Programming Guide

DTC MICRO FILE

PROGRAMMER'S GUIDE

	page No	٠.
I.	Memory Overview -:	
II.	Program Environment -1	
	Physical Boundaries - 2	
	Monitor/Program Relationship -	2.
	Parameter Passing	2
	System Entry Point Definitions	-3
٠	'Snapshot' Activity	-4
III.	System Program Interfaces	-7
	Terminal/Line I/O	-7
	Logical I/O	-10
	Physical Disk I/O	-14
	Real Time Clock	-19
	Digital Display Control	_ 20
	Miscellaneous Routines	-20
IV.	Disk Layout and File Structures	-31
V.	Micro File Device Addresses	-38

I. Memory Overview

ROM-

0000 - 07FF	Monitor
0800 - OFFF	Terminal and Line I/O
1000 - 17FF	Disk I/O, Logical I/O, Real-time clock routine
1800 - 1BFF	Monitor Extensions

RAM-

1C00 - 1FFF Reserved

2000 - 223F	Terminal and line I/O Control and Buffers
2240 - 224F	File Control Block Pointers
2250 - 25FF	Monitor and Disk I/O Control and Buffers
2600 - 26FF	EXEC Program Area
2700 - 27FF	Transient System Command Program Area
2800 - TOP	Applications Program Area and Stack

II. Applications Program Environment

Physical Boundaries-

The applications program is typically loaded and executed at 2800H and may extend up to 3FFFH, in the minimum system, excluding a reasonable space for the program "stack". This is a 6K region. The program communicates with the "outside world" by calling various system routines which permit reading and writing to the disk, the terminal, and to the line.

It is important to note that certain "control" information is contained in RAM between the addresses 2000H and 27FFH. If the applications program inadvertently stores data into certain of these control elements, it can be disastrous to the Micro File system.

Monitor/Program Relationship-

Each program is "called" by the monitor (or EXEC process) as if it were a subroutine. This means that the program relinquishes control at program completion time simply with the RET instruction. The program must of course maintain the proper stack alignment for this technique to work. An alternate method of restoring control to the Monitor is to jump directly to the MON entry point. This method should only be used in error situations since it effectively terminates an active EXEC process.

Parameter Passing-

When a program acquires control it may access the statement which invoked it and glean its parameters from the statement. The program must handle parsing of the parameters, itself. At program entry the following conditions exist:

Register pair BC - Points to the first character of the "program name" term of the statement.

Register D = Number of characters in the statement, excluding the CR/LF sequence.

Example-

The following statement would give the program (XPROG) control with BC containing a pointer to the character X and a count of 17_{10} in the D register.

RUN XPROG AAA BBB CCC

Certain system subroutines, described in Section III, are available to ease the parsing problem. The following routines are useful in dealing with parameters in command statements.

STBNB - Scan to blank, then to non-blank.

HEX - Get a hex field.

GNAME - Get a file name.

DNUMB - Get a disk number.

System Entry Point Definitions-

Entry to the system routines have been provided at the beginning of each of the three major sections of ROM. Each entry point is a jump instruction to the proper routine. This allows a section to change internally without affecting the programs which use the routines. The following list of EQU statements defines the system entry points. You must include the applicable statements in your source program (Also, don't forget the ORG 2800H).

Registers are generally not preserved by called system routines. Exceptions are so noted in Section III.

Monitor Entry Points:

MON	EQU	40H	;	Monitor Primary Entry Point		
WMAPS	EQU	43H	•	Write Allocation Maps		
RMAPS	EQU	46H	;	Read Allocation Maps		
GMSS	EQU	49H	;	Get A Sector		
RMSS	EQU	4CH	•	Release A Sector		
CLEAN	EQU	4FH	1	Clean-Up Allocation Maps		
SERR	EQU	52H	;	System Error		
DERR	EQU	55H	;	Disk Error		
CDERR	EQU	58Н	;	Command Error		
STBNB	EQU	5вн	;	Scan To Blank, Then To Non-Blank		
HEX	EQU	5EH	;	Get A Hex Field		
GNAME	EQU	61H	;	Get A File Name		

```
DNUMB
          EQU
                64H
                          Get A Disk Number
   MOVE
          EOU
                67H
                          Move Subroutine
                      ;
                          Scan Directory For A File Entry
   GENT
          EQU
                6AH
                      ;
   GHOLE
          EQU
                6DH
                          Scan Directory For A Hole
   EROS
          EOU
                70H
                          Erase Function
   COMP
          EQU
                73H
                          Compare Subroutine
   EXENT
          EQU
                76H
                          EXEC "Call"
                      ;
   OVERL
          EQU
                79H
                          Overlay Function
   EXEN3
          EQU
                7CH
                          Special Entry For Snapshot
                      ;
   GLERR
          EQU
                7FH
                      ;
                           'LOAD'-ABORT
Terminal/Line I/O Entry Points:
   TGET
                800H
          EQU
                          Get A Char From Terminal
   LGET
          EQU
                803H
                          Get A Char From Line
   TGETE
          EQU
                806H
                          Get A Char From Terminal W/Echo
   LGETE
          EQU
                809H
                          Get A Char From Line W/Echo
   TPUT
          EQU
                80CH~
                          Put A Char To Terminal
                      ;
   LPUT
          EQU
                80FH
                          Put A Char To Line
   TBRK
          EOU
                812H
                          Send Break To Terminal
   LBRK
          EQU
                815H
                          Send Break To Line
   TIN
          EQU
                818H
                          Get A Text Line From Terminal
  LIN
          EQU
                81BH
                          Get A Text Line From Line
                      ;
   TOUTX
          EQU
                81EH
                          Put A Text Line To Terminal
  LOUTX
          EOU
                821H
                          Put A Text Line To Line
   TOUT
          EQU
                824H
                          Put A Text Line to Terminal With CR/LF
  LOUT
          EOU
                827H
                          Put A Text Line To Line With CR/LF
   ?TIO
          EQU
                82AH
                          USART Interrput
   INIT
          EQU
                82DH
                          Terminal/Line Initialization
  ABORT
          EQU
                830H
                          Test For Operator Break
   TIMON
          EQU
                833H
                          USART Checking Routine
Terminal/Line I/O Control Elements
   TCAUX
                200FH ; Terminal Control Block
          EQU
  LCAUX
          EQU.
                2017H;
                          Line Control Block
   TRBUF
          EQU
                201FH ;
                          Terminal Receive Buffer
  LRBUF
                          Line Receive Buffer
          EOU
                2125H :
                20A2H; Terminal Transmit Buffer
   TXBUF
          EQU
                          Line Transmit Buffer
  LXBUF
          EQU
                21A8H ;
```

Disk I/O Entry Points:

UNLOD EQU 1000H; Unload A Head

READ EQU 1003H; Read A Sector

WRITE EQU 1006H; Write A Sector

FORM EQU 1009H; Format A Sector

RTZ EQU 100CH; Reset To Track 00

ZSECT EQU 100FH; Zero A Sector

HEDS EQU 1012H; Unload All Heads

Real-time Clock Routines:

?RTC EQU 1015H; Real-Time Clock Interrupt

Logical I/O Entry Points:

OPEN EQU 1018H; Open A File

CLOSE EQU 101BH; Close A File

CREAT EQU 101EH; Create A File

ERASE EOU 1021H; Erase A File

RENAM EQU 1024H; Rename A File

TOP EQU 1027H; Go To Top Of The File

ALLOC EQU 102AH; Acquire A Sector

DEALL EQU 102DH; Release A Sector

DREAD EQU 1030H; Read A Sector Of The File

DWRIT EQU 1033H; Write A Sector To The File

System Control Elements:

UTIMR EQU 2007H; User Timer Value

XTIM EQU 200AH; User Timer Action Address

MCDSK EQU 2250H; Active Disk Number

SFILE EQU 2252H; Active File Name And Type (6 Chars)

Digital Display Control:

DSPZ EQU 1800H; Zero Display

DSPHL EQU 1803H; Display HL Contents

DSPP EQU 1806H; Increment Display

DSPM EQU 1809H; Decrement Display

Snapshot Sample:

```
*AS SHOT DØ Ø9ØØ
                                        (Program to illustrate
  2800
                   ORG 2800H
                                        snapshot)
  2800 3EAA
                   MVI
                         A, ØAAH
  2802 FF
                   DB
                         ØFFH
  2803 Ø1CCBB
                   LXI
                         B, ØBBCCH
  2806 FF
                   DB
                         ØFFH
  2807 11EEDD
                   LXI
                         D, ØDDEEH
  280A FF
                   DB
                         ØFFH
  280B 21FFFF
                   LXI
                         H,ØFFFFH
  28ØE FF
                   DB
                         ØFFH
  28ØF C9
                   RET
*SAVE SHOT 3100 310F 2800
                                       (Save the program)
*RUN SHOT
                                       Execute without SNAP T
2803 AA 46 2506 0406 2800 3FFD
                                       (A cc BC DE HL SP)
2807 AA 46 BBCC 0406 2800 3FFD
280B AA 46 BBCC DDEE 2800 3FFD
280F AA 46 BBCC DDEE FFFF 3FFD
*ED SNAP
                                       (Create SNAP T)
NEW FILE:
-I MD 2800 2810
-I MD 3100 3112
-E
                                       (Execute with SNAP T)
*RUN SHOT
2803 AA 46 2506 0406 2800 3FFD
2800 3E AA FF 01 CC BB FF 11 EE DD FF 21 FF FF FF C9
2810 BB 29 CD AE 38 7E B7 F8 23 7E FE 20 CA 91 36 21
3100 7E 71 BF C9 23 7E E6 7F FE 0A 3E 0D C2 F2 30 D1
3110 E1 B7 C9 21 27 31 CD 19 31 5E 23 56 23 7E 23 12
2807 AA 46 BBCC 0406 2800 3FFD
2800 3E AA FF 01 CC BB FF 11 EE DD FF 21 FF FF FF C9
2810 BB 29 CD AE 38 7E B7 F8 23 7E FE 20 CA 91 36 21
3100 7E 71 BF C9 23 7E E6 7F FE 0A 3E 0D C2 F2 30 D1
3110 E1 B7 C9 21 27 31 CD 19 31 5E 23 56 23 7E 23 12
280B AA 46 BBCC DDEE 2800 3FFD
2800 3E AA FF 01 CC BB FF 11 EE DD FF 21 FF FF C9
2810 BB 29 CD AE 38 7E B7 F8 23 7E FE 20 CA 91 36 21
3100 7E 71 BF C9 23 7E E6 7F FE 0A 3E 0D C2 F2 30 D1
3110 El B7 C9 21 27 31 CD 19 31 5E 23 56 23 7E 23 12
280F AA 46 BBCC DDEE FFFF 3FFD
2800 3E AA FF 01 CC BB FF 11 EE DD FF 21 FF FF FF C9
2810 BB 29 CD AE 38 7E B7 F8 23 7E FE 20 CA 91 36 21
3100 7E 71 BF C9 23 7E E6 7F FE 0A 3E 0D C2 F2 30 D1
3110 E1 B7 C9 21 27 31 CD 19 31 5E 23 56 23 7E 23 12
```

Miscellaneous:

SNAPS EQU 180CH; Initiate Snapshot Activity

EROS EQU 180FH; Erase Function

MFI EQU 1812H; 'FILES' Command Process

The 'Snapshot' Activity-

This feature is used to debug application programs. It is automatically activated when an RST 7 (0FFH) instruction is encountered in a program. When activated, it prints the snapshot location and the contents of all registers and the condition flags at that point. In addition, it activates and EXEC*-like program (\$SNAP) which executes the statements in the text file SNAP T.

Finally, control is returned to the program with the machine state restored.

Generally, SNAP T contains one or more MD statements to dump relevant blocks of memory. If a file SNAP T does not exist, just the registers and condition flags are printed before returning control to the program.

Usage-

The programmer uses the EDIT command to create a text file on disk 0 whose name is SNAP T. He then inserts RST 7 instructions into his program at strategic points, either by patching or at assembly time. Next, he loads and executes his program. The program will execute normally except for snapshot data being printed at the terminal.

^{*}Refer to the Operator's Manual

III. System Program Interfaces

Terminal/Line I/O

The program communicates with the terminal or line in terms of messages or individual characters. The message level is principally used for typing instructions to the operator and for reading his responses. The character level is only used with the program as a special need that is not provided by the message level, such as the monitoring of an incoming character stream for special control characters. The various entry points are described in pairs since a basic function applies either to the local terminal or to the line.

Message Level Routines-

There are six routines at this level, providing the indicated functions:

 $\rho \Phi^{\mu} T / TOUT$ - Output messages to the terminal, followed by CR/LF.

LOUT - Output messages to the line, followed by CR/LF.

f¢oty TOUTX - Output message to the terminal, without CR/LF.

LOUTX - Output message to the line, without CR/LF.

fin - Retrieve message from the terminal.

LIN - Retrieve message from the line.

The message output routines are used in the following manner:

Calling Sequence:

LXI H,MSG ; Point To Message

CALL TOUTY
TOUTX
LOUT
TOUTY
TOUTY

JNZ BREAK ; Detect Break

Where MSG is the beginning of a "message" consisting of a single byte character count of the number of characters in the text of the message, followed by the message itself. Messages are usually created at assembly time with DB statements:

MSG: DB 17, 'This is a Message'

FOR ALL PAYY ROUTINGS THE FORT HO IS PASSED IN C-REC

Upon return from one of the output routines:

If condition = 0, the message text was transmitted successfully.

If condition \neq 0, a "break" occurred during transmission.

A message input routine retrieves a line of text, performs CWL editing on it, and deposits it in the caller's buffer.

Calling Sequence:

LXI H,Buff ; Buffer Origin

 $exttt{MVI} exttt{M,SIZE}$; Set Buffer Size

CALL TIN ; Retrieve the Message

JNZ EXCEP ; Break or Buffer Exhaust

Upon return from one of the input routines:

If condition $\neq 0, \ldots$

and A = 1 - buffer was exhausted.

and A =-1 - a "break" occurred.

BUFF + 0 = Buffer size as established prior to the CALL.

BUFF + 1 = Number of characters of text input, including the trailing CR and LF.

BUFF + 2 = Text of message plus CR and LF.

TO BUFF + N

BCHEK - CHECK TEXT TERM REPORTS (0= HO, 3 TEST)

TCHP - TEST TERM CHARACTER PRESENT IN INFORMATION (0: HO 5: YES)

TBUFE TEST TEXA OUT PUT BUFFER EMETY (10 YES, 3 = HO)

PINIT THITTACIZE A FORT

COSE CHAR SUPPRISEDED.

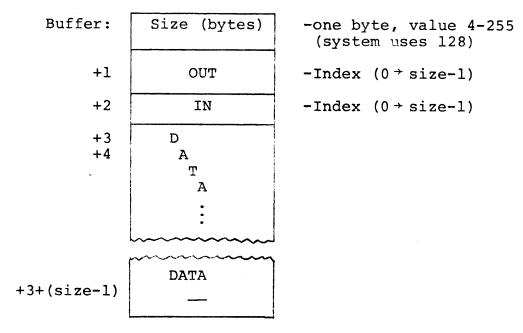
TECHE TO THE LONG ASSIST TO SEE THE DEFENDENT FAIR NO TO CHES UST NEFEED)

System Interface-

TCAUX and LCAUX receive the BREAK signal from the terminal or line. A BREAK received, sets the associated location non-zero.

Data is bidirectionally buffered via -RBUF (receive) and -XBUF (send). It is not recommended that buffer pointers be manipulated except via calls to the system I/O routines (TIN, TGET, etc). The transmit buffers, in particular, contain extra information for the sending of BREAK and for delays following line-feed and tab.

It is possible, by looking at buffer pointers, to determine that data is in the buffers. All buffers are structured as follows:



A "circular buffering" technique is used to maintain the buffers. "IN" indexes to the next free (empty) byte in the buffer. As data is put into the buffer, IN is advanced. The next byte after the last buffer location is the first buffer location (wrap-around). Likewise, "OUT" indexes to the next data byte to be removed from the buffer.

Whenever "IN" \neq "OUT", data is in the buffer. A situation where "IN" = "OUT" indicates an empty buffer (no data).

Character Level Routines-

PEET

TGET - Get a single character from the terminal.

LGET - Get a single character from the line.

These routines retrieve a single character from the proper input buffer and present it to the caller in the A register. If a break occurs, the A register and condition flag are set to zero.

PGET E

TGETE - Get a single character from the terminal w/echo.

LGETE - Get a single character from the line w/echo.

Get a single character from Terminal/Line receive buffer. Echo it if Full Duplex or Echoplex, except if it is a character, word, or line delete character (but echo such a character if it is in transparent mode from the terminal).

Upon return:

A = Character. If a break is seen, return is made with

A = 0. In such a case echoing may not have been done.

PPUT

TPUT - Put a single character to the terminal.

LPUT - Put a single character to the line.

Put a single character to the Terminal/Line transmit buffer. Checks for (NUL), (TAB, LF), and other things which require special action.

Input condition:

A contains character to be sent.

Return conditions:

0 = Character was put

Non 0 = Break stopped put

PBRK

TBRK - Send a break to the terminal.

LBRK - Send a break to the line.

Logical I/O

The routines described in this section provide access to the files in the system and relieve the programmer of the need to know about the physical characteristics of recorded files, such as linkage conventions and directory format etc....

Most of the routines require that a File Control Block (FCB) be provided by the program which describes the file to be accessed. A pointer to the FCB is provided when the routine is called.

Summary of routines:

OPEN - Opens a file

CLOSE - Closes a file

CREAT - Creates a file

ERASE - Erases a file

RENAM - Renames a file

TOP - Positions to top of the file

DREAD - Reads next sector of the file

DWRIT - Writes next sector of the file

ALLOC - Allocates a sector

DEALL - Deallocates a sector

The success or failure of a request is indicated to the calling program by the status of the zero condition flag and the contents of the A register.

If condition = 0, the requested operation was successful.

If condition ≠ 0, the requested operation was not successful and the specific reason for its failure is encoded in the A register.

^

FILE CONTROL BLOCK

		_	
0	Control Flags	TASK 1	ч ф.
1	-Spare Disk #		
2	User # Disk #		
3-7~1)	File Name		
8 II-13	File Type		
. 9 - 10 · 14 · 17	First Sector Of File SECTOR - TEACK		
11 TE18-21	Current Sector Of File		
13-147:-25	Next Sector Of File SECTOR - TRACK		
15 - (6-2-27)	Buffer Address		
		-	
17 ²⁸	User-Defined Extents	_	
1			

OPEN - Open a File

This routine opens a file. The caller provides an FCB with the filename and filetype filled in. The disk number may also be provided if known. If not known, it should be set to -1 and the routine will search all disks starting at disk g.

Calling Sequence:

LXI H, FCB

CALL OPEN

The Return Codes are:

0 = Open Successfully Done

1 = Unrecoverable I/O Error

2 = Invalid FCB parameter

3 = File not found

4 = File already open

5 = Maximum 'Opens' already exist

CLOSE - Close a File

This routine closes a file. The file must have previously been opened.

Calling Sequence:

LXI H, FCB

CALL CLOSE

The Return Codes are:

0 = File Successfully Closed

1 = Unrecoverable I/O Error

2 = Invalid FCB parameter

3 = File Not Open

CREAT - Create a File

Create a file and allocate its 1st sector. This routine creates a file as requested by the caller. The caller supplies the disk No., Filename and Filetype.

Calling Sequence:

LXI H, FCB

CALL CREAT

The Return Codes are:

0 = Create Successful

1 = Unrecoverable I/O Error

2 = Invalid FCB Parameter

3 = File Already Exists

4 = Disk is Full

ERASE - Erase a File

This routine erases a disk file, recovers the space and re-writes the directory record with a hole. The caller supplies the Filename, Filetype and Disk No.

Calling Sequence:

LXI H, FCB

CALL ERASE

The Return Codes are:

0 = File Erased

l = Unrecoverable I/O Error

2 = Invalid FCB Parameter

3 = File Not Found

4 = File Is Currently Active (is open)

RENAM - Rename a file

This routine alters the name of an existing file.

Calling Sequence:

LXI H, PARMB

CALL RENAM

The parameter block (PARMB) has the following format:

PARMB (+) 0-1 - Not Used

2 - Disk Number

3-7 - Old File Name

8 - Old File Type

9-13 - New File Name

14 - New File Type

The Return Codes are:

0 = Rename Successful

l = Unrecoverable I/O Error

2 = Invalid PARMB Parameter

3 = 'Old' File Does Not Exist

4 = 'New' File Already Exists

5 = 'Old' File is Currently Active

TOP - Go to top of file

This routine positions the file pointer to the top of the file. The next 'Read' command will result in the 1st record of the file being read.

Calling Sequence:

LXI H, FCB

CALL TOP

The Return Codes are:

0 = 'Top' Operation Successfully Done

1 = Unrecoverable I/O Error

2 = Invalid FCB Parameter

3 = File Not Open

DREAD - Read the next sector

This routine reads the next sector from the file into the users buffer. The FCB disk addresses are used and updated.

Calling Sequence:

LXI H, FCB

CALL DREAD

The Return Codes are:

0 = Read Successfully Done

1 = Unrecoverable I/O Error

2 = Invalid FCB Parameter

3 = End-Of-File

4 = File Not Open

DWRIT - Write the next sector

This routine writes the next sector of a file. It assumes an 'Append' type of write. It updates the FCB pointers.

Calling Sequence:

LXI H, FCB

CALL DWRIT

The Return Codes are:

0 = Write Successful

1 = Unrecoverable I/O Error

2 = Invalid FCB/Buffer-Link Parameters

3 = Disk is Full

4 = File Not Open

ALLOC - Allocate a sector

This routine allocates a disk sector to the caller. The Disk No. for which the caller is requesting is passed to the A-Reg. The sector address (if request is successful) is passed back to the caller in H & L.

Calling Sequence:

MVI A, DISKNO

CALL ALLOC

The Return Condition Codes are:

Zero = Allocation Successful

Non-Zero = Disk is full or I/O Error

DEALL - Deallocate a sector

This routine returns a disk sector to the available pool. Upon entry, the A-Reg contains the Disk No. and H & L contain the sector address.

Calling Sequence:

LHLD DADDR (Get Disk Sector Address)

MVI A, DISK NO (Get Disk Number)

CALL DEALL

Return Condition Codes are as follows:

Zero = Deallocation Successful

Non-Zero = Invalid Contents of A, H and/or L - or - unrecoverable I/O Error.

Physical Disk I/O

All data transfers to and from the disk(s) are handled by the Disk I/O Driver. There are six entry points to the Disk I/O Driver, as described in the following paragraphs. The caller must set the parameters which describe his requested operation into the proper registers and then CALL the desired routine.

Entry Point Summary:

RTZ - Reset to track 00

UNLOD - Unload a head

HEDS - Unload all heads

WRITE - Write a sector

ZSECT - Zero a sector

FORM - Format a sector

READ - Read a Sector

All registers except the A register are preserved by these routines. The A register is used to pass the return code to the caller.

Parameters are passed to these routines in the registers as follows:

A = Disk Number (0 or 1)

D&E = Track and Sector numbers, respectively

H&L = Data Buffer Address

Return Codes-

Upon return to the caller, immediately after the CALL instruction, the contents of the accummulator and the condition code reflect the success or failure of the request. The caller must therefore supply a JNZ instruction to his own error routine to take appropriate action for the indicated return code. If the zero condition is set it means the command was completed successfully.

Return Code Meanings

- 0 = The command was successfully executed.
- 1 = The drive-not-present status bit is high.
- 2 = The file-inoperable status bit is high.
- 3 = The file-busy status bit is high.
- 4 = The disk-ready status bit is low.
- 5 = The disk is write-protected. (Only after \underline{WRITE} FORM or ZSECT.
- 6 = The sector-error status bit is high. This generally means the disk has not been formatted.
- 7 = The CRC-error status bit is high.
- 8 = Position Error.
 This generally means the recorded track or sector
 number is inconsistent with what should be recorded
 on the sector.
- 9 = Command Error

A parameter provided by the caller is incorrect in one of the follow ways:

- A) The disk number given is beyond the Disk I/O Driver's configuration.
- B) The track number is too large.
- C) The sector number is too large.
- D) The Read operation is not valid. The caller's data buffer address is too low. It points to ROM or a control block area.
- 10 = Data Error after WRITE.

Error Recovery-

Error recovery is an integral part of the disk I/O driver and occurs transparently to the caller before control is returned to him.

The disk I/O driver provides error recovery for "soft" errors as follows:

Read-

A re-read is attempted 10 times or until successful. If not successful, after 10 attempts, the head positioner is stepped in the <u>same direction</u> 1 track position and then in the <u>opposite direction</u> 1 track position. The read is then attempted as many as 10 additional times. If still not successful, it is a hard error.

The "re-positioning" of the head is by-passed if positioned on the outermost or innermost track of the disk.

Write-

The command is attempted 10 times or until successful. If not successful after 10 attempts it is a hard error.

Routine Descriptions-

RTZ - Reset to track 00

This routine moves the positioner arm of the drive to track 0.

Required parameter:

A = Disk Number

UNLOD - Unload a head

This routine moves the head away from the recording surface. This routine is used when prolonged inactivity for the disk is anticipated.

Required parameter:

A = Disk Number

HEDS - Unload all heads

This routine makes multiple calls to UNLOD to move all heads away from the recording surfaces.

Required parameters:

None

FORM - Format a sector

This function is functionally similar to WRITE except that it does not verify track position before the write and does not perform a read-back after the write. This routine is used exclusively by a Disk Formatting Program.

Required Parameters: Same as WRITE

ZSECT - Zero a sector

This routine is functionally similar to WRITE except that it writes all zeros to the specified sector and does not perform a read-back. It is used by the sector deallocation mechanism to ensure that sectors that are available for subsequent allocation are zeroed. This routine should not be used by an applications program.

Required Parameters:

A = Disk Number

D&E = Track and Sector Numbers, respectively

Real Time Clock

The Real-Time Clock is available to the user for timing purposes. To use it, first store an action address at XTIM and a timer value into UTIMR. When the time interval has expired, control will be transferred to the address in XTIM via a CALL. The interrupt system is active, but the interrupt priority is such that only the panel reset button is still active. Therefore, the timer action routine should be very short. Return from the timer action routine to re-enable the full interrupt system.

The time interval is measured in units of 16ms. UTIMR is loaded with the number of intervals (+0, -1) to wait. UTIMR is counted down by one each clock period until it reaches zero. Because the clock is usually running, the store to UTIMR should be made with interrupts disabled, or with an SHLD.

Digital Display Control-

There are four routines for manipulating the digital display on the front panel:

DSPZ - Zero the display

DSPHL - Display HL contents

DSPP - Increment Display (Decimal)

DSPM - Decrement Display (Decimal)

Registers D & E are not used by these routines.

When a program acquires control, the display has been set to zeros.

Miscellaneous Routines

MON - Monitor primary entry point

This routine performs the following tasks:

- 1) Sets the stack pointer to the top of RAM.
- 2) Establishes the interrupt level and enables interrupts.
- 3) Pushes a vector to MON into the stack (pseudo CALL).
- 4) Types the operator prompt (*).
- 5) Waits for a command from the terminal.
- 6) Upon receipt of a command, zeros the display and dispatches to the proper process.

MON acquires control after the "reset" greeting message. Thereafter, it acquires control after each system command or program has completed its task. The single exception to this occurs when the program has been activated by a statement from a file (EXEC process), in which case control is returned to the EXEC process.

A program uses this entry point directly (with a JMP) only when an error condition has been encountered which requires operator intervention. However, a program restores control to this point indirectly, at the completion of its task, with an RET instruction. This is true so long as commands are being supplied individually by the operator (not EXEC).

SERR - System Error

This routine types 'SYSTEM ERROR AT XXXX' and then jumps to MON, terminating the current activity. XXXX is an address retrieved from the stack which simply shows the point in the program where the error was detected.

A program calls this entry point when a situation is encountered which should never occur, and if processing were allowed to continue, would produce indeterminate or disastrous results.

Normal usage is:

CMP B ; Compare

CNZ SERR; Exit if NE, system error

DERR - Unrecoverable disk error

This routine types 'DSKERR N AT XXXX' then jumps to MON, terminating the current activity.

A program calls this entry point when the return condition from a call to a physical disk I/O routine is intolerable.

Normal usage is:

CALL WRITE ; Write a sector

CNZ DERR ; Exit if not successful

CDERR - Command Error

This routine types 'CMDERR' and jumps to MON, terminating the current activity.

A program jumps to this entry point if it encounters an error in content, or form of the activating command statement.

Normal usage is:

CALL STBNB ; Scan To Next Non-Blank String

JZ CDERR; Command Error If Not There

CMBERR - INCE OFFRE EVERT CHAR CONSIDERS OFFICE

LARIT

ABORT - Test for Operator or Line Break

This routine is called to determine if the operator has depressed the break key or if a break has occurred on the line. If either has occurred, control is given to entry point MON of the monitor. Otherwise, control is returned to the caller.

Normal usage:

CALL ABORT ; Terminate if break

OVERL - Overlay Function

This routine is used to retrieve and execute a program overlay.

Input conditions:

A MCDSK contains the disk number

ECSFILE contains the 5-character file name followed by the filetype 'P'.

This routine is either CALLed or executed by a direct JMP. Since program overlays typically terminate with an RET instruction, you can either CALL this routine to regain control when the program overlay is finished, as in a "main" program, or you may use a JMP instruction, which will return control to the next outer level when the program overlay is finished.

STBNB - Scan To Blank, Then to Non-Blank

This routine scans the command statement to the beginning of the next character string, maintaining the pointer in BC and the residual character count in D. It sets the condition code to indicate whether a string was found or not.

A program calls this routine during his command statement parsing activity.

Input conditions:

- BC points to a character position of the command statement.
 - D contains the residual character count of the command statement.

Return conditions:

If condition $\neq 0$,

BC points to the first character of the string.

D contains the residual character count.

If condition = 0,

The statement was exhausted without encountering a character string.

Registers E, H & L are not used.

Note:

The calling program must maintain BC and D if subsequent calls to STBNB are to be made.

HEX - Get a hex field equivalent

This routine translates a 1 to 4 character text string of hexadecimal digits in the command statement to its binary equivalent, maintaining BC and D. All registers are used:

Input conditions:

- BC points to the first character of the string.
 - D contains the residual character count of the command statement.

Return conditions:

HL contains the binary equivalent of the string.

BC and D are updated to the terminating blank character position.

Other:

A jump to CDERR (Command error) occurs if one of the following conditions is detected:

- 1) Hex string is longer than 4 digits.
- 2) A non-hexadecimal character is encountered other than a blank.

GRAT - Set 15T LINK ADVISED TO THE FORM & WEEK NUMBER'S CATHLOC ENTRY - HL = FCB (USES DESK USER & FCB ROPPER) FRIT = 3N2 AN CATHLOC FRISTS

IZ : B,C,DE COIDAIN DISK ADIMAN

GNAME - Get a file name s/ca

an FCB

This routine moves five characters of a character string from the command statement to a storage area designated by the caller, padding with blanks on the right if fewer than five characters are in the string. BC and D are maintained. Processes the complete file specific for the string is 34 more than the string is a string of the specific form.

All registers are used.

Input conditions:

- BC points to the first character of the string.
- D contains the residual character count.
- HL points to the caller's storage area. (FCB)

Return conditions:

- BC points to the string terminator character
- D contains the residual character count

HL has been incremented by 5:

2 = file spec OK

Normal usage:

LXI H, SFILE ; File name destination

CALL GNAME ; Retrieve name from text

JHZ error JC wild

DNUMB - Get the disk number

This routine establishes the active disk number for subsequent processing. It stores the default disk number (0) into MCDSK, and calls STBNB to scan the command statement to the disk number parameter. If the parameter is present, it translates the parameter to its internal code and stores it into MCDSK. BC and D are maintained.

Registers E, H & L are not used.

Input conditions:

- BC points to the preceding text string or its terminating blank.
 - D contains the residual character count.

Return conditions:

If condition = 0, a parameter string was not found, the default disk number is in effect.

If condition \neq 0, the disk number was explicitly declared with a parameter.

PNUMB- GFT FORT HE.

FASSES BACK PORT IN THE P. P. P. P. RETURN CODE SAME AT DHUMB

Other:

A jump to CDERR (Command error) occurs if one of the following conditions is detected:

- 1) The parameter does not begin with 'D'.
- 2) The second character is not a numeric, or is larger than the largest valid disk number.

Note:

This routine can only be used when the disk number parameter is required in the "next" parameter position or when it is optional as the <u>last</u> parameter of the command statement.

MOVE - Move routine

This routine simply moves a block of data from one location to another.

Registers D & E are preserved.

Input conditions:

BC points to the source data block

HL points to the destination

A contains the count of the number of bytes to be moved.

Normal usage:

MVI A,50 ; Get block size

LXI B, FROM; Get block location

LXI H,TO ; Get destination

CALL MOVE ; Move the block

CLOSE - Closes All files for task n. n is passed in the A-reg WMF - Call to wait for system resources } Destroy A. freezes RMF - Call to release system resources } B,C,DE, HI

COMP - Compare routine

This routine compares one data block with another.

Input conditions:

- BC points to the first data block
- HL points to second data block
 - E contains a count of the number of bytes to be compared.

Return conditions:

- If condition = 0, the first data block equals the second data block, and BC and HL point to the byte positions just beyond their respective data blocks.
- If condition ≠ 0, the data blocks are not equal, and BC and HL point to the bytes which yielded the unequal result.

Normal Usage:

MVI E,36 ; Size of compare

LXI B, BLK1; First block location

LXI H, BLK2 ; Second block location

CALL COMP ; Compare

JZ EQUAL ; Jump if a match

GENT - Scan directory for a file's entry

This routine scans the directory of the specified disk (in MCDSK) for the entry which represents the specified file (in SFILE). If a matching entry is found, the track/sector address of the first sector of the file is returned to the caller. All registers are used.

BC = A (FCR)

Input conditions:

A MCDSK contains the disk number

Bic SFILE contains the 5-character file name and 1-character file type.

HEL BUTTER ALDRESS TON RELECTED DISIE RECORDS TIVE

Return conditions:

If conditions = 0,

-DE contains the track/sector address of the first sector of the file

DE BC

DLINK contains the track/sector address of the directory sector which holds the entry.

HL CONTAINS ADDRESS OF DIRECTORY ENTRY

WBUPL holds the directory sector data.

If condition , a matching entry for the specified file name was not found in the directory.

If condition) o It is a disk error code

GHOLE - Get a hole in the directory for a new entry

This routine finds or creates a slot for a new directory entry on the specified disk.

All registers are used:

Input conditions:

A-reg MCDSK contains the disk number

HEL DIFFORMY PECORE SURFER

Return conditions:

If condition = 0,

HL points to the usable entry location

DJE

DLINK contains the track/sector address of the directory sector which holds the entry

CALLERY SUFFER holds the directory sector data

If condition disk space was exhausted when attempting to add a sector to the directory.

If condition 20, it is a disk error code

Normal usage:

A program calls this routine only after calls have been made to GENT to determine if the file already exists, and to RMAPS to prepare for the possibility of sector allocation to extend the directory.

EROS - Erase a file

This routine calls GENT to determine if the specified file exists and, if it does, changes the directory entry to an "available slot" for subsequent use and restores the sectors of the file to the sector allocation map.

All registers are used:

Input conditions:

A-Kic MCDSK contains the disk number

Øic SFILE contains the file name and type

HEL BUFFER APPLACES FOR READING DISH RECEADS INTO

Return conditions:

HEGATIVE

If condition *0, the specified file is not represented in the directory.

If condition = 0, the file has been erased.

If condition >0, it is a disk error code

RMAPS - Read the allocation maps

This routine reads the sector allocation maps from the specified disk into the sector allocation map buffers.

A program uses this routine in conjunction with GMSS, RMSS, and WMAPS. RMAPS is called as a first step before making calls to GMSS and/or RMSS, which alter the number of available sectors.

The program must set the proper disk number into MCDSK prior to calling RMAPS.

Registers B, C, D, E, H & L are preserved.

Upon return:

If condition \neq 0, the A register contains an error code as returned from READ.

Normal usage:

CALL RMAPS ; Read the maps

CNZ DERR ; Trap disk errors

GMSS - Get a sector

This routine examines the resident sector allocation map buffers and returns the next available sector to the caller, deleting that sector's bit from the resident map.

This routine presupposes that a call to RMAPS has been provided by the program prior to its being called and further, that a call to WMAPS will ultimately follow.

A = DISK M4.

Register B, C, H & L are preserved.

Upon return:

MECATTOR

If condition # 8, disk space has been exhausted.

If condition = 0, D & E = Track number and sector number, respectively, of allocated sector.

If condition >0, it is a disk error code

Normal usage:

CALL GMSS ; Get a sector

JNZ EXH ; Space-exhaust routine

RMSS - Release a sector

B,C,

This routine ensures that the track number and sector number passed in D & E are valid and then restores the sector's bit to the resident map. The released sector is then written with zeros.

The routine presupposes that a call to RMAPS has been provided by the program prior to its being called and that a call to WMAPS will ultimately follow.

A = DISK NO.

Registers B, C, D, E, H & L are preserved.

Upon return:

If condition ≠ 0, the A register contains one of the following error codes:

following error codes:

11 = Non-valid track or sector number,
 or the resident allocation map
 has been destroyed.

12 = The sector being released is already
 represented in the map.

Normal usage:

LHLD SECT ; Get the sector ID

XCHG ; To proper registers

CALL RMSS ; Release it

JNZ ERR3 ; Error Routine

WMAPS - Write the allocation maps

This routine examines the "write" flags of the sector allocation map buffers and appropriately updates the sector allocation maps on the disk.

A program uses this routine in conjunction with RMAPS, GMSS, and RMSS. WMAPS is called as a final step after one or more calls to GMSS or RMSS have altered the resident sector allocation map.

The program must set the proper disk number into MCDSK prior to calling WMAPS.

Registers B, C, D, E, H & L are preserved.

Upon return:

If condition \neq 0, the A register contains an error code as returned from WRITE.

Normal usage:

CALL WMAPS /; Write the maps

CNZ DERR / ; Trap disk errors

CLEAN - Clean-up allocation maps

This routine determines if allocation maps are currently resident which have not been written to their disk and, if so, writes them.

A program calls this routine as a housekeeping measure to ensure that an improper or incomplete sector allocation/ deallocation activity is properly "closed-out" on the disk.

Registers are not preserved.

/This routine stores a disk number into MCDSK if the maps are written.

IV. Disk Layout and File Structures

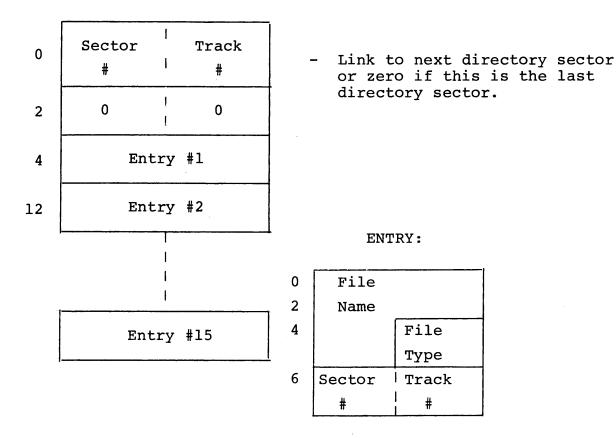
Each disk that enters the system is formatted in such a manner as to enable certain system routines to manage the space and files that will reside on the disk. In other words, each disk has a pre-defined "structure" comprised of the following items:

- 1) Label Sector
- Track 0, Sector 0 of every disk contains the label of the disk. The label is in the standard output message form and is written when the disk is formatted or when the LABEL command is used to re-label the disk.
- 2) Directory Sectors
- These sectors contain an entry for each file stored on the disk. Entries are added as files are created and "deleted" when they are erased. The entry effectively points to the beginning sector of the file it represents. Directory sectors are assigned near the mid-point of the positioner-arm travel to minimize "seek" time.
- - Allocation Map Sectors These sectors contain bit patterns which represent the unused sectors on the disk. Each sector of the disk is represented by a single bit in the allocation map in a particular position. When a sector is allocated, its corresponding bit in the allocation map is set to zero and the updated map is written to the disk. Conversely, when a sector is released, its bit in the map is set to one, etc. Allocation map sectors are assigned near the mid-point of the positioner-arm travel to minimize "seek" time.
- 4) File Storage Sectors
- The remainder of the disk consists of unassigned sectors or sectors containing files. Unassigned sectors are written with all zeros when the disk is initially formatted and are also written with all zeros when they are released. Therefore, any sector containing non-zero data is part of a file or is part of one of the above items.

Directory Sectors Structure -

When the disk is formatted, it is written with a directory that occupies 4 sectors. The first 4 directory sectors are on track 39, sectors 1, 9, 17, and 25. The 4 logically-linked sectors are separated from each other so that the entire directory can be scanned in one disk revolution. The directory is empty at this time but contains enough space to accommodate 60 entires. (The directory automatically expands to additional sectors if more than 60 files are stored).

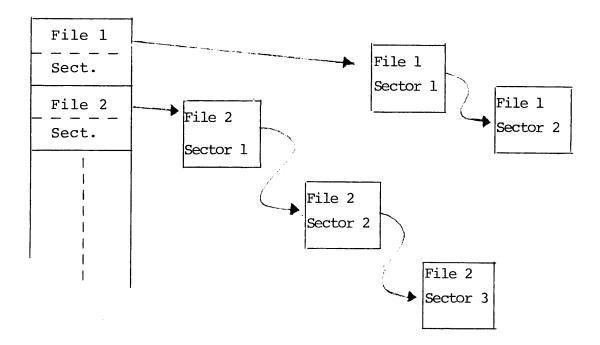
Each directory sector has the following format:



Directory sectors are initially written with all fields set to zero except the link field.

When a file is added, an entry is created and inserted into the directory in the first available entry "slot" scanning from the first directory sector to the last. When a file is erased, its entry is found and the first byte of the entry is set to zero, thereby becoming a slot in the directory for subsequent re-use by a new entry.

A schematic that shows the relationship between the directory and the files is given below:



Allocation - Map Sectors Structure

When the disk is formatted, it is written with an allocation map that occupies 4 sectors. The allocation map sectors are on track 39, sectors 0, 8, 16 and 24. Each of the sectors represents a portion (approx 1/4) of the total available sectors on the disk. Each potentially available sector on the disk is represented by a particular bit position in one of the allocation map sectors. If the bit is =1, the sector is available for allocation, if it is =0, the sector is already in use. Each allocation map sector has the following format:

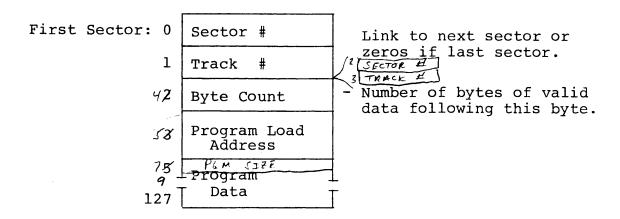
0	FIRST TRACK #	DIR. OF ALLOCATION	 1 or -1
2	LAST TRACK #	0	
4	SECTOR (COUNT	 Number of sectors available in the bit map.
6	AVAILABLE- SECTOR BIT MAP	•	Each 4 bytes represents 1 track of storage

Sectors are allocated within a track in the following order:

- 0, 4, 8, 12, 16, 20, 24, 28,
- 1, 5, 9, 13, 17, 21, 25, 29,
- 2, 6, 10, 14, 18, 22, 26, 30,
- 3, 7, 11, 15, 19, 23, 27, 31,

Program-File Sectors Structure

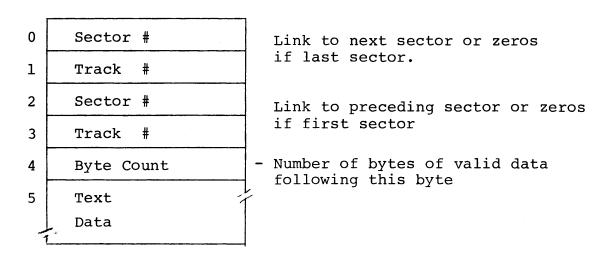
Each program file (P) consists of sectors in the following format:



Subsequent sectors of the file differ from the first one in that there is no Program Load Address field. Program data then starts in the byte #3 position.

Text File Sector Structure

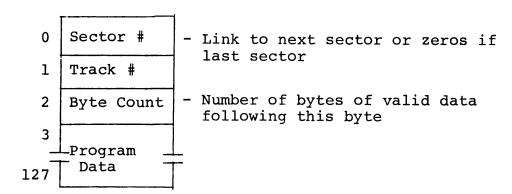
Each text file (T) consists of sectors in the following format:



The text data of the file consists of character strings bounded by CR/LF pairs. A sector may therefore contain part of a string or several strings.

BASIC-File Sector Struction

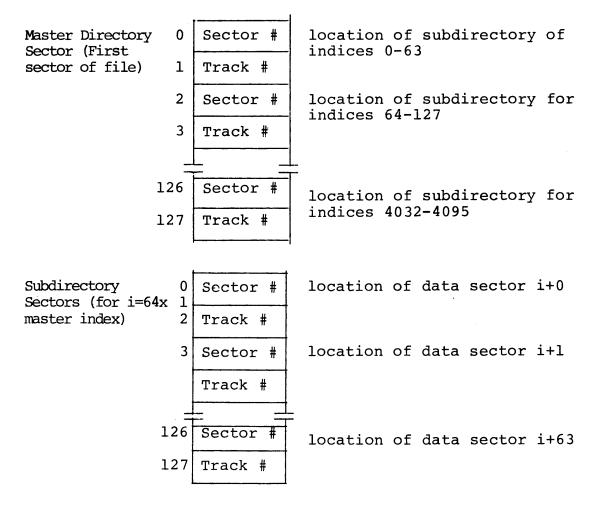
Each BASIC file (B) consists of sectors in the following format:

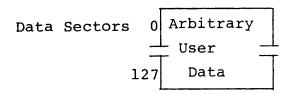


Program data is a compressed form of the BASIC source program, as saved by a "SAVE" command.

Random File Sector Structure

Each random file (R) consists of sectors in the following format:





If a data sector is absent, the subdirectory pointer is zero. If no data sectors exist for any entry in a particular subdirectory, the subdirectory is released and the master directory entry is zero.

The index range is 0 to 4095

V. MICRO FILE DEVICE ADDRESSES

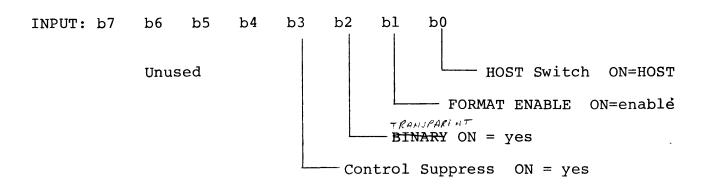
```
00
          Interrupts
   01
          Real Time Clock
02-03
          Reserved
04 - 07
          Spare
   08
          Read Disk Status - Write CMDS
                                                     INFT 1 (D0 + D1)
   09
          Read Chars - Write Chars
   0A
          Read Sect Under Head - Req Disk XFER
   0B
                                 - Write Strobes
   0C
                                 - Head Position
0D-0F
          Reserved
10-18
          Spare
18-1C
          INTF 2 (D2 + D3)
1D-1F
          Reserved
   20
          Data in
                               - Data Out
   21
          Status in
                               - Function Out
                                                     TERMINAL
                                                     USART 1
   22
          Speed/Parity in
                              - Clk Speed Out
   23
         Echo/Delay in
                               - Reset Out
   24
                                     **
   25
                                     11
                                                     LINE
   26
                                     **
                                                     USART 1
   27
28-2F
         USART Board 2
            "
30 - 37
                       3
            **
                  11
38-3F
                       4
   40
         Status in
                              - Indicators Out
                                                     Board 1
            **
   41
                               - Term/Line Parity &
                                 Reset-Format-Enable
                                 Function
42-43
            **
                                                     Board 2
44-45
            "
                                        11
                                                     Board 3
46-47
            **
                                                     Board 4
```

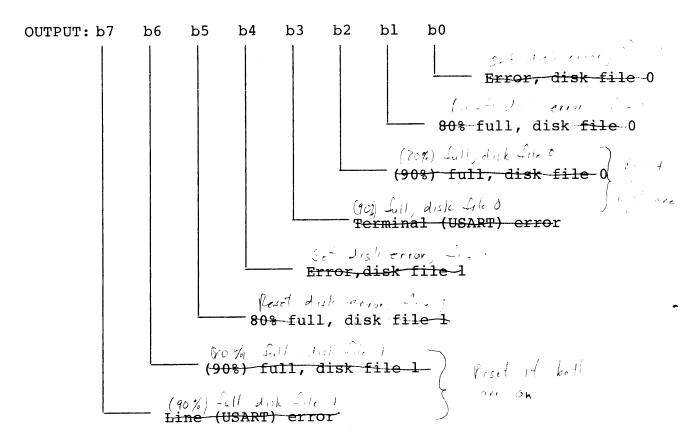
2F

Panel Switches and Lights

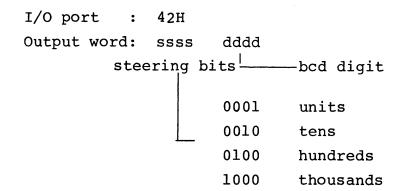
The panel switches are accessed with I/O Port: 40H

Input from 40H reads the switches
Output to 40H sets the lights as follows:





In addition, there is a 4 digit decimal display that can be set:



I/O PORT DEFINITIONS & USE OF INTERRUPTS FOR THE DTC MICRO FILE

Interrupt System

From the programmer's point of view, there are % interrupt levels, 0-7, arranged in a priority scheme with 0 the highest priority and $\mathcal{T}_{\mathcal{C}}$ the lowest. The priority interrupt chip is an output device, controlled as follows:

I/O Port # : 0

b7 b0

Output word: xxxx0sss

SSS is a binary level number. When sent to the interrupt chip, it enables all interrupt levels, sss and below.

E.G: to enable all levels use:

MVI A,7

OUT INTRP

Once an interrupt has been acknowledged, you must function some level in order to re-establish interrupts (The priority chip locks out further interrupts until refunctioned).

Currently Assigned Interrupt Levels:

<u>"R"#</u>	<u>Level</u>	RST Address	<u>Use</u>
R7	0	00Н	Power on
R6	1	08H	Front Panel Reset
R5	2	10H	Real Time Clock
R 4	3	18H	USARTS
	-		
(R2	86	28H	Disk Drive) - Provisional

Real Time Clock

The Real Time Clock can provide interrupts at 4, 8, 16 or 32 ms. intervals. For the Micro File System, 16 ms. is selected. The clock is controlled as follows:

I/O Port #: 1

Output word: Bit 0: 0 = Reset Interrupt Request 1 = Leave Interrupt Request

Bit 1: 0 = Disable RTC 1 = Enable RTC

Normal Use is:

MVI A,3

OUT CLOCK To start the clock

USARTS

Two USARTs are provided, "Terminal" and "Line". They have port numbers: $20_{\rm H}$ and $24_{\rm H}$, respectively. The sub-ports and functions are allocated as follows:

PORT#	INPUT	OUTPUT
Port+0 Port+1 Port+2	USART data (RCV) USART status Oppp Osss Speed parity	USART data (XMT) USART function 0000 ssss — Clock Speed
Port+3	ddddd <u>eee</u> -duplex Delay	00000001 clears all XMT interrupt requests Port 23 only OF SIFAR USANT HILL INCLUDING HILL OF SIFAR USANT
sss -	Speed Code	ppp - Parity 2 - CLEAR USART HY
000	- 110	000 - EVEN
001	- 150	001 - ODD
010	- 300	010 - MARK
011	- 1200	011 - SPACE
100	- 2400	100 - NONE (8-bit)
101	- 4800	
110 ** 5 2A 8 * ** 2A,	- 9600 20 1. In applies that while is the legal bit	

ddddd	-	Delay	eee	_	Duplex
			0.01		1.6
00001	_	NO DELAY	001	_	Half
00010	-	100 ms	010	-	Echoplex
00100	-	200 ms	100	-	Full
01000	-	400 ms			
10000	-	800 ms			