence demonstrates clearing of a small block of memory locations that previously all contained 377's:
30/ 377 0<LF>
31/ 377 0<LF>
32/ 377 0<LF>
33/ 377 0<LF>

In the above sequence DBG typed all but the initial "30/" and the repeated "0 <LF>"'S.

Breakpoints:

Breakpoints are set using the X command. For example:

30X

causes the first unused breakpoint to be set at location 30. Similarly,

.X

or just

X

will cause a breakpoint to be set at the current location pointer.

There is capacity for setting 8 different breakpoints numbered internally 0 thru 7. When an X command is executed the first free breakpoint is allocated to the breakpoint being set. If there are no free breakpoints, a question mark is printed, and one of the existing breakpoints must be deleted before a new one can be set. When any breakpoint is encountered the address of that breakpoint is always displayed for the user by DBG:

5 BREAK @1000

means that breakpoint number five was encountered at octal location
1000.

It is permissable to change the instruction at any breakpoint at anytime
while running DBG.

2 BREAK @1000
1000/ XRA A ORA A<CR>

If an RST instruction is executed which is the same RST used by DBG but
was not inserted by DBG as a breakpoint, DBG will type out the
breakpoint number as 10:

10 BREAK @205

It is possible to proceed from such breakpoints, but this kind of
conflict between user RST's and DBG RST's usually indictates that a user
RST service routine is not being executed and DBG is intercepting the
RST. Under these circumstances, the breakpoint RST should be changed so
it does not conflict with user RST's.

        Changing the RST used by DBG.

It may become necessary to change the RST used by the debug package to
another RST. to accomplish this you can use DBG itself to make a
modification which will allow you to set the breakpoint RST to any of
the eight possible RST's. Start looking for the first MVI instruction
in DBG by entering symbolic typeout mode and line feeding until you find
it.
```
$S       CR>            ;OPEN SYMBOLIC MODE
12722/   SHLD 4205 <LF> ;OR WHATEVER
ETC.
13000/   MVI A,377      ;DISPLAY RST SETUP INSTRUCTION
         MVI A,317  <CR> ;CHANGE IT TO RST 1.
13000G                  ;RESTART DEBUG PACKAGE
```

It is important to note that when DBG is started, it always initializes
the appropriate RST location (0,10,20,30,40,50,60,70 octal) to a JMP
instruction to the breakpoint handling code inside DBG. Thus, when DBG
is started initially, it will always clobber (store into) locations
60,61 and 62 octal with a JMP instruction.

Checking Breakpoints:

The Q command causes all program set breakpoints to be displayed.

Example (assuming DBG has just been started):
10X
20X
377X
Q
0@10
1@20
2@377

Each breakpoint is typed out by its number, an at (@) sign, and then the address where the breakpoint is set. Any breakpoints that are not mentioned by a 'Q' command are not in use.

Removing Breakpoints:

Typing Y (carriage return) will remove all breakpoints. Typing Y followed by <DIGIT> will remove breakpoint DIGIT>:
Y5
Y

Execution:

The 'G' or go command permits entry of user code at an arbitrary location, the address should be followed by a G. Thus:

30G

will cause execution to begin at octal location 30. As in other situations the current location pointer can be used in place of an address. Thus

.G

or simply

G

will cause execution to begin at the user address indicated by the current location pointer.

If a user program loops endlessly (a typical symptom is that no response is made to input) the debug package can be re-entered by stoping the program (either with Control-C if monitor interrupts are being used or by manually stoping the machine) and restarting DBG from the monitor or at its starting location.

### Proceeding from a Breakpoint:

If it is desired to proceed from the last encountered breakpoint, the single character command "P" can be used. Restrictions: This command cannot be used if no break point has yet been encountered during execution or user code. If this is tried, a question mark will be typed.

### Multiple Proceeds:

An number before a P has a different meaning than before a G. Such a number indicates the number of times a P command should be executed (the number of times that any encountered breakpoint should be ignored) before control is returned to DBG. Thus the command

30P

will cause execution to proceed in the user program until breakpoints have been encountered 30 (octal) times. This feature is especially useful in proceeding from a loop that contains a breakpoint.

### Typing out a Sequence of Locations:

The "T" command is used to type out a sequence of locations in the current I/O mode. The format of the command is:
X,YT

where X is the beginning address and Y is the ending address. For instance:

100,500T

would type out the first 256 bytes of the monitor in the current I/O mode.

### DBG System Documentation

The debug package resides directly above the monitor. whenever the debug package is entered, it saves the user registers in memory inside the debug package. It then replaces any breakpoints that may have been set with the original instruction contained in the location where the breakpoint was set. The RST location is then initialized to point to the DBG breakpoint service routine.

When a 'G' or 'P' command is typed, DBG replaces all instructions which have breakpoints set at their locations with the DBG breakpoint RST, and saves the original instruction in a table inside DBG so the original instruction may be restored if DBG is restarted or a breakpoint is encountered. Then all the original user registers are restored. If this is a 'P' command, a complicated sequence of operations are performed to correctly execute the instruction located at the breakpoint address. This is especially difficult for CALL's,and RST's, as the instructions are actually executed inside the debug package, and not at the breakpoint location. After simulation of the breakpointed instruction is finished, DBG jumps to the instruction after the one

breakpointed.   (or  in  the  case  of true conditional JMP's, CALL's or
RST's to the proper location).  When a breakpoint RST is  executed,  DBG
saves all the users registers and restores breakpointed instructions.

The debug package program name is DBG and is loaded by typing:
  ?dbgCR>
(RUN THE TAPE)
DEBUG

Debug starts at 5100Q and ends at 11577Q, allowing AM2 to be  in  memory
at the same time.

APPENDIX A

ABSOLUTE LOAD TAPE FORMAT

BEGIN/NAME RECORD

BYTE 1      125 OCTAL            BEGIN SYNC

BYTES 2-4 NAME                   PROGRAM NAME

BYTES 5-N COMMENTS               END STATEMENT DOCUMENTATION
                                 (I.E.PROGRAM REVISION AND DATE)

BYTE   N+1 15 OCTAL              TERMINTATES THE PROGRAM NAME
                                 RECORD

PROGRAM LOAD RECORD

BYTE 1      74 OCTAL             LOAD SYNC  BYTE

BYTE 2      0-377 OCTAL          NO. OF LOAD BYTES(N)

BYTE 3      L.S. BYTE            LOAD ADDRESS

BYTE 4      M.S. BYTE               "       "
            DATA BYTES

BYTE N+5  *                      CHECKSUM BYTE

    * THE CHECKSUM IS GENERATED BY ADDING W/O CARRY ALL BYTES
EXCEPT THE FIRST TWO.

END-OF-FILE RECORD

BYTE 1    170 OCTAL              PAPER/AUDIO CASSETTE EOF

  BYTE 2-3                       BEGIN EXECUTION ADDRESS

APPENDIX B

ASSEMBLY MEMORY MAP

MEMORY BLOCKS                          BOUNDARIES
----------                          TOP OF MEMORY
   USER SPACE
                                    FIRST WORD OF PROGRAM STORAGE
                                    (SET BY ORG OR ORR IF GIVEN)
   SYMBOL TABLE
   (CHAR. LENGTH
   +3 BYTES/SYMBOL)
                                    SYMBOL TABLE BUILDS UP FROM HERE
   TEMPORARY
   ASSEMBLER
   VARIABLES
                                    FIRST WORD OF VARIABLE STORAGE
   ASSEMBLER
   PROGRAM
   STORAGE
     (VERSION 1)
                                    FIRST WORD OF ASSEMBLER
                                    (5100Q)
   MONITOR
   TABLES &
   HANDLERS

   RESTART
   TRAPS
                                    BOTTOM OF MEMORY


        RECOMENDED MEMORY LAYOUT WITH EDITOR AND ASSEMBLER

MEMORY BLOCKS                          BOUNDARIES
----------                          TOP OF MEMORY
   EDITOR BUFFER
   AREA
                                    FIRST WORD OF BUFFER
   USER AREA

   SYMBOL TABLE

   ASSEMBLER
   (VERSION 2)
                                    FIRST WORD OF ASSEMBLER
                                    (11520Q)

                                    DEFAULT START OF EDIT BUFFER
   EDITOR
                                    EDITOR STARTS AT 5100Q
   MONITOR
                                    BOTTOM OF MEMORY

# Appendix C

## Assembler Errors

Error codes are the first two characters on the line following occurance of an error. The characters replace the characters that are normally echoed on a TTY or Compter terminal.

B#  No origin specified

D#  Double defined symbol

I#  Illegal operand
   Undefined byte symbol
   String not allowed
   Name value must be defined
   ORR or ORG must be defined value

N#  No name defined

O#  Memory overflow
   Program space not large enough

Q#  Symbol table overflow
   Program storage should begin at higher memory address

S#  Symbol undefined

## Monitor Errors

All monitor error messages are output inplace of the 2 spaces in the monitor prompt.
Example:
```
  ?OPN FIL,EV,A
H*?
```

A*  Attempt to store over monitor
   No program can load before 5100Q.

C*  Typing a Control C caused a return to the monitor.

D*  Name already in PTL
   Use the CLR utility to remove the program name from the PTL.

F*  I/O table full
   Use the CLS utility program to close an unused symbolic name
   freeing space in the table to open the name you need.

H*  Hardware device undefined
   An attempt was made to open a symbolic name to a nonexistant

hardware device.

L*       Load error.
         A checksum error occured while loading a program.

M*       Memory malfunction
         Memory not working or nonexistant.
         After storing into memory the stored byte did not compare
         exactly with the value stored. This is checked when the
         monitor loads a program.

N*       I/O device name not open
         Before trying to read from a symbolic device, that device
         should be opened.  See open command under monitor utility
         programs.

P*       Program table(PTL) full
         Use the CLR utility to clear the name of a program that is'nt
         needed any longer.

S*       Syntax error in command line

U*       Program name not in ptl

V*       Illegal operation
         An illegal operation code was given in a monitor control block.

Appendix D.

If you need to set up any special purpose I/O handlers, the following addresses may be useful.

    Console input interrupt routine - 4414Q
    Console input non-interrupt routine - 3436Q
    Console output routine - 3422Q
    Console tab counter - 4614Q
    ACR input routine - 4521Q
    ACR output routine - 4532Q
    ACR tab counter - 4617Q
    Edit buffer read routine - 3374Q

The structure of the monitor table that would need to be modified is as follows:

```
DB      "DN"     ;TWO CHARACTER DEVICE NAME
DW      DEVIN    ;ADDRESS OF DEVICE INPUT ROUTINE
DW      DEVOUT   ;ADDRESS OF DEVICE OUTPUT ROUTINE
DW      DEVTAB   ;ADDRESS OF DEVICE TAB COUNTER
                 ;THIS IS ONE BYTE OF STORAGE USED
                 ;TO STORE INFORMATION ON THE CURSOR POSITION
```

Their is room in the table for an entry to be added at location 220Q.

Example:

Suppose you want to have input from the ACR echoed on your terminal and to refer to it in an OPN command as the "AT". The following added at location 220Q would accomplish this.

```
        DB      "AT"
        DW      3527Q    ;THIS FORCES INPUT FROM THE ACR
        DW      3575Q    ;THIS CAUSES OUTPUT TO BE SENT TO THE CONSOLE
        DW      TABL     ;THIS ALLOWS THE TABS TO WORK CORRECTLY
TABL:   DS      1
```

If you want to use monitor I/O with a device not supported by Mits, this can be accomplished by writing your own handler and putting it in high memory. Input routines should return a character in the A register while output routines should output the character in the A register. The routine should check the status of the device, loop until ready, and perform any character conversions if needed(ie. BOUDOT to ASCII or ASCII to BOUDOT).

Example:

You have a high speed paper tape reader connected to I/O port 12 and 13. Its motion is controled by the output to port 13. A 1 turns the reader on and a 0 turns it off.

Put the following routine in unused memory(probably in the highest locations in your machine).

```
HSRD:   MVI     A,1      ;SET A = TO 1
        OUT     13Q      ;TURN ON READER
        IN      12Q
        RAR              ;ROTATE STATUS BIT INTO THE CARRY FLAG
```

```
        JC      TRMIN    ;WAIT FOR INPUT READY BIT TO GO LOW
        XRA     A
        OUT     13Q      ;SHUT READER OFF
        IN      13Q      ;INPUT THE CHAR FROM TERMINAL
        RET
AT LOCATION 220Q PUT THE FOLLOWING:
        DB      "PT"     ;DEVICE NAME
        DW      HSRD     ;ADDRESS OF READER INPUT ROUTINE
        DW
        DW               ;SINCE NO OUTPUT ROUTINE TAB COUNTER NOT NEEDED
```

To open ABS to the high speed reader in order to read  in  absolute
program tapes, use the following open command.
?OPN ABS,PT

Appendix E.


When the Altair is first turned on, there is random garbage in its memory. The monitor is supplied on a paper tape or audio CASSETTE. Somehow the information on the paper tape or cassette must be transfered into the computer. Programs that perform this type of information transfer are called Loaders.

Since initially there is nothing of use in memory, you must toggle in, using the switches on the front panel, a 20 instruction Bootstrap Loader. This loader will then load the MONITOR. SO, to Load the MONITOR, follow these steps:

1)      Turn the Altair ON.

2)      Raise the STOP switch and RESET switch simultaneously.

3)      Turn your terminal (usually a Teletype) to LINE.

Because the instructions must be toggled in via the switches on the front panel, it is rather inconvenient to specify the positions of each switch as up or down. Therefore, the switches are arranged in groups of 3 as indicated by the broken lines below switches 0-15. To specify the positions of each switch we use the numbers 0-7 as shown below:

```
                    Switches
                    --------
Leftmost        Middle          Rightmost               Number
--------        ------          ---------               ------
Down            Down            Down                    0
Down            Down            Up                      1
Down            Up              Down                    2
Down            Up              Up                      3
Up              Down            Down                    4
Up              Down            Up                      5
Up              Up              Down                    6
Up              Up              Up                      7
```

So, to put the octal number 315 in switches 0-7, the switches would have the following positions:

```
                        Switches
7       6   /   5       4       3   /   2       1       0
UP      UP  /   DOWN    DOWN    UP  /   UP      DOWN    UP

     3                  1                   5
```

Note that switches 8-15 were not used. Switches 0-7 correspond to the switches labeled DATA on the front panel. A memory address uses all 16 switches.

The Bootstrap Loader is the following program:

The following Bootstrap Loader is for users loading from paper tape
and using a REV 1 Serial I/O board.

Address/Data

| | |
|------|------|
| 000 | 041 |
| 001 | 256 |
| 002 | 017 |
| 003 | 061 |
| 004 | 022 |
| 005 | 000 |
| 006 | 333 |
| 007 | 000 |
| 010 | 017 |
| 011 | 330 |
| 012 | 333 |
| 013 | 001 |
| 014 | 275 |
| 015 | 310 |
| 016 | 055 |
| 017 | 167 |
| 020 | 300 |
| 021 | 351 |
| 022 | 003 |
| 023 | 000 |

The following 21 byte Bootstrap Loader is for users loading  from  paper
tape and using a REV 0 Serial I/O board on which the update changing the
flag bits has not been made.  If the update has been made, use the above
Bootstrap loader.

| | |
|------|------|
| 000 | 041 |
| 001 | 256 |
| 002 | 017 |
| 003 | 061 |
| 004 | 023 |
| 005 | 000 |
| 006 | 333 |
| 007 | 000 |
| 010 | 346 |
| 011 | 040 |
| 012 | 310 |
| 013 | 333 |
| 014 | 001 |
| 015 | 275 |
| 016 | 310 |
| 017 | 055 |
| 020 | 167 |
| 021 | 300 |
| 022 | 351 |
| 023 | 003 |
| 024 | 000 |

The following Bootstrap Loader is for users with the MONITOR supplied on an Audio Cassette.

```
000    041
001    256
002    017
003    061
004    022
005    000
006    333
007    006
010    017
011    330
012    333
013    007
014    275
015    310
016    055
017    167
020    300
021    351
022    003
023    000
```

88-PIO BOOTSTRAP

```
000    041
001    256
002    017
003    061
004    023
005    000
006    333
007    000
010    346
011    040
012    310
013    333
014    001
015    275
016    310
017    055
020    167
021    300
022    351
023    003
024    000
```

2 SIO BOOTSTRAP

```
000    076
001    003
002    323
003    020
004    076
```

```
005        021 (=2 STOP BITS
006        323 025=1 STOP BIT)
007        020
010        041
011        256
012        017
013        061
014        032
015        000
016        333
017        020
020        017
021        320
022        333
023        021
024        275
025        310
026        055
027        167
030        300
031        351
032        013
033        000
```

4 PIO BOOTSTRAP

```
000        257
001        323
002        020
003        323
004        021
005        076
006        004
007        323
010        020
011        041
012        256
013        017
014        061
015        034
016        000
017        333
020        020
021        346
022        100
023        310
024        333
025        021
026        275
027        310
030        055
031        167
032        300
033        351
034        014
```

035          000

MITS HIGH SPEED READER BOOT

```
000      257
001      323
002      020
003      323
004      021
005      323
006      022
007      057
010      323
011      023
012      076
013      014
014      323
015      020
016      076
017      004
020      323
021      022
022      076
023      016
024      323
025      023
026      041
027      256
030      017
031      061
032      051
033      000
034      333
035      020
036      346
037      100
040      310
041      333
042      021
043      275
044      310
045      055
046      167
047      300
050      351
051      031
052      000
```

So, to load the Bootstrap Loader:

4)      Put switches 0-15 in the Down position.

5)      Raise EXAMINE.

6)      Put 041 (the data for address 000) in switches 0-7.

7)      Raise DEPOSIT.

8)      Put the data tor the next address in switches 0-7
        (For address 001 this is 175)

9)      Depress DEPOSIT NEXT.

10)     Repeat steps 8-9 until the entire Loader is toggled in.


Next check that tne Bootstrap Loader is in correctly:

11)     Put switches 0-15 in the Down position.

12)     Raise EXAMINE.

13)     Check that lights D0-D7 correspond with the data that should  be
        in  address 000.  A light 'on' means the switch was up;  A light
        'off' means the switch was Down.  So  for  address  000,  lights
        D1-D4  and  D6-D7  should be off, and lights D0 and D5 should be
        on.

        If the correct value is there, go to step 16.
        If the value is wrong, go to the next step, 14.

14)     Put the correct value in switches 0-7.

15)     Raise DEPOSIT.

16)     Depress EXAMINE NEXT.

17)     If you have not checked  the  entire  Bootstrap  Loader,  Repeat
        steps 13-16 until you have.

18)     If you found a mistake,  go  back  to  step  11  and  check  the
        Bootstrap Loader again.

19)     Put the tape of the MONITOR into the tape reader.  Be  sure  the
        TAPE  is  positioned at the beginning of the leader.  The leader
        is the section of tape at the beginning with  6  out  of  the  8
        holes punched.
        If you are loading from Audio Cassette, put the cassette in  the
        recorder.  Be sure the tape is fully rewound.

20)     Put switches 0-15 in the Down position.

21)     Raise EXAMINE.

22)     There are six different Bootstrap Loaders, one for each of the
        six types of I/O boards listed in the load option chart. Be
        sure that you use the correct one for your particular board.

                LOAD OPTIONS

| OCTAL LOAD DEVICE MASKS | SWITCHES UP | OCTAL CHANNELS | STATUS BITS ACTIVE |
|---|---|---|---|
| SIOA,B,C (NOT REV 0) 1/200 | NONE | 0,1 | LOW |
| ACR 1,200 | A15 (AND TERMINAL OPTS.) | 6,7 | LOW |
| SIOA,B,C (REV 0) 40/2 | A14 | 0,1 | HIGH |
| 88-PIO 2/1 | A13 | 0,1 | HIGH |
| 4PIO 100/100 | A12 | 20,21 | HIGH |
| 2SIO 1/2 | ALL (AND A10 UP=1 STOP BIT DOWN=2 STOP BITS | 20,21 | HIGH |

23)     If the load device is an ACR, the terminal options (see second
        chart) can be set on the switches (along with a15) before the
        loadein is done. If A15 is set, the checksum loader will ignore
        all of the other switches and the monitor will ignore A15.

24)     If the load device and the terminal device are not the same, and
        the load device is not an ACR, then only the load options should
        be set before the loading. When the load completes ,the MONITOR
        will start-up and try to send a message to the load device.
        Press STOP, EXAMINE loaction 5121, set the terminal option
        switches, and then depress RUN.

TERMINAL OPTIONS

| TERMINAL DEVICE | SWITCHES UP | OCTAL CHANNEL DEFAULT |
|---|---|---|
| SIOA,B,C (NOT REV 0) | NONE | 0,1 |
| SIOA,B,C (REV 0) | A14 | 0,1 |
| ( A9 WILL BE IGNORED FOR THIS BOARD ) | | |
| 88-PIO | A13 | 0,1 |
| 4PIO | A12 | 20,21 (INPUT) |
| | | 22,23 (OUTPUT) |
| 2SIO | A11 | 20,21 (A10 UP=1STOP BIT |
| | | DOWN=2STOP BITS) |

Iif sense switch A9 is raised, interrupt I/O will not be enabled.   See Appendix F for discription of this feature.

The default channels listed above may be changed as desired  by  raising A8  and  storing  the  lowest  channel  number  (INPUT  FLAG CHANNEL) in location 7777 (OCTAL).

Note:   The "INPUT FLAG CHANNEL" may also be refered to as the  "CONTROL CHANNEL" in other Altair documentation.

The above information is useful  only  when  the  load  device  and  the terminal  device are not the same.  During the load procedure A8 will be ignored;  therefore,the board from which the monitor is loaded  must  be strapped for the channels listed in the load option chart.

25)     When loading paper tape from a device connected to a  SIO  A,B,C
        or  a 88-PIO board, start the tape reading and then depress run.
        If the device is connected to a 2SIO or  4PIO  depress  RUN  and
        then  start  the tape reader.  If you are loading from cassette,
        turn the cassette recorder to play.  Wait 15  seconds  and  then
        depress RUN.

26)     The new Checksum Loader will display 7647 on the address  lights
        when  running  properly.  When an error occurs (checksum "C"-bad
        data, memory "m"-data won't store properly, overlay  "O"-attempt
        to load over top of the checksum loader) the address lights will
        then display 7637. The ASCII  error  code  is  stored  in  the
        accumulator (A) and is being output on channels 1, 21, and 23.

27)     When the tape finishes reading, the MONITOR should start up  and
        print  the  normal  prompt  -  ?   .   If  you  are loading from
        cassette, STOP the player  imediately  so  other  files  can  be
        loaded.

Appendix F.

## Audio Cassette Users

The following table shows the order and length of files on the cassette of Package II.

| Program<br>Name | Time from Start of Tape<br>(in seconds) |
|---|---|
| MONITOR | 13 - 125 |
| ASM | 120 - 230 |
| EDT | 240 - 310 |
| AM2 | 320 - 415 |
| DBG | 430 - 510 |

When you are about to record a new file on a cassette, position the cassette after the last file. When using either the editor or assembler to dump out a file, start the recorder a few seconds before flipping sense switch 15. A gap of this type should be inserted between all files on a casette.

## ASCII Line Input

The following describes the action taken for various special characters.

<CR> - Ends a line. The monitor returns to the calling program when typed. It is not counted in the line length returned. A line feed is also written out if input is being echoed.
<LF> - ends a line. Only a line feed is echoed. See above.
<ESCAPE> - Ends a line. $ is echoed. See above.
Octal 0 - Ignored
<Control A> - rubs out complete line typed.
<RUBOUT> - Backspaces one character for each one typed.
<Control> z - End of file, branches to address given in control block.

## Interrupt I/O

Package II now supports input interrupts from the terminial device. One I/O card in the Altair can be wired for input interrupts directly to the bus interrupt line(PINT), or to the lowest priority on the vectored interrupt card. If the terminal is set for interrupts, typing a <Control C> will stop execution of a program and return to the monitor. All registers are saved in the register save area as described in the monitor section of this manual.

# Assembler Versions

Two copies of the assembler are supplied in Package II. Version 1 loads starting at location 5100Q and the symbol table starts at 123660. Version 2 loads at location 11520Q and the symbol table starts at 17006Q.

Running version 1 gives you maximum memory for program space but does not allow the editor to be resident at the same time. Version 2 lets you load Debug or the editor between the monitor and the assembler allowing you to assemble directly from the editors buffer using the edit buffer read feature. If this setup is to be used the editor must be patched to move the buffer area. The buffer area is assumed to be immediately after the editor, and if not changed, would destroy the assembler. See Appendix B for recommended memory layout.

## Absolute File Names

Version 1 - ASM
Version 2 - AM2