ANN

ADAM

E.O.S.

PROGRAMMER'S MANUAL

Adam News Network

EOS Programmer's Manual

Here it is! The EOS Programmer's Manual by Guy Cousineau. It's a very good document, and I hope that you find it useful.

ELEMENTARY OPERATING SYSTEM PROGRAMMER'S GUIDE

Contents

FOREWORD

The EOS (Elementary Operating System), is one of three operating systems available to the ADAM programmer. The other two, (CP/M (now replaced by the superior and user friendly T-DOS), and 057), are best suited for "work horse" or practical programs, and game graphics, respectively.

The EOS, the subject of this work, is a sort of bridge between the extremes of the "work horse" T-DOS, and the graphics game 0S7 operating systems. Thus the three operating systems complement one another.

Inasmuch as it is in the best interest of ADAM owners to have as many good programs for the ADAM as possible, (thus extending its usefulness, and thereby its practical lifetime), the ADAM News Network, (ANN), felt that it was desirable to publish a work that would help all interested parties write programs with the greatest facility that could be made available to them. ANN wants to encourage the ADAM owner to explore the field of Machine Language (ML) programming, and to encourage the development of ML programs. This work is designed to help ADAM owners use the EOS routines as correctly, and as effectively as possible.

Thus, through the generous work of Guy Cousineau, the author, this work is presented to help all who would, begin programming efforts to the benefit of us all.

The EOS, of course, must be addressed via ML instructions. There is, on the other hand, much to be said of the advantages of programming in the BASIC language. It is certainly easier to understand initially. But the direct use of ML programming instructions, though more time consuming perhaps, allows the writer to have program compactness, flexibility, and speed of execution, not possible by the use of any other programming language. Furthermore, some of the hardware specific functions on the ADAM can only be addressed by ML routines.

ANN also wishes to convey to all readers. and interested parties the understanding that the publication of this work is not the culmination of an EGO TRIP on the part of ANN or on the part of the author. If there are things found herein that are unclear, confuse, or clutter the mind, the reader is encouraged to write to, or otherwise get in touch with the ADAM community. Call the FIDO NET ADAM conference, any ADAM BBS, Compuserve, and direct whatever question you may have to the author or to any other ADAMite who might be in a position to answer such queries or implement your suggestions. No one will be offended. In fact, the contrary is the case:

ANN and its members encourage such dialogue.

Whether or not other works outlining the best uses of the 0S7 and the T-DOS operating system will. be published in the future will depend upon the response to this publication. ANN is dedicated to the promotion of anything that will help ADAM users. If a need is shown for such works, they will surely be forthcoming.

Having been requested to write this "INTRODUCTION", I would like to say that we are indeed fortunate to have Guy Cousineau as the author. I know Guy, and have worked with him on projects; I have always found him to be very personable and understanding. I personally have had to go back for repeated explanations of some items which he has written in the past, and found that he is not an AUTHORITY but a TEACHER. He has patience beyond reason, and tries very hard to help whenever he can. He is indeed an expert, but he is not an AUTHORITY who cannot bear to have his writings and. ideas questioned.

In his work, Guy does an excellent job of making the EOS routines understood. I have an original EOS manual, and have noticed that there are many invalid assumptions made there relative to the level of understanding of the reader.

Guy has clarified it greatly where things were nebulous.

We are fortunate indeed!

Mel Ostler

[ADDRESS REMOVED]

ACKNOWLEDGEMENTS

Before going any further, I feel obliged to. say thank you to all those who knowingly or not have made this work possible. This includes programmers from which I have picked up information or programming tricks, people who have encouraged me in my ML efforts, people who have published information on the ADAM operating system, and others who have generally been a support in my efforts on behalf of ADAM.

At the risk of forgetting some of the major contributors, I will list a few sources. If I leave any one out, it is definitely not intentional: consider yourself honourably mentioned in the above paragraph.

```
        Tony Morehen            for getting me. involved in ML
        Peter and Ben Hinkle    for their Hacker's Guides
        Barry Wilson            for starting up ANN
        Mel Ostler              for his books on ADAM and his introduction
        Bruce Walters           for his input on the sound routines
        James Walters           for his help with sound and interrupts
        Ian Cottrell            (non-ADAMite) who has been a ML mentor
        Ron Mitchell            for resurrecting the OTTAWA user's group
        ANN as a whole  for reviving my interest in ADAM
        ADAMITES                for encouragement by buying my software and
asking all those "not so silly" questions
```

OVERVIEW

What is the EOS? It is the Elementary Operating System of the ADAM, and not the Extended Operating System, as some people call it. It consists of a series of routines which help the programmer interface with the hardware

attached to the ADAM. The EOS takes up the top 8K of memory and uses an additional 3K of buffers just below it. This 8K section has routines to address the keyboard, printer, screen, and storage devices. Each routine has a specific calling convention and may return information about the status of the operation or of the operating system itself.

The EOS is used by applications programs such as SmartBasic, AdamCalc, SmartFiler, etc. Other programs such as FileManager and other AJM software products (this is the only plug you will see in this manual) make use of the EOS to handle its elementary functions.

This manual talks a bit about the benefits of machine language programming, the EOS structure, and memory configuration. The greater part of the manual, however is devoted to the explanation of EOS routines. Not all routines will be covered. Some of the incomplete or non-implemented routines will be skipped so not to confuse the reader. It is intended as a reference tool for the machine language programmer, whether beginner or advanced. The first section deals with executive calls. As most application programs do not require these, it is recommended to skip over this section and come back to it later. Otherwise, you might try to absorb a lot of technical information which is not required. It has been placed first since it is logical in the sequence of operation.

Considering that Assemblers usually generate HEXADECIMAL codes, all addresses quoted in this manual will be in hexadecimal format only. This manual does not propose to explain Z-80 programming or the use of the Op-Codes. Someone with a knowledge of Z-80 programming will be able to use the manual to effectively access the operating system. In addition, some of the examples and complete programs/routines supplied in this manual may help the programmer pick up tricks about Z-80 programming or the EOS itself.

Please address any technical questions to:

Guy Cousineau

[ADDRESS REMOVED]

MACHINE LANGUAGE PROGRAMMING

Why use machine language programming? My three favourite reasons are SPEED, SIZE, and CONTROL.

By skipping superfluous steps and compressing operations into the most effective structure, you can achieve processing speed which has no match in any language, whether it be BASIC, Fortran, C, Pascal, etc. Think of the fastest BASIC routine you have seen to sort a series of strings. Then look at the speed of ADAMCALC's sort feature. The difference is amazing.

The programmer can effectively manipulate routines and common subroutines into a package that occupies less space than any other language. While some powerful BASIC programs may be quite short, consider that they have 27K of overhead: the BASIC itself! A Self-contained Machine language program can be written using only the EOS to handle console and file I/O. I have seen some self-contained machine language programs which take less than 1K and perform several functions.

Control is another key aspect of machine language programming. There is no need to assume that a particular SYSTEM routine (like BASIC's FOR, INT, etc.) will perform as they should. The programmer writes all his own support routines which perform the operation in the manner that suits the programmer's purpose. It is a bit more demanding but the outcome is usually a very efficient and compact program.

Bear in mind that when working with any language, the computer will ALWAYS do what you tell it to do; this is not always the same as what you intend it to do. It is much easier to throw the computer into a death spin when writing in machine language: you don't have SmartBasic checking your syntax and reporting other errors.

EOS STRUCTURE

The EOS uses a jump table to gain access to its functions. A jump table is a series of standard entry points which then pass control over to the routine that actually does the work The advantage of a jump table is that revisions to the operating system do not affect the addresses that the programmer uses to access the system's functions. The EOS routines can be subdivided into 9 categories, Each is composed of several routines to perform a specific series of tasks:

```
Executive Calls
```

These are the high level operations such as startup, device scanning, initialization, etc. Most of these routines are used on a cold boot to get the computer in running order. Additionally, self booting software will make use of some of these routines to set up the computer according to the programmer's requirements.

```
Console (Screen) Output
```

These routines take care of printing characters on the screen. They make use of the Video Ram routines outlined below.

```
Printer Interface
```

These routines take care of printing characters on the ADAM printer. Although parallel printer interfaces are available, the EOS had made no provision for this feature.

```
Keyboard Interface
```

These routines take care of fetching characters from the keyboard.

```
File Operations
```

These routines take care of disk/tape input output at the file level. Files can be created, opened, read, written to, closed, deleted, etc. in a fashion equivalent to the same commands in SmartBasic.

```
Device Operations
```

These are the routines which interact directly with the devices attached to the ADAM. They include the keyboard, printer, disk drives and tape drives. The Video Ram is not considered a device by the EOS.

```
Video Ram Management
```

These routines handle the movement of information (characters, sprites, shapes, colour, graphics, etc.) to and from video memory. It is these routines which actually put something on the screen.

```
Game Controllers
```

Since the joysticks are not devices, these routines handle the reading and decoding of information from the joysticks.

```
Sound Routines
```

The EOS has complex routines which are used by some software (like the arcade games) to generate elaborate sound effects.

MEMORY USAGE

```
FROM    TO      CONTENT


D390    D3FF    File Control Blocks


D400    DFFF3   1K blocks used for directory and file I/O
E000    E1F5    Video Ram routines
E1F6    E2C6    Joystick routines
E2C7    E617    Sound routines
E618    F3D9    File routines
F3DA    F445    Data used to INIT a medium
F446    F4B9    Device routines
F4BA    F4FB    Keyboard routines
F4FC    F5D4    Printer routines
F5DC    F831    Screen routines
F832    FA9D    Executive routines
FA9E    FBFD    Device Control Block Routines
FBFF    FC2F    Data tables
FC30    FD5F    JUMP TABLES
FD60    FEBF    Various data tables, storage, and stack
FEC0    FEC3    Processor Control Block
FEC4    FFFE    Device Control Blocks (one for each device)
```

The average program can use all memory below D390. This leaves the full EOS available to perform any function which many be required.

A program which requires additional memory may choose to use its own buffers for read/write operations. If it does not make use of the FILE routines, it may overwrite up to DFFF and gain an additional 3K of usable memory.

When even more memory is required, An application program may supply its own video drivers, and joystick routines (if required). It can then use memory up to F445 and gain another 5K. This leaves just the raw device drivers in the EOS. Provided no complicated video routines are required, this can be a viable alternative for very large programs.

EOS ROUTINES

This is the start of the technical section of the manual. It describes routines which can be accessed form the EOS jump table. For the sake of uniformity and ease of reading, each routine description will be on one page with the title of the routine at the top of the page. Descriptions will use the following format:

```
JUMP TABLE ADDRESS:    The address to CALL.  If you want to determine the
actual address of the routine, skip the first byte at this address (the JUMP
instruction) and extract the REAL address from the next two bytes.


ENTRY:  A       Description of values which must be
        BC      placed in registers to tell the function
        DE      what to do.  Only the relevant registers
        HL      will be shown.


EXIT:   A       Some routines return information.  The exit
        BC      values will help you make the best use of the
        DE      information provided.  You will also note that
        HL      some routines PRESERVE register values. This can
                be used to your advantage to save program space.
```

DESCRIPTION:

The description section will usually be in two parts: the first describes the purpose of the routine the second explain how the routine does its job. When a routine calls another major EOS routine, its name will usually be shown in UPPERCASE along with the page number in brackets.

EXAMPLES:

Where it can help illustrate the use of a routine, examples will be provided. These will be in machine language mnemonics with comments in the right margin.

EXECUTIVE CALLS

INITIALIZE EOS

```
JUMP TABLE ADDRESS:    FC3O


ENTRY:  none
EXIT:   B       boot device number
DESCRIPTION:    ~       ~       ~
```

When you turn on the computer or pull the reset switch, the EOS is entered at this routine. This is the one that gets the computer in working order by setting the stack, initializing tables, setting up the devices, and checking for the presence of a boot tape or disk. While this function is used only at a cold start, a programmer may use it to reboot by prompting for an "insert media" and calling FC3O to clear memory and boot the media.

The routine starts by setting the EOS stack. It then nulls out all the data tables starting at FD6l. The EOS revision number is written to FD6O. It calls SET VDP PORTS(83) and SOUND OFF(104). It then nulls all the video RAM with a call to FILL VRAM(95) and switches in the normal memory configuration with a call to BANK SWITCH(20). The next step is to perform a HARD INIT(12) to set all the devices. Next is a call to INITIALIZE FILE MANAGER(41) which sets up the file management buffers. It then scans for the presence of media in disk 1, disk 2, tape 1, and tape 2 in that order. The first to contain media is presumed to be the boot media. Block 0 of that media is loaded at C800. The next step is very important: The device number of the boot media is placed in register B and a jump is made' to G800 where the media boot code can be executed. Boot code should store the boot device number so further media activity can be performed on the default media. If no boot media is found, a jump is made to the memory resident processor (electronic typewriter) via the JUMP TO SMARTWRITER(19) routine.

EXAMPLES:

To prompt the user to insert a disk or tape and perform a boot, you can use the following instructions:

```
        LD      HL, INSERTMSG  ;prompt message.
        CALL    PRINT   ;your print to screen routine.
        GALL    FC6G    ;read keyboard (wait for character).
        JP      FC3O    ;do the EOS boot.
```

If you wish to boot a disk or tape without resetting the EOS tables or performing an initialization, you can replace the last instruction with a jump to F86A. Note that this is the direct address for EOS revision 5 and that it may not work on other revisions to the EOS. The BOOT device is still returned in B to the Boot block of the media. It is up to the programmer to pass the boot media to the main program. You could store it at (FD6F) which is the EOS-5 address of current device.

EXECUTIVE CALLS

HARD INIT

```
JUMP TABLE ADDRESS:    FC5D

ENTRY:  none

EXIT:   A       destroyed
        BC      \
        DE      preserved
        HL      /
```

DESCRIPTION:

The function of this routine is to set the processor control block which in turn controls the allocation of device control blocks. It synchronizes the Z-80 processor with the 6801processor which controls the AdamNet.

This routine starts by initializing the Processor Control Block to FECO. Compare to the SOFT INITIALIZATION(18) routine which allows the user to set the PCB. It calls HARD RESET NET(13) and the DELAY(14) to initialize the system. The device control blocks are nulled out and a call to SYNCHRONIZE CLOCKS(l5) is made. The final step is a call to SCAN FOR DEVICES(l6) to allocate a DCB for each found device.

Although it would be considered a drastic measure, this routine could be called to effectively reset the ADAM and re scan all the devices.

```
EXAMPLES:       none
```

EXECUTIVE CALLS

HARD RESET NET

```
JUMP TABLE ADDRESS:    FC60

        ENTRY:  none

EXIT:           A       zero
                C       reset port number
                others  unchanged
```

DESCRIPTION:

This function does a hardware reset on the AdamNet. It gets the Net port number from location FC28 and sends a reset request (0F). It waits awhile and sends an idle command to the net. The routine returns the port number in register C but it is very unlikely that you will ever need this information.

This routine can be called to reset the net without clearing all the DCB's

```
EXAMPLES:       none
```

EXECUTIVE CALLS

DELAY AFTER HARD RESET

```
JUMP TABLE ADDRESS:    FC3C

ENTRY:  none

EXIT:   A                       0
               others           preserved
```

DESCRIPTION:

This routine pauses for 114 clock cycles. It is used after a network reset to make sure that the devices have time to idle down. With 3 million clock cycles per second, 114 is a blink of an eye for us but almost an eternity for the Z-80 processor. For some reason, the EOS designers did not include this delay as part of the HARD RESET NET(13) routine and the delay MUST be called for proper timing.

While the routine as-is has limited value, you can make use of it to create your own timed delays by patching directly into the routine. Remember the default values so you can restore them when you are through. Firstly, address F962 has a default value of 1. For the delay to have any substance, this should be reset to 0. Address F965 has a default value of 0. This is the one we can modify to create delays ranging up to 2.5 minutes.

```
               VALUE   DELAY   VALUE   DELAY
               2       1       53      30      1/2 minute
               4       2       71      40
               7       4       88      50
               9       5       106     60      1 minute
               18      10      212     120     2 minutes
               36      20
```

EXAMPLES:

```
        LD              A,0
        LD              (F962) ,A
        LD              A,53
        LD              (F965) ,A               ;set 30 second delay.
        CALL            FC3C                    ;do it.
        LD              (F965) ,A               ;note that A is zero (save an
instruction).
        INC             A
        LD              (F962) ,A
```

EXECUTIVE CALLS

SYNCHRONIZE CLOCKS

```
JUMP TABLE ADDRESS:    FCBl
ENTRY:  none

EXIT:                  zero flag indicates success
        A       error code if NON-ZERO
        others  preserved
```

DESCRIPTION:

The purpose of this routine is to get ADAM's internals synchronized. Two synchronize requests are sent through the Processor Control Block. It is essential that these requests function properly. Otherwise, the ADAM NET may not function properly. The Synchronize Clock request must be sent each time the network is reset.

One of the byproducts of this function is to cancel out All active device control blocks. Thus a SCAN ACTIVE must be done after each synchronize clock function call.

EXAMPLES:

```
        CALL    FCB1            ;request the synchronize.
        JR      Z,COOD  ;the synchronization went ok.
```

;

;here you must decide how to handle the error

;you can call the function again or abort entirely

```
;                       telling the user that ADAM has serious problems
```

;

GOOD:

```
        CALL    FC8A            ;scan for active devices.
```

EXECUTIVE CALLS

SCAN FOR DEVICES

```
JUMP TABLE ADDRESS:    FC8A

ENTRY:  the Processor Control Block must be set

EXIT:   A              zero
               others  preserved
```

DESCRIPTION:

The purpose of this function is to find all the active devices on ADAM NET. It begins by zeroing out all Device Control Blocks. This will effectively remove devices which are no longer on line. On the other hand, the scan will pick up devices which have been powered up after the ADAM has turned on. Thus you should make sure that all devices are turned on before calling this function.

An area of 314 bytes is cleared above the 4-byte processor control block. It represents the 15 DCB's which are 21 bytes each. A count is kept in PCB+3 of the total number of devices found and a 21-byte DCB is set up for each device found. If you wish to check the number of devices found, you will have to FIND the PCB and examine its byte number 3.

EXAMPLES:

```
        CALL    FC8A            ;set up the DCB's.
        CALL    FC5A            ;get the PCB address into IY.
        LD      A, (IY+3)       ;number of devices.
```

;

;now you can tell your program or viewer

;how many devices you found

;

EXECUTIVE CALLS

RELOCATE PCB

```
JUMP TABLE ADDRESS:    FC7B

ENTRY:  HL      Address to relocate PCB to

EXIT:   A       83H    zero flag set
        BC      preserved
        DE      preserved
        HL      new PCB address
```

DESCRIPTION

This function is used to relocate the Processor Control Block. Although I can see no particular reason for this function, COLECO must have thought that it might be a requirement with some of the expansion hardware or for special purpose applications.

The routine begins by getting the current PCB location since commands must still be sent through the current PCB location. The new address is written to the PCB data area and a SET command is sent to the NET. The routine then loops endlessly waiting for the net to acknowledge. If it does not, the system will hang up; there is no time out for this operation. The last step is updating the CURRENT PCB location in memory.

Once the PCB has been relocated, all devices are effectively put off line. You have to issue a SCAN FOR DEVICES (16) to restore all devices. I am not sure if it is necessary to perform a hard reset, but to be on the safe side, I would recommend using the SOFT INITIALIZATION(18) instead of this routine to relocate the PCB.

EXAMPLES:

```
        LO      HL, 8000H               ;address to relocate to.
        CALL    FC7B                    ;move the PCB please.
        CALL    FC8A                    ;scan for devices.
```

EXECUTIVE CALLS

SOFT INITIALIZATION

```
JUMP TABLE ADDRESS:    FC8D

ENTRY:  HL      New PCB address

EXIT:   A       destroyed
        others  unchanged
```

DESCRIPTION:

The function of this routine is to set the processor control block which in turn controls the allocation of device control blocks. It synchronizes the Z-80 processor with the 6801 processor which controls the AdamNet.

This routine starts by initializing the Processor Control Block to the value supplied in HL. After that, its execution is virtually identical to HARD INIT(12). It does a hard reset, a delay, a synchronize, and a scan for devices.

Since you need to do all these housekeeping functions when you want to relocate the PCB, it is best to call this vector instead of RELOCATE PCB.

EXAMPLES:

EXECUTIVE CALLS

EXIT TO SMARTWRITER

JUMP TABLE ADDRESS:FCE7

```
ENTRY:  none

EXIT:   none
```

DESCRIPTION:

This is the routine used by the COLD BOOT sequence (power up) when no bootable media is found. It bank switches in the SMARTWRITER ROM and jumps to it. If you wish to abort a program you can call this routine to effectively halt all operations.

Since this routine does a BANK switch, it is essential that the STACK be located in upper memory when it is invoked. Although it is not a standard location, I suggest using 65535 since it is used only temporarily and the SmartWriter will set its own stack when it takes over.

EXAMPLES:

```
CALL    ABORTYN         ;ask user to abort.
RET     NZ                      ;changed his mind.
LD      SP,FFFF         ;set stack to upper half.
JP      FCE7                    ;let's get out.
```

EXECUTIVE CALLS

SWITCH MEMORY BANKS

```
JUMP TABLE ADDRESS:    FD14

ENTRY:          A       desired memory configuration

EXIT:   A       current configuration
        B       current configuration
        C       memory port number
        DE      preserved
        HL      preserved
```

DESCRIPTION:

This routine can switch either the upper or the lower 32K of memory to any of 4 configurations. It simply sends the requested configuration to the memory bank switch port. Although it is a simple routine, it is easy to get into trouble with it. When the memory configuration is changed, program control may wind up in the switched bank with unpredictable results. Since the EOS is in the top half of memory, the TOP HALF should never be switched using this routine. If you do, the EOS will effectively disappear, and your program will crash. It is also essential that the Routine which calls this function be also located in the upper half of memory (above 8000H) or program control will also be lost.

Consult the appendix for ADAM's memory bank configuration.

EXAMPLES:

This is the breakdown of the routine used to switch banks. If you wish to switch in the upper half of memory, you will require a routine similar to this in the lower 32K to make the switch.

```
        LD      A, (FC27)               ;get the port number.
        LD      (SAVE) ,a               ;save the address in lower memory
somewhere.
        LD      C,A                     ;put port in C.
        LD      A,config                ;put in the configuration you want.
        OUT     (C)                     ;switch it over.
```

;

;do what you wish to do in here

;

```
        LD      A, (SAVE)               ;get the port back.
        LD      C,A
        LD      A,normal                ;bring the EOS back into top half.
        OUT     (C)                     ;send request to port.
```

CONSOLE OUTPUT

INITIALIZE CONSOLE

```
        JUMP TABLE ADDRESS:    FC36


ENTRY:          B       number of columns (0 to 31)
        C       number of lines (0 to 23)
        D       home column
        E       home row
        HL      pointer to pattern name table
```

EXIT: all registers lost

DESCRIPTION:

This routine is used to set up a WINDOW for screen display. Registers B and C contain the, number of rows and columns while registers D and E contain the upper left corner of the window. The other parameter required is base address of the Pattern Name Table. If you have previously set up VDP, you should have NOTED what that address was. If you; are using a routine in conjunction with SmartBasic, the default pattern name table address is 1800H.

The routine stores lines and columns, sets up minimum and maximum values for ROW and COLUMN based on the supplied parameters. You can set up many windows and jump around between them by repeating calls to this routine. When this routine exits, the default cursor (an underline) is placed in the top left corner of the window. You can then send a move cursor command (see console display page 18) to place it at the appropriate location in the window.

EXAMPLES:

This routine sets up a 12 line window in the centre of the screen:

```
        LD      B,20                    ;20 columns.
        LD      C,12                    ;12 rows.
        LD      D,6                     ;home column.
        LD      E,6                     ;home row.
        LD      HL, 1800H               ;or whatever your pattern address is.
        CALL    FC36                    ;set up the screen.
```

CONSOLE OUTPUT.

CONSOLE DISPLAY REGULAR

JUMP TABLE ADDRESS: FC33

ENTRY: A character to send

EXIT: all registers preserved including A

DESCRIPTION:

This routine prints whatever character is in the accumulator to the screen. It presumes that the VDP has been set up and that a window has been defined. It will print ALL characters including the graphic representation of the control codes (0-31). If you wish to send a control CODE, use the routine on the next page.

The routine first sends the character to video RAM. Then it advances the cursor position, going to the next line if required. If the cursor is on the last line, the screen is scrolled.

EXAMPLES:

This subroutine is used to print an incoming message in register HL. It presumes that the string to print is followed by a null (ASCII 00).

PRTSTR:

```
        LD      A, (HL)
        OR      A
        RET     Z                       ;the string was over.
        CALL    FC33                    ;print the thing.
        INC     HL                      ;remember HL was preserved.
        JR      PRTSTR          ;loop until end of string.
```

CONSOLE OUTPUT

CONSOLE DISPLAY SPECIAL

JUMP TABLE ADDRESS: FC39

ENTRY: A character to print or PLACE CURSOR request
D column to go to if PLACE CURSOR
E row ,to go t9 if PLACE CURSOR

EXIT: all registers preserved including A

DESCRIPTION:

This routine, like console display on previous page will print a character on the defined window. It begins however by checking for 12 special control codes. If it finds one of these, it executes the following control functions:

CONTROL CRARACTER	KEYBOARD EQUIVALENT	FUNCTION PERFORMED
08	BACKSPACE	move cursor left one
0A	^J	move cursor down one line (line feed)
0C	^L	clear screen and home cursor
0D	RETURN	return cursor to start of line (must send line feed if new line wanted)
16	^V	delete to end of line
18	^X	delete to end of screen
1C	^\	place cursor at position DE
80	HOME	home the cursor (no clear)
A0	up arrow	move up
Al	right arrow	move right
A2	down arrow	move down
A3	left arrow	move left

Note that there is no CURSOR ON or CURSOR OFF. In EOS-5, you can replace 3 bytes starting at F658 with. ZEROS to turn the cursor off. Be sure to remember the values to turn it back on again.

EXAMPLES:

When the string STRING is printed, it will send the cursor home, skip 2 lines, print HELLO, and clear the rest of the screen. Note that the first two lines would not be erased by this operation:

```
        LD      HL,STRING
```

REPEAT:

```
        LD      A, (HL)
```

```
                OR      A
                JR      Z,CONT
                CALL    FC39
                INC     HL
                JR      REPEAT
STRING:  DB     80H,A2H,A2H, 'HELLO' ,18H,0
```

CONT:

;Program continues here

PRINTER INTERFACE

PRINT CHARACTER

	JUMP TABLE ADDRESS:	FC66
ENTRY:	A	character to print
EXIT:		zero flag set - successful
	A	error code if NZ
	others	preserved

DESCRIPTION:

This routine sends one character to the printer. Compare to print buffer on next page.

If the printer is not found, error code number 1 is returned in the accumulator. This means the NET does not know the printer is there. You should re scan for active devices and if the printer is still not found, you should abort with a warning to the user.

If the printer is busy, the routine returns error code number 2 in the accumulator. You can wait a bit and retry.

If the printer is off-line (i.e. not working), error code 3 is returned. If this happens, do NOT retry; it will serve no purpose.

If the printer is ready, the character is placed in the print buffer and the printer is asked to print it via the PRINT BUFFER routine on the next page.

EXAMPLES:

RETRY:

```
        LD      A, character            ;what you want to print.
        CALL    FC66                    ;ask printer to do it.
        JR      Z,GOOD
        AND     127                     ;strip acknowledge bit if any.
        DEC     A,                      ;was A one?
        JP      Z,NODCB         ;try and find the printer
        DEC     A                       ;was A two?
        JR      Z, RETRY        ;let's loop until it works.
        DEC     A                       ;was A 3?
        JP      Z,PTRDIED       ;printer died?  Now what do we do?
```

GOOD:

;

;continue program

PRINTER INTERFACE

PRINT BUFFER

JUMP TABLE ADDRESS: FC63

ENTRY: HL pointer to string (terminated with ASCII 03)

EXIT: Zero Flag set - successful
A error code if non-zero
others preserved

DESCRIPTION:

This routine will print a string pointed to by register HL. The bytes are sent to the printer in groups of 16. If the string (or remainder) is less than 16 then the partial string is sent and then the routine exits. You can use this routine to print strings of any length: an entire document if you wish. The nice thing about it is that you only need to place a "03" at the end of the string.

The routine stays in control until the entire string is printed. Since the printer is a very slow device, you may choose the alternate method using START WRITE and END WRITE explained on the following pages. The error codes are the same as for PRINT CHARACTER:

1 No printer
2 Printer busy
3 Printer off line (idle)

EXAMPLES:

This is an example of a subroutine used to print a string. The subroutine expects HL to point to the data and takes care of errors:

```
        LD      HL,STRING               ;print this.
                CALL    PRINTIT         ;ask subroutine to do it
```

;

;

PRINTIT:

```
        CALL    FC63                    ;print the whole thing.
        RET     Z                       ;OK..
        AND     127                     ;strip acknowledge bit.
        DEC     A
        JR      Z,NOPRINT       ;No DCB found.
        DEC     A
        JR      Z, PRINTIT              ;retry.if busy.
```

;if we get here then the printer is off line.

NOPRINT:

;if we get here then try FINDING the printer again.

PRINTER INTERFACE

PRINTER STATUS

```
JUMP TABLE ADDRESS:     FC84

ENTRY:   none

EXIT:                   Zero flag set - READY
         A              error code if non zero
         IY     address of DCB (only if no error) others preserved
```

DESCRIPTION:

Although it might be considered a bit of a waste, this is a handy routine to use. You don't need to remember the DEVICE number of the printer(2) and call REQUEST STATUS.

If you wish to check the availability of the printer prior to sending a character or string, use this routine. When used in conjunction with the routines on the next pages, it allows you to do other things while you are waiting for the printer to be READY.

The most common errors are

```
         1              no DCB
         2              busy (it is printing something)
         3              off line (idle)
```

EXAMPLES:

This routine waits for the printer to be ready and then sends the character in A to be printed.

```
         LD     C,A             ;save the character.
```

RETRY:

```
         CALL   FC84            ;check status.
         JR     Z,READY
         AND    127             ;strip acknowledge bit if any.
         CP     2
         JR     Z , RETRY
```

;

;process other errors here.

;

READY:

```
        LD      A,C
        CALL    FC66            ;print it.
        RET
```

PRINTER INTERFACE

START PRINT CHARACTER

JUMP TABLE ADDRESS:	FC9F	
ENTRY:	A	character to print
EXIT:		Zero flag set - successful
	A	error code if non-zero
	others	preserved

DESCRIPTION:

This routine is used to set up background printing. If the printer is READY, it asks it to print a character and returns the control to the user. Since the printer is very slow, you can perform a few other tasks and use END PRINT CHARACTER to check if the printer is done.

The error codes are the same as the other print routines.

EXAMPLES:

This subroutine expects a character to print in A. It sends it to the printer first (since it is slow); then to the screen. Finally it waits until the printer is done prior to returning.

```
        LD      C,A             ;save the character.
        CALL    FC9F            ;start print.
        JR      NZ, PERR
        LD      A,C             ;get character back.
        CALL    FC66            ;print to screen.
        JR      NZ,SERR
```

RETRY:

```
        CALL    FC42            ;is printer done?
        JR      NZ,PERR  ;must check error FIRST.
        JR      NC,RETRY        ;if no error and not completed then wait.
                RET                     ;yes we are finished.
```

PERR: ;process printer error here.

SERR: ;process screen error here.

PRINTER INTERFACE

END PRINT CHARACTER

JUMP TABLE ADDRESS: FC42

ENTRY: none

EXIT:
CARRY SET		completed
NO CARRY and ZERO		not completed
NON ZERO		error
A	error code if no carry and non-zero	
others	preserved	

DESCRIPTION:

This is the companion routine to start print character. It finds the DCB and returns an error if not found. It will also return an error if the printer is off line or is busy. If the printer is done, the carry flag is set prior to returning to caller.

This routine does not retain control while a character is being printed. It allows the programmer to do some co-processing.

EXAMPLES:

The routine shown in start print buffer is not the most effective way of maximizing throughput since the print routine does only one task and spends the rest of the time waiting. Once it is done, the programmer still uses valuable processing time to FETCH another character to send to the routine. Another approach is to send the FIRST character to print using START PRINT and not check for completion. All subsequent characters can be sent to this routine which waits for the first to be done prior to sending another.

```
        LD      A,CHAR
        CALL    FC9F            ;this is the first character printed.
```

;

;

```
        LD      A,CHAR
                CALL    PRINTC  ;all subsequent chars to this routine.
```

;

;

PRINTC:

```
        LD      C,A             ;save new character to send.
```

WAIT:

```
        CALL    FC42            ;end print the last one?
        JR      NZ,PERR  ;oops.
        JR      NC ,WAIT
        LD      A,C
        CALL    FC9F            ;start to print this one.
        RET     Z               ;good start.
        PERR:                           ;here we have a printer problem.
```

PRINTER INTERFACE

START PRINT BUFFER

JUMP TABLE ADDRESS: FC9C

ENTRY:	HL	points to a string terminated with ASCII 03
EXIT:		Zero Flag set - success
	A	error code if non-zero
	others	preserved

DESCRIPTION:

This routine was intended to print up to 16 characters on the printer in background. It requires the use of the companion END PRINT BUFFER routine on the next page. The routine starts by finding the printer and returns an error if the printer is not found or busy. It then looks through the input string for a 03 in the first 16 characters. If none is found, it STARTS to print the first 16 characters. The routine may have been intended to handle longer strings since it keeps track of where it is (when longer than 16).but does not seem to have been fully implemented.

Since the ADAM printer is slow, you will have about a second and a half (almost an eternity for the Z-80) to perform some other work while the printer is busy doing this task. You can periodically check if the printer is done and then send more characters When you send your last 16 (or less), you can just carry on doing something else, provided you check if the printer is done prior to starting another print sequence.

EXAMPLES:

See example on next page which sends a LONG string to the printer.

END PRINT BUFFER

JUMP TABLE ADDRESS: FC3F

ENTRY: none

EXIT:		
	CARRY SET	completed
	NO CARRY	and ZERO not completed
	NON ZERO	error
	A	error code if no carry and non-zero
	others	preserved

DESCRIPTION:

This is the companion routine to START PRINT BUFFER(33). It allows you to perform other tasks and check if the printer is done the 16 (or less) characters you sent. Once completed (or error), you can send another with START PRINT BUFFER(33).

EXAMPLES:

The following routine sends a LONG string (which has a 03 at the end) to the printer. It makes use of the wait time to call another routine which could perform any small task.

```
        PUSH    HL              ;save start of string.
        ID      A,3
        ID      BC,65535

                CPIR                    ;find the 03.

        LD      HL,0

        OR      A

                SBC     HL,BC           ;this is the number of characters

        SRL     H

        RR      L

        SRL     H

        RR      L

        SRL     H
```

```
        RR      L
        SRL     H
                RR      L               ;this is the number of 16-byte packets to send.
        LD      B,H
                LD      C,L
                INC     BC              ;correct count
                POP     HL              ;get back original pointer;
```

MORE:

```
                CALL    FC9C            ;start to print.
        JR      NZ,ERROR
```

WAIT:

```
        CALL    ABCD            ;routine better save BC and HL.
        CALL    FC3F            ;is printer done?
        JR      NZ, ERROR
        JR      NC,WAIT
        LD      DE,16
        ADD     HL,DE           ;advance pointer
        DEC     BC
        LD      A,B
        OR      C               ;is BC zero?
        JR      NZ,MORE
```

KEYBOARD STATUS

JUMP TABLE ADDRESS:	FC81	
ENTRY:	none	
EXIT:	Zero flag set - no errors	
	A	error code if non-zero
	IY	address of DCB (only if no error)
	others	preserved

DESCRIPTION:

This routine is used to find out if the keyboard is active. It will return a 1 in the accumulator if there is no DCB. This means that the keyboard was not found. It may indicate hardware failure or just that the keyboard was not plugged in when the system was turned on. Error code 3 means that the device is idle.

While it may be useful to check the keyboard status in this manner, you still need another EOS call to actually fetch a character from the keyboard. This routine could be used in a power up sequence to ensure the keyboard is there.

EXAMPLES:

KEYBOARD INTERFACE

READ KEYBOARD

JUMP TABLE ADDRESS: FC6C

ENTRY: none

EXIT: Zero flag set - no error

A	character if non zero else it contains the error code
others	preserved

DESCRIPTION:

This routine starts by calling START READ KEYBOARD(28). If this fails, an error code is returned. The routine then keeps calling END READ KEYBOARD(29) until a character or an error is received. The routine remains in control until a key is pressed; this is like SmartBasic's GET command.

For simple keyboard input, this routine is adequate. It should not be intermixed with the START and END read commands as you may wind up missing a character.

EXAMPLES:

This routine requests characters from the keyboard and places them in a buffer at (HL) until <RETURN> is pressed.

```
        LD      HL,BUFFER       ;point to a memory area for data.
        CALL    GETENTRY        ;fill it up please
```

;

;more program here.....

;

GETENTRY:

```
        CALL    FC6C            ;read keyboard.
        JR      NZ, ERROR       ;something went wrong.
        LD      (HL) ,A         ;save character even if <CR>
        CP      13
        RET     Z               ;yes we have a full entry.
        INC     HL
        JR      GETENTRY        ;get another character.
```

KEYBOARD INTERFACE

START READ KEYBOARD

JUMP TABLE ADDRESS: FCA8

ENTRY:	none	
EXIT:		Zero flag set - successful
	A	error code if non-zero
	others	preserved

DESCRIPTION:

This routine asks the keyboard to fetch a character. The Keyboard will not acknowledge until it a key has been pressed. After that, the only thing the programmer needs to do is call END READ KEYBOARD(39) to see if there is a character waiting.

If there is no DCB or the keyboard is off line, an error code will be returned.

EXAMPLES:

see page 39

KEYBOARD INTERFACE

END READ KEYBOARD

JUMP TABLE ADDRESS: FC4B

ENTRY:	none	
EXIT:	No Carry	no character waiting.
	Carry set	we have an answer (could be an error).
	Zero flag set	No error.
	A	keyboard character if carry set.
		error code if zero flag set.

DESCRIPTION:

This routine asks the keyboard what is happening. It returns a complex set of readings using the Carry and Zero flags. It is important to check these in the correct order to trap all the conditions. First, check the carry flag to see if the operation is complete. The next step is to check the zero flag for errors.

A combination of START and END read is the most effective way of reading the keyboard since it allows you to check the keyboard for an interrupt command while some other process is ongoing (similar to catching CONTROL-S when listing in SmartBasic).

EXAMPLES:	Routine to check for abort within a process.

IGNORE:

```
        CALL    FCA8                    ;start read (only if not previously done).
    JR      NZ,ERROR
    MORE:
        CALL    PRINTCHR                ;send once character to the screen/printer.
    CALL    FC4B                    ;is there anything?
        JR      NC,MORE                 ;so let's print some more.
    JR      NZ,ERROR        ;we have a keyboard error.
    CP      19                      ;is it CONTROL-S?
    JR      Z,WAIT          ;wait until other key pressed.
    CP      3                       ;is it CONTROL-C?
        JR      NZ,IGNORE       ;if not ignore key but reset read.
```

;abort code goes here.

Routine to wait for a character and return it in B.

STARTGET:

```
    CALL    FCA8                    ;enter here is need to start the keyboard.
        JR      NZ,ERROR                ;start read.
```

```
WAITCHAR:                                   ;enter here if start already done.
        CALL    FC4B                        ;end read.
        JR      NC ,WAITCHAR  ;wait until acknowledge.
        JR      NZ, ERROR           ;bad keyboard again.
        LD      B,A                         ;save character.
        CALL    FCA8                        ;restart read for next time.
        JR      NZ, ERROR
        RET
```

FILE OPERATIONS

INITIALIZE FILE MANAGER

JUMP TABLE ADDRESS: FCBA

ENTRY: DE pointer to 3K of memory for FCB transfer buffers

HL pointer to 105 bytes for three File Control Blocks

EXIT: all registers preserved

DESCRIPTION:

This routine establishes three file control blocks and 3 1K buffers for the data sent/received through the FCB's. It also sets up and sets to AVAILABLE three 35 byte file control blocks.

The EOS INIT sequence sets up three default FCB's and buffers. programmer, however, may wish to move these elsewhere.

EXAMPLES:

This routine shows the default setup for the FCB's:

```
LD      HL,D390         ;reserve 112 bytes for FCB's (round number?)
LD      DE,D400         ;3K ending at EOOO.
CALL    FCBA            ;set up file manager.
```

FILE OPERATIONS

CHECK DIRECTORY FOR FILE

JUMP TABLE ADDRESS:		FCFC	
	ENTRY:	A	device
		HL	pointer to file name
EXIT:			Zero flag set means file found
		A	error code if non zero
		BC DE	Start block of file if found
		HL	preserved

DESCRIPTION:

This routine ALWAYS uses the first file control block which is reserved for directory work. It sets the FCB to read block 1 of the medium which is presumed to be the directory. It extracts the directory size from byte 12 of block 1 of the media. The next step is to check if this is a directory by looking for the 55 AA 00 FF at bytes 13, 14, 15, and 16. It then updates the file counters and begins searching for the file name. First, it looks for a BLOCKS LEFT entry; if found, the routine aborts with error code 5 in A. Deleted entries are also skipped.

The next part is to compare the file name with the user supplied name in (HL). The search routine has the capability of skipping the FILE TYPE byte based on a flag at FDD8. If non-zero, then the file type is expected to match. In order to set this flag properly, you should use the FIND FILE 1 or FIND FILE 2 routines.

The routine continues scanning entries, reading additional blocks if necessary in order to find the file. If the file is found it's start block is placed in registers BCDE and the file name is preserved in HL.

EXAMPLES:

```
        LD      A,8                     ;read tape 1.
        LD      HL, FILE        ;look for this file.
        CALL    FCBA                    ;find the file.
        JR      Z,FOUND
        CP      5
        JR      Z,NOTFOUND      ;reached BLOCKS LEFT.
```

;

;interpret and process other errors like bad block here.

;

```
FILE:   DB      'thisfileA',03          ;note the file type and the 03.
```

FIND FILE 1

```
        JUMP TABLE ADDRESS:    FCCC


        ENTRY:          A       device
        DE      pointer to file name
        HL      pointer to user buffer


EXIT:           Zero flag set means file found
        A.      error code if NZ
        BC DE   file start block if found
        (HL)    directory entry if found
        HL      preserved
```

DESCRIPTION:

This routine looks for a match of a file name and file type. To search only for a name, use FIND FILE 2 on the next page It sets the search flag and calls CHECK DIRECTORY FOR FILE (42). If the file is found, it copies 23 bytes from the directory entry (excluding the date bytes) to the user buffer pointed: to by HL. Registers BCDE also contain the start block of the file.

While this routine may be more informative than CHECK DIRECTORY, the programmer must ensure that incoming registers are properly set. Note also that this routine uses DE to point to the name while the other uses HL.

EXAMPLES:

This routine finds a file and determines the size of the file (in blocks).

```
        LD      A,4                     ;use disk 1.
        LD      DE,FILE         ;point to the file name.
        LD      HL, BUFFER      ;store the entry here.
        CALL    FCCC                    ;find the file.
        JR      NZ,ERROR
        PUSH    HL                      ;HL buffer pointer was saved.
        POP     IX                      ;copy HL to IX.
        LD      L, (IX+19)
        LD      H, (IX+20)              ;get the file size into HL.
                                        ;note BCDE have the start block.
```

;

;

;

```
        ERROR:
```

;

;interpret and process other errors like bad block here.

;

.

```
FILE:   DB      'thisfileA',03          ;note the file type and the 03.
```

FILE OPERATIONS

FIND FILE 2

```
JUMP TABLE ADDRESS:    FGFF

ENTRY:       A device
        DE      pointer to file name
        HL      pointer to user buffer

EXIT:   Zero flag set means file found
        A       error code if NZ
        BC DE   file start block if found
        (HL)    directory entry if found
        Hi.     preserved
```

DESCRIPTION:

This routine is identical to FIND FILE 1 on the previous page with only one exception. It sets the FILE NAME ONLY flag of the file search. Thus a file will be found whether its type is A a H h or any other type. When calling this routine, your FILE NAME POINTER must have a file type because the length of the search string is important.

This routine can be useful prior to making a file (of any type). You may wish to check if the same name exists under another type to avoid conflicts. It can also be useful when loading ENCODED file names which specify their function or use according to the file type. If the user says LOAD FILE JUNK, you can find the file, check the file type, and verify that it is suitable to the desired operation.

EXAMPLES:

This routine finds a file and extracts the file type byte for comparison.

```
        LD      A,4     ;use disk 1.
        LD      DE,FILE ;point to the file name.
        LD      HL, BUFFER      ;store the entry here.
        CALL    FCFF    ;find the file.
        JR      NZ, ERROR
```

;

;since we know the file name was correctly parsed, we need no safety valve

;

```
        LD      A,03    ;find. the end of name.
        PUSH    BC      ;save start block.
        LD      B,1     ;look a long way.
        CPIR
        POP     BC      ;get back start block.
        DEC     HL      ;now points to the 03.
```

```
        DEC     HL      ;now points to file type.
        LD      A, (HL)         ;here it. is now check it.
```

;----------

ERROR:

;----------

```
FILE:                   DB      'thisfileA' ,03  ;note the file type and the
03.
```

FILE OPERATIONS

FIND FILE IN FCB

```
JUMP TABLE ADDRESS:     FCF0

          ENTRY:  HL        Pointer to file name

EXIT:             Zero flag set means file found
                  A                   error code if non-zero (including not found)
                                              or contains FCB number if found
          B       file mode if file found
          HL      points to File Control Block if found otherwise preserved
          others  preserved
```

DESCRIPTION:

This routine checks the second and third file control blocks for a match of the user supplied file name. If the file is not found, error code 5 is returned in A with the zero flag reset. If the file is found, the file mode is returned in register B, the FCB number in A, and HL points to the start of the FCB so you can decode the DATA in the file.

EXAMPLES:

```
          LD        HL, FILE   ;point to the file name.
          CALL      FCF0               ;find the file.
          JR        NZ , NOTFOUND      ;this is the only possible error.
```

;

;now we have the file mode, the FCB number and a pointer to the FCB

;

;----------

NOTFOUND:

;----------

```
FILE:     DB        'thisfileA',03          ;note the attribute and the 03.
```

FILE OPERATIONS

CHECK FILE MODE

JUMP TABLE ADDRESS:		FCF9	
ENTRY:	HL	directory entry	
	IX		pointer to FCB
EXIT:	Zero flag set mode OK		
	A	error code if non zero	
	others	preserved	

DESCRIPTION:

This routine examines the file mode in the incoming FCB in IX. If the mode is out of range, error code 17 is returned. The Attributes of the file entry are then checked against the requested mode; If the file is READ PROTECT and the request is READ or WRITE PROTECT for WRITE, error code 20 is returned.

While this routine is called by other EOS routines such as READ, WRITE, DELETE, etc. There may be some occasions where the programmer may wish to perform his own mode check. The calling syntax is also strange. If IX points to a File Control Block, it also points to a file name. So why do I also need it in Hi.?

The File modes are:

0	unused (FCB is available for use)
1	read
2	write
3	update (e.g. change attributes) or Read/Write
4	fill rest of space on medium (MAKE with size of 0)

EXAMPLES:

```
        LD      HL, FILE  ;point to the file name.
        CALL    FCFO      ;find the file in FCB.
        JR      NZ , NOT FOUND    ;this is the only possible error.
        PUSH    HL
        POP     IX        ;copy to IX as well.
        CALL    FCF9      ;check mode.
        RET     Z         ;mode OK.
        CP      17
        JR      Z, BADMODE        ;mode is not 1 2 3 4.
        CP      20
        JR      Z,ILLEGAL         ;mode incompatible with attributes.
```

FILE OPERATIONS

MAKE FILE

```
JUMP TABLE ADDRESS:    FCC9

ENTRY:  A       device
        BC DE   file size in bytes
        HL      pointer to file name

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine begins by determining the number of K required for the file. If BCDE are zero, then the file will be allocated ALL the remaining space on the medium. This technique is used when you create a file without yet knowing how long it will be. It then calls READ BLOCK(74) to get the. directory and verifies that it is indeed a directory. It then looks through the directory entries to find a HOLE of the right size. If the file length' is zero, then it will scan through until it finds BLOCKS LEFT or runs out of room. At the same time, it calls FIND FILE(43) to make sure the file does not exist. After scanning the entire directory, it checks to see if a suitable file was found. The correct directory block is re-read and the file data is copied to the directory entry. The directory is then re-written to disk via WRITE BLOCK(79).

Note that this routine only creates a directory entry of the size requested. The data is not written to the file.

EXAMPLES:

```
        LD      A,8             ;use tape 1.
        LD      BC , 0
        LD      DE, 12345       ;12,345 bytes in the file.
        LD      HL, FILE        ;point to name of file.
        CALL    FCC9    ;make the directory entry.
        JR      Z,GOOD
        CP      11
        JR      Z,TOOBIG        ;file too big (bigger than 67MEG).
        CP      6
        JR      Z,EXISTS        ;file name exists in directory.
        CP      12
        JR      Z,DFULL         ;directory is full.
        CP      13
        JR      Z,MFULL         ;media is full.
```

;

;more errors like bad block,. etc.

UPDATE FILE IN DIRECTORY

```
JUMP TABLE ADDRESS:    FCCF

ENTRY:  A       device
        DE      pointer to file name
        HL      address of user File Control Block

EXIT:   Zero flag set no error
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine starts by finding the file in the directory; it aborts if it is not, found. It moves 23 bytes (everything except the date) into the directory entry. It then calls WRITE BLOCK(79) to re-write the directory entry.

This routine is used to update a file's attributes, adjust the file size, etc. It is used by CLOSE FILE(53) to update the directory when finished with file. It presumes the file already exists on the selected media.

EXAMPLES:

This routine finds a file, changes the attributes, and rewrites the entry.

```
        LD      A,4     ;use disk 1.
        PUSH    AF      ;save device.
        LD      DE,FILE ;point to the file name.
        PUSH    DE      ;save file name.
        LD      HL,BUFFER       ;store the entry here.
        CALL    FCCC    ;find the file.
        JR      NZ , ERROR
        PUSH    HL      ;HL buffer pointer was saved.
        POP     IX      ;copy HL to IX.
        SET     3, (IX+12)      ;set the system bit of the file.
        POP     DE      ;file name.
        POP     AF      device.
        CALL    FCCF    ;re write entry
        JR      NZ, ERROR       ;let's find out what went wrong.
```

;

;

ERROR:

```
        AND     127     ;strip acknowledge bit if any.
;       interpret errors here.
;               1               DCB not found
;               2               Device busy
;               3               Device off line
;               5               File not found
;               22              Device error
```

OPEN FILE

JUMP TABLE ADDRESS: FCC0

```
ENTRY:          A       device
                B               mode
        HL      pointer to file name

EXIT:           Zero flag set means no error
        A       file number OR error code
                B       file number if no error
        others  preserved
```

DESCRIPTION:

This routine sets up a File Control Block for the user to access the requested file. It begins by checking the 2 File Control Blocks If both are busy error code 7 is returned in register A. It then tries to find the file in the directory. If the file is found, it calls MODE CHECK(48) to ensure the request can be processed. It then sets the control bytes in the FCB (current block, last block, etc.) so the file is ready to be accessed. It also sets up the buffer associated with the FCB as the position in the file and copies the start block of the file to the current block in the FCB.

At this point, the routine checks if the request is a write; if so the job is done. If the request is read, it sets an internal flag (bit 7 of the mode) only if the file size is 1K. This warns the other file routines not to READ another block from this file. It then calls READ BLOCK(74) to pre read the first block of the file.

This routine has two problems. If the file size is zero, it does not initialize the File Control Block properly and may wreak havoc on the media on write operations. The next problem is that it does not tell you where the raw data from the medium is placed. It presumes that you will call READ FILE(54) or WRITE FILE(56) to perform transfers.

It is very important when opening a file to REMEMBER what the FILE NUMBER is. All other access to the file will be made via the FILE NUMBER. The EOS allows you to have two files- opened simultaneously, although they must be on the same medium. Otherwise, a fourth FCB would be required to hold the directory of the other medium. This may have been an oversight in the design of EOS. At the time, there was only the tape drive to worry about. This means that you can't use exclusively the EOS FILE FUNCTIONS to copy a file from one medium to another without buffering.

EXAMPLES:

```
        LD      HL, FILE        ;point to a file name.
        LD      A,8             ;use tape 1.
        LD      B,1             ;let's read the file.
        CALL    FCC0            ;open the file.
        JR      NZ, ERROR       ;let's see what went wrong.
        LD      (FNUM),A        ;save the file number for later.
```

FILE OPERATIONS FILE I/O

CLOSE FILE

```
        JUMP TABLE ADDRESS:    FCC3
ENTRY:  A       File number


EXIT:                   Zero flag set successful
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine is used to close off a file and free up its File Control Block. It begins by checking the validity of the file number (can only be 1 or 2 since there are only 2 File Control Blocks for FILES). If the file number is bad, error code 9 is returned. If a request is made to close a file in an FCB that is empty, error code 9 is also returned. It then checks the WRITTEN BIT in the file mode (40H) to see if it the buffer needs flushing. If so, the buffer is written out via WRITE BLOCK(79). If the WRITE is unsuccessful, the routine aborts without clearing the FCB. This gives you a chance to retry. The last step is a call to UPDATE FILE IN DIRECTORY(50). Regardless of the outcome of this routine, the FCB is marked as empty and errors (if any) reported to the user.

This last step can also lead you into trouble. You could have written several K to a file, the data would be there, but if the directory is not updated, you have no way of finding that information again. If you get a BAD BLOCK/CRC error, the only way of knowing if it was when the buffer was flushed or when the directory was written is by checking if the FCB is closed (perhaps that was the intent).

EXAMPLES:

```
        LD      A,( FNUM)       ;get back the file number when opened.
        CALL    FCC3            ;close it off.
        JR      Z,COOD          ;no errors.
        AND     127             strip acknowledge bit if any.
        CP      9
        JR      Z , BADNUM      ;file number is bad.
        CP      3
        JR      Z,IDLE          ;device off line.
        CP      24
        JR      Z,BADDIR        ;directory fails check bytes.
        ;etc.
```

READ FILE

```
        JUMP TABLE ADDRESS:    FCD2


ENTRY:  A       file number
        BC      number of bytes required
        HL      where to put the data


EXIT:   Zero flag set means success
        A       error code if non zero
        BC      number of bytes actually transferred
        others  preserved
```

DESCRIPTION:

This routine will read a specified number of bytes form a file which has been OPENED(52). It begins by checking the validity of the file number and that the mode is compatible with the request. If the FCB is on the LAST BLOCK, it checks to see if it can deliver the requested number of bytes from the current position on the buffer. Otherwise, it checks how many bytes are remaining in the current block to fill the request. If there is not, it starts by delivering the last remaining bytes in the buffer and tries to read in the next block of the file. This cycle continues until file end is reached or all the bytes are delivered, making successive calls to READ BLOCK(74). If all bytes cannot be delivered, an end of file error code 10 is returned in A. Other error codes indicate other problems such as bad block, device idle, etc.

You can use this routine to read from 1 to 65535 bytes from the input file. Prior to making the call, be sure you have sufficient buffer space starting at (HL) to accept all the bytes requested. The handy thing about this routine is that it allows you to read in an entire file without knowing it's size. Just ask for a large amount and if error code 10 is returned, BC will tell you the file size in bytes.

EXAMPLES:

```
                LD      HL, FILE        ;point to a file name.
                LD      A, 8            ;use tape 1.
                LD      B,1             ;let's read the file.
                CALL    FCC0            ;open the file.
                JR      NZ, ERROR       ;let's see what went wrong.
                LD      (FNUM) ,A       ;save the file number for later.
```

;.....more code

```
                LD      A, (FNUM)       ;get file number.
                LD      BC,20000        ;we have room for 20000 bytes.
                LD      HL, BUFFER      ;starting at this address.
                CALL    FCD2    ;read them in.


                JR      Z , NOTALL      ;all read in so file bigger than 20000
                CP      10
```

```
                JR      NZ, ERROR       ;this is not an EOF error, what
happened?
                LD      (FSIZE) ,BC     ;save this as the file size.
```

.

WRITE FILE

```
        JUMP TABLE ADDRESS:    FCD5



ENTRY:  A       file number
        BC      bytes to write
        HL.     pointer to the data

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine is very similar to READ on the previous page except that it writes out the specified number of bytes. It fills up the current block, writes it out if full via WRITE BLOCK(79) If the mode is 3 (read/write) it pre-reads the next block prior to writing additional data. Error handling is similar to READ.

The advantage of this routine is that it lets you write sequentially or randomly to a file. When using random write, you must use FILE MODE 3 when opening the file and pre-read up to the end of the previous record prior to writing out the current one.

EXAMPLES:

This routine opens a file for random read and updates record number 10.

```
                LD      HL, FILE        ;point to a file name.
                LD      A, 8            ;use tape 1.
                LD      B,3             ;let's read/write the file.
                CALL    FCC0            ;open the file.
                JR      NZ , ERROR      ;let's see what went wrong.
                LD      (FNUM) ,A       ;save the file number for later.
```

;more code

```
                LD      E,9;read 9 records.
```

MORE:

```
                LD      A,( FNUM)       ;get file number.
                LD      BC,20           ;each record is 20 bytes.
                LD      HL,BUFFER       ;starting at this address.
                CALL    FCD2            ;read them in.
                JR      NZ, ERROR       ;let's see what happened.
```

```
DEC     E
JR      NZ,MORE
LD      A, (FNUM)
LD      BC,20
LD      HL,NEW          ;this is the new data.
CALL    FCD5            ;update record 10.
JR      NZ, ERROR       ;something went wrong.
```

FILE OPERATIONS FILE I/O

TRIM FILE

JUMP TABLE ADDRESS: FCED

```
ENTRY:          A       device
        DE      pointer to filename


EXIT:           Zero flag set means no errors
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine is used to correctly close a file which was MADE [MAKE FILE](49) with a file length of zero. Initially, all remaining space was allocated to the file. The routine starts by finding the file and placing it in its internal FCB buffer. It compares the reserved size with the used size. If they are the same, the routine exits with no error. It then checks that the file is indeed the last entry in the directory. If it is not, the routine exits without an error; it just does not trim. If all the above conditions are met, the directory entry for the file and for BLOCKS LEFT are adjusted to reflect the USED file size. The directory is updated via a call to WRITE BLOCK(79). The only errors this routine generates are those outside its control such as File Not Found, Missing Media, Bad Block, etc.

Failure to TRIM a file which was made with a file length of 0 will effectively disable the rest of the medium. Since the BLOCKS LEFT will be zero, no more files will be allowed.

EXAMPLES:

FILE OPERATIONS FILE I/O

INITIALIZE DIRECTORY

```
JUMP TABLE ADDRESS:    FCBD



ENTRY:  A       device
        C       number of K in directory
        DE      size of medium (160K, 256K, 320K, etc.)
        HL      Volume name

EXIT:                   Zero flag set means no error
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine is equipped to correctly initialize a directory of any size on a medium of any size. It starts by filling a 1K buffer with zeros. It then moves in the 4 default directory entries: VOLUME, BOOT, DIRECTORY, and BLOCKS LEFT. It then copies the user supplied name into the volume entry, making sure that the volume name is not longer than 11 characters. If it is longer, it is truncated without an error. It then sets the directory size in the volume entry by adding 8OHEX to it and placing it in the attributes byte of the volume name. The volume size is then copied to the volume entry and the directory size to the directory. The number of blocks in the directory are subtracted from the volume size to determine the blocks left and that entry is updated. It then writes out the directory with an indirect call to

WRITE ONE BLOCK(80).

If this routine is carefully used, it can successfully initialize any directory. Unlike SmartBasic's INIT command, this routine does not check if the directory is protected. The only protection against an INIT from this function is a write protect tab.

Also, unlike SmartBasic's INIT, it does NOT put a JUMP TO SMARTWRITER in block zero of the medium. If the INIT disk was previously a bootable medium, you may get undesirable results if the disk is BOOTED.

EXAMPLES:

```
        LD      A,4             ;do disk one.
        LD      C,3             ;give it 3 blocks of directory.
        LD      DE,360          ;this is a double sided 5 1/4 disk.
        LD      HL, NAME        ;give it this volume name.
        CALL    FCBD            ;do it
        JR      NZ , ERROR      ;what happened?
```

;

```
NAME:   DB      'Guy's Disk',03
```

RESET FILE

```
JUMP TABLE ADDRESS:    FCC6

ENTRY:  A       file number

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This function rewinds a file back to the first byte so it can be read over from the beginning. It begins by checking the validity of the file number and checks if the FCB is indeed in use. Upon failure of either of these conditions, error code 9 is returned. It then checks if the current block has been modified. If so, it is written out via WRITE BLOCK(79). The FCB pointers are reset to the beginning of the file. If the mode is WRITE, the USED FILE SIZE is set to 1. If the mode is READ or READ/WRITE, the first block of the file is pre-read into the FCB buffer.

This function may be handy when TWO PASS work is performed on a file. After reaching the end or a determined point, the file can be rewound without resorting to CLOSE FILE(53) and OPEN FILE(52). While it can be used in conjunction with random access files, it is awkward for this use since the file is always rewound to the start rather than a specific point. One way to handle Random access is to rewind the file, read and ignore x-l records, and finally bring in record x. When a random file is very long, this process can be very time consuming, especially from tape.

The recommended approach for randomly accessing files is to calculate your own offsets and read it directly using READ BLOCK(74) WRITE BLOCK(79). Your program will have to remember what the highest record is and update the directory entry accordingly. Care must be exercised not to over run the maximum file size as allocated in the directory. The safest thing to do is likely to adopt the SmartFiler technique and allow only one data base per medium and have it occupy all the space regardless of the size.

EXAMPLES:

Consult the example in WRITE FILE(56). After the file has been opened and written to. It will be necessary to rewind it to access a smaller record number (e.g. 5):

```
        LD      A,(FNUM)        ;get the file number.
        CALL    FCC6            ;rewind to start.
        JR      NZ,ERROR
```

;

;now use the same technique shown on page 41

;to pre read records which are ahead of the desired one.

;

FILE OPERATIONS FILE I/O

GET DATE

JUMP TABLE ADDRESS:FCDB

```
ENTRY:  none


EXIT:                   Zero flag set means success
        A       error code if non zero
        B       day in Binary Coded Decimal
                        C       month
                D       year
                others  preserved
```

DESCRIPTION:

This routine extracts the system date from the EOS. If all 3 date bytes are zero, an error code 4 is returned in register A. Otherwise, registers B C D contain the day, month, and year.

The date is stored in BCD format. This is a combination of hexadecimal and decimal and can more accurately defined as the ASCII representation of the decimal characters of the date. Thus January 28, 1987 would be represented by the following HEX numbers: 28 0l 87.

EXAMPLES:

This subroutine gets the date and displays it in decimal on the screen. There are more elegant ways of printing BCD numbers, but this one is easy to follow:

SHOWDATE:

```
        CALL    FCDB
        LD      E,32            ;the ASCII value of space.
        LD      A,B
        CALL    PRINTD          ;print the two digits.
        LD      A, C
        CALL    PRINTD
        LD      A,D             ;and fall through to FRINTD.
```

PRINTD:

```
        PUSH    AF              ;save character.
        AND     F0              ;strip the lower bits.
        RRCA
        RRCA
        RRCA
        RRCA                    ;move to lower 4 bits.
        OR      E               ;add in the ASCII offset.
```

```
        CALL    PRINTA          ;routine to call print character and check
errors.
        POP     AF
        AND     0F              ;strip the top bits.
        OR      E
        CALL    PRINTA          ;print the other half.
        LD      A,' '           ;put a space between.
        CALL    PRINTA
        RET
```

PUT DATE

```
JUMP TABLE ADDRESS:    FCD8


ENTRY:          B       day
        C       month
        D       year
```

EXIT: all registers preserved

DESCRIPTION:

This routine updates the system date with the user supplied date. It will not return an error since no device operations or validity checks are performed. See GET DATE on previous page for date formats.

The only time that the EOS uses the system date is when a file is CREATED [MAKE FILE](49). At that time, the system date is written to the file control block of the file. Subsequent actions on a file will not change the system date. Even a call to UPDATE FILE IN DIRECTORY(50) will not change the date. Thus the date in a file entry is intended to reflect the creation date of the file.

If you wish to have your files DATED correctly, you should ask the user for the date, read it from a clock, etc. and then PUT in in the EOS via this function.

EXAMPLES:

Following is a machine language routine which may be used from SmartBasic to PUT the date in the EOS. It presumes that memory addresses 28000 to 28002 contain the day, month, and year in the correct BCD format.

```
        LD      HL, 28000
        LD      B,(HL)
        INC     HL
        LD      C, (HL)
        INC     HL
        LD      D, (HL)
        CALL    FCD8
        RET
```

This routine can be POKED anywhere in available memory; following are the related DECIMAL POKE values.

33 96 109 70 35 78 35 86 205 216 252 201

^^^^^

This is 28000; change to reflect a different source address for the data.

DELETE FILE

```
JUMP TABLE ADDRESS:    FCE1



ENTRY:  A       device
        HL      pointer to file name

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine calls FIND FILE(43) to see if the file exists. It then extracts the attributes from the file entry to see if the file is locked (bit 7). If it is, error code 16 is returned. If not, the deleted bit (2) is set and the directory entry re written via UPDATE FILE IN DIRECTORY(50).

The EOS uses its own internal buffers to perform this function. Thus two files can be opened and delete file called without fear of running out of buffers.

EXAMPLES:

```
                LD      A, 8            ;use tape 1.
                LD      HL, NAME        ;use this file name.
                CALL    FCE1            ;try and delete.
                RET     Z               ;it worked.
                CP      16
                JR      Z,LOCKED
        CP      5
        JR      Z,NOFILE                ;file not found.
        CP      24
        JR      Z,BADDIR                ;medium contained invalid directory.
```

;etc.

FILE OPERATIONS FILE I/O

RENAME FILE

```
JUMP TABLE ADDRESS:    FCDE

        ENTRY:          A       device
        DE      old name
        HL      new name

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine is used to change a file's name; it will not change the file's attributes. It begins by checking if the NEW NAME exists via FIND FILE(43). If it exists, the routine exits without generating an error!(l)! It then tries to find the old file name. If it is not found, error 5 is returned. The next step is to move 12 bytes of the new name on top of the old name and rewrite the directory via UPDATE FILE IN DIRECTORY(50).

(l)This unfortunate oversight can cause many problems. For example, you may try to rename an "A" file to a" and get a no error condition when nothing was done. It seems this routine initially returned an error but COLECO decided not to generate an error because they expected the FIND FILE routine to generate errors. While this is true with the second find, it is not of the first. PROGRAMMERS BEWARE! You must look for the target file name yourself.

EXAMPLES:

Here is a routine to correctly rename a file.

```
        LD      A,4             ;use disk 1.
        LD      DE,NEW          ;point to the new file name.
        LD      HL,BUFFER       ;store the entry here.
        CALL    FCCC            ;find the file.
        JR      Z,EXISTS        ;then rename must abort.
        LD.     A,4
        LD      DE,OLD
        LD      HL,NEW
        CALL    FCDE            ;now we can try to rename.
        RET     Z               ;yes we got it.
```

;

;process errors here

DEVICE OPERATIONS

FIND PCB

```
JUMP TABLE ADDRESS:    FC5A

ENTRY:  none

EXIT:   IY             current PCB address
        others  preserved
```

DESCRIPTION:

This simple routine just returns the address of the current Processor Control Block. It is the header to the Device Control Blocks which follow immediately after it. The first byte of the PCB (IY+0) indicates its status. The lower 7 bits are usually a 2, indicating it is busy... it is always busy. The top bit is set to indicate it has acknowledged the last command it has received. The next 2 bytes contain a pointer to the PCB itself. While this may seem redundant, it is essential to the proper operation of a RELOCATE PCB(17) command.

Byte 3 (IY+3) may be of interest. It indicates how many devices were FOUND when the net was scanned. Remember that the keyboard and ADAM printer are one device each. The tape drives 1 and 2 only count as 1 device since they share the same processor. Thus, a BASE ADAM will respond with three devices. An ADAM with two disk drives will have 5 devices. A maximum of 15 devices can be handled by the PCB.

EXAMPLES:

DEVICE OPERATIONS

FIND DCB

```
JUMP TABLE ADDRESS:    FC54 or FC57

ENTRY:  A       device
```

```
EXIT:           Zero flag set means no error
                A       device ID or error code
        IY      pointer to DCB for selected device or garbage if error
                others  preserved
```

DESCRIPTION:

This routine is called internally by various device routines. It returns a pointer to the Device Control Block in register IY. If the user wishes to write his own device handling routines, this one will be the backbone of all operations Unless the NET is re-scanned, it is not necessary to FIND the DCB before every operation. EOS device routines can be speeded up by making your own table of DEVICES and accessing them directly. The EOS originally had two different routines for finding DCB's. While one has been eliminated, its jump table entry points to the active routine.

See DEVICE CONTROL BLOCK STRUCTURE(119) for details of the DCB commands.

EXAMPLES:

The following routine sets up a table for 15 devices and writes their DCB address (or zero) in the table:

```
        LD      B,15
        LD      HL,TABLE+31             ;point to the top of the device table.
```

FIND:

```
        LD      A,B
        CALL    FC54
        LD      DE,0    ;prepare to write a zero if none.
        JR      NZ,NODCB
        PUSH    IY
        POP     DE      copy DCB address to DE.
NODCB:
        LD      (HL) ,D ;write address to table.
        DEC     HL
        LD      (HL) ,E ;remember we are working backwards.
        DEC     HL
        DJNZ    FIND
```

DEVICE OPERATIONS

REQUEST DEVICE STATUS

```
JUMP TABLE ADDRESS:    FC7E
ETTRY:  A      device

EXIT:   Zero flag set means no errors
        A      error code if non-zero
               IY      DCB address or garbage if DCB not found.
        others  preserved
```

DESCRIPTION:

This routine finds the DCB(67) for the requested device. It asks the device for status and waits for a reply. When the DCB replies, the status is compared with 80HEX and control is returned to the user with IY pointing to the DCB.

This routine returns the status of the DCB itself. For the status of the device, see the routine on the next page.

EXAMPLES:

```
        LD      A,4             ;check disk 1.
        CALL    FC7E            ;how is it doing?
        RET     Z               ;everything is fine.
        AND     7F              ;strip the high bit.
        CP      1
        JR      Z,NODCB         ;sorry no disk one.
        CP      2
        JR      Z,BUSY          ;leave me alone I am working.
        CP      3
        JR      Z,IDLE          ;device off line.
```

;etc.

DEVICE OPERATIONS

GET DEVICE STATUS

```
JUMP TABLE ADDRESS:    FCE4

ENTRY:  A       device

EXIT:                  Zero flag set means no errors
        A       error code if non-zero or status
        others  preserved
```

DESCRIPTION:

This routine gets the device specific status which was reported during the REQUEST FOR STATUS on the previous page. In order to get a complete status report, both these routines should be called and evaluated.

The routine does not perform a STATUS request, it just reads the DEVICE STATUS FLAG from the device control block. This is a complex bit pattern indicating presence of an alternate device and whether or not media is present. These bit patterns are placed in the upper and lower nibble of the status which is returned in A. If you are checking a PRIMARY device (one with a number from I to 15), you should check the lower nibble. If you are checking a secondary device (one with a number from 17 to 31), you should check the upper nibble.

```
The bit pattern is as follows: 0000    unknown error/or all OK
                       0001    CRC Error
                       0010    Missing Block
                       0011    Missing Media
                       0100    Missing Drive
                       0101    Write protect tab in place
                       0111    Drive Error
```

EXAMPLES:

The following routine asks for the device status flag. the device status of the tape drives and reads

```
               LD      A,8             ;ask about tape 1.
               CALL    FC7E            ;get status first.
               JR      NZ, ERROR       ;problem with device.
               LD      A, 8
               CALL    FCE4            ;ask for device status.
               AND     0F              ;use lower nibble only.
               CP      4
               JR      Z , NODEVICE    ;there is no tape 1.
               CP      3
               JR      Z,NOMEDIA       ;no tape in the drive.
               LD      A,18H
               CALL    FCE4            ;ask about tape 2.
               AND     F0              ;strip low bits this time.
               RRCA
               RRCA
```

```
                RRCA
                RRCA                            ;shift high to low.
```

;now process as above for tape 2 status.

DEVICE OPERATIONS

SOFT RESET DEVICE

```
JUMP TABLE ADDRESS:    FC90

ENTRY:  A       device

EXIT:                  Zero flag set means success
        A       error code if non zero or 128 if complete
        others          preserved
```

DESCRIPTION:

This routine can be used to reset a device to its default inactive state. While there are routines to specifically reset the keyboard and printer, they both call this routine. Other devices like the tape and disk drives, must be reset from this routine.

The routine begins by finding the DCB for the device; it returns an error if not found. It then checks if the device is busy; a busy device is not reset as this may interrupt a crucial activity like writing a block to disk. The next step is to ask the DCB to RESET the device. The routine then waits for the DCB to acknowledge the command and compares the result with 128 which is the OK status. Thus, if there are any problems, a NON ZERO flag is returned to the user for interpretation.

EXAMPLES:

```
        LD      A, 8            ;reset tape 1.
        CALL    RESET           ;do it.
```

;

;more code

;subroutine to reset the device in A

;

RESET:

```
        CALL    FC90
        RET     Z               ;reset went OK
        AND     127             ;strip acknowledge bit.
        CP      3
        JR      Z,BUSY          ;device is doing something
        CP      1
        JR      NZ,NODCB        ;sorry I don't have a device.
```

DEVICE OPERATIONS

SOFT RESET KEYBOARD

```
JUMP TABLE ADDRESS:    FC93



ENTRY:          none

EXIT:                   Zero flag set if success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine simply loads the keyboard device(l) and jumps to SOFT RESET DEVICE(71). It has one important function...take off the SHIFT LOCK key. Sometimes, your program may expect a lower-case character or a number from the keyboard. If the KEY PRESS is not echoed to the screen, you may receive a $ instead of a 4, reject it, and the user is wondering what happened. Sending a SOFT RESET to the keyboard, makes sure that the LOCK key is released.

EXAMPLES:

DEVICE OPERATIONS

SOFT RESET PRINTER

```
JUMP TABLE ADDRESS:    FC96

ENTRY:          none

EXIT:                   Zero flag set means success
        A       error code if non-zero
        others  preserved
```

DESCRIPTION:

This routine also loads the PRINTER device number(2) and jumps to SOFT RESET DEVICE(71). As with the keyboard reset, it accomplishes a useful function. When the printer receives a reset, it returns the print head to the left margin and returns the printer to FORWARD MOTION. This function is highly recommended before any printing job. If the last job was aborted in mid stream, you want to make sure you are not left hanging in the middle of a line.

Note that the printer will not be reset if it is busy (printing something). You should make sure it is not busy by performing a status check first.

EXAMPLES:

DEVICE OPERATIONS READ/WRITE BLOCK

READ BLOCK

```
JUMP TABLE ADDRESS:    FCF3

ENTRY:  A       device
        BCDE    block number
        HL      destination for data

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This is the most intensive READ BLOCK routine. It begins by calling READ ONE BLOCK(75). If an error other than TIMEOUT is returned, it retries up to twice to complete the request. if the command is not accepted, then a media error is returned (22). If the command is accepted or a timeout, it CALLS REQUEST DEVICE STATUS(68) until the time out clears or an error is returned. It then gets the MEDIA status (upper or lower nibble) to check for errors. If the media or the device is missing, then error code 22 is returned. If there is no media error, a second READ request is sent. After two retries, the command aborts.

While this routine catches a variety of errors, it almost always returns the same error code 22. It may be difficult for the program to interpret the errors generated by the routine. If you just want to tell the user there is an error, then use this routine since it should catch all the possibilities. If you want to decode the errors yourself, use the one on the next page.

Note that the retries in this routine do little good since a tape or disk error is not cleared by the controller until the drive door is opened and the media "adjusted".

Note also that the second read request does not really slow down the operation since the tape and disk drives have a 1K memory buffer and can instantly return the data if the block number is the same.

COLECO had anticipated VERY LARGE media when it designed the READ BLOCK routines. Accordingly, you must send a quadruple precision number as the BLOCK to read. Register BC contains the HIGH BLOCK NUMBER (usually zero), and register DE, the LOW BLOCK NUMBER (up to 65535K). When we have devices on the ADAM NET which have more than 64MEG of storage, then we will start using BC.

EXAMPLES:

```
                LD      A, 4            ;read disk one.
                LD      BC,0
                LD      DE,100          ;read block 100.
                LD      HL, 12345       ;put the data here.
                CALL    FCF3            ;try and read.
                JR      NZ, ERROR       ;guess we got an error code 22.
```

DEVICE OPERATIONS READ/WRITE BLOCK

READ ONE BLOCK

```
JUMP TABLE ADDRESS:    FC69

ENTRY:  A       device
        BCDE    block number
        HL      RAM address to put data

EXIT:                   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine is quite simple compared to READ BLOCK on the previous page. It calls START READ ONE BLOCK(76) and aborts on error. It then CALLS END READ ONE BLOCK(77) until the request is completed or an error returned.

This routine will return an error if the DCB is not found or if the device is busy doing something else. It will also return an error if the device is off line (idle) or if the device itself returns an error. While not as elegant as READ BLOCK, it gives the programmer the opportunity to check for other errors.

EXAMPLES:

This routine tries to read one block. If the returned error is not BUSY or NO DCB, the programmer checks if media is missing. Then the appropriate action can be taken.

```
        LD      A,4                     ;read disk one.
        LD      BC,0
        LD      DE,100                  ;read block 100.
        LD      HL,12345                ;put the data here.
        CALL    FC69                    ;try and read.
        JR      Z,GOOD
        AND     127                     ;strip the acknowledge bit if any.
        CP      1
        JR      Z,NODCB                 ;oops.
        CP      2
        JR      Z,BUSY                  ;device is busy on something else.
        CP      3
        JR      Z,IDLE                  ;device is off line.
        LD      A,4
        CALL    FCE4                    ;ask for device status.
        AND     0F                      ;use lower nibble only.
        CP      4
        JR      Z,NODEVICE              ;there is no disk 1.
        OR      A
        JR      NZ,NOMEDIA              ;no disk in the drive.
```

DEVICE OPERATIONS READ/WRITE BLOCK

START READ ONE BLOCK

```
JUMP TABLE ADDRESS:    FCA2
ENTRY:  A       device
        BCDE    block number
        HL      RAM address to put data


EXIT:   Zero flag set means success
        A       error code if non zero
        others  preserved (including A if no error)
```

DESCRIPTION:

This routine starts by calling FIND DCB(67) and aborts if there is an error. It then checks if the device is off line (idle). It then writes the pertinent data into the DCB and issues a READ request. Control is then returned to the user so the device (tape or disk) can do its work while other activities are in progress.

Have you ever noticed that the SUPER GAME Buck Rogers goes to the disk or data pack WHILE THE GAME IS IN PROGRESS? It does that using START READ and END READ on the next page. The reason for this is to load the NEXT SCREEN so it is instantly ready for use after finishing this screen. Unless you are extremely good, you never have to wait after finishing a screen. This process known as background loading can be quite effective when properly used.

The effective transfer rate of the DATA drives is 1400 bytes per second. Thus, it takes close to a second to READ a block after the correct place has been found. During that period of time, hundreds of operations may be performed by the Z-80 if there is no need to WAIT for the data.

EXAMPLES:

See END read for a complete example.

DEVICE OPERATIONS READ/WRITE BLOCK

END READ ONE BLOCK

```
JUMP TABLE ADDRESS:    FC45

ENTRY:          A       device

EXIT:           Carry flag set means complete
                Zero flag set means no error if CARRY
                A       error code if non zero
                others  preserved
```

DESCRIPTION:

This routine starts by finding the DCB(67) and returns an error if not found. It then checks if the device is off line (idle). It then checks if the DCB has acknowledged the command (start read). If not, the CARRY FLAG is reset (no carry) and control returned to the caller. If the command has been acknowledged, the status is compared to 8OHEX, the carry flag set, and control returned to the caller.

It is important to check the CARRY flag first. If it is not set, then don't presume there is an error. If the Carry is set and a NON zero condition exists, then there was an error. In order to decode the error, start by subtracting 80HEX and comparing to the DCB error code table (92).

EXAMPLES:

These excerpts show a potential game situation which uses background loading of ONE block from disk. It calls several subroutines which are not detailed but can be used as a general guide for such uses:

```
        LD      A,4             ;use disk 1.
        LD      BC,0
        LD      DE,17           ;get from block 17.
        LD      HL, 12345       ;put it here.
        CALL    FCA2            ;start read.
        JP      NZ, ERROR       ;oops.
```

WAIT:

```
        CALL    READJOY         ;read the joystick.
        CALL    MOVE            ;move player if requested.
        CALL    FIRE            ;execute fire button if any.
        CALL    UPDATE          ;move monsters, obstacles, etc.
        LD      A,4
        CALL    FC45            ;are we done?
        JR      NC ,WAIT        ;command not acknowledged yet.
        JP      Z,DONE          ;background loading completed.
        AND     127             ;strip acknowledge bit.
        CP      1
        JR      Z,NODCB
```

```
        CP      2
        JR      Z,BUSY
        CP      3
        JR      Z,IDLE
```

DEVICE OPERATIONS READ/WRITE BLOCK

WRITE BLOCK

```
JUMP TABLE ADDRESS:    FCF6

ENTRY:  A       device
        BCDE    block number
        HL      where to write data

EXIT:   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This is the intensive WRITE BLOCK routine. It begins by calling WRITE ONE BLOCK(80). If an error other than TIMEOUT is returned, it retries up to twice to complete the request. if the command is not accepted, then a media error is returned (22). If the command is accepted or a timeout, it CALLS REQUEST DEVICE STATUS(68) until the time out clears or an error is returned. It then gets the MEDIA status (upper or lower nibble) to check for errors. If the media or the device is missing, - then error code 22 is returned.

While this routine catches a variety of errors, it almost always returns the same error code 22. It may be difficult for the program to interpret the errors generated by the routine. If you just want to tell the user there is an error, then use this routine since it should catch all the possibilities. If you want to decode the errors yourself, use the one on the next page.

Note that the retries in this routine do little good since a tape or disk error is not cleared by the controller until the drive door is opened and the media "adjusted".

Contrary to READ BLOCK(56), this routine does not ask for a second write as it would require rewinding the tape and writing again. This operation would be quite slow.

EXAMPLES:

```
        LD      A, 24           ;write to tape 2.
        LD      BC,0
        LD      DE,100          ;write block 100.
        LD      HL, 12345       ;put the data found here.
        CALL    FCF3            ;try and write.
        JR      NZ, ERROR       ;guess we got an error code 22.
```

DEVICE OPERATIONS READ/WRITE BLOCK

WRITE ONE BLOCK

```
JUMP TABLE ADDRESS:    FCB4

ENTRY:  A       device
        BCDE    block number
        HL      memory address to start writing from

EXIT:   Zero flag set means success
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine calls START WRITE ONE BLOCK(81) and returns if there is an error. It then repeatedly calls END WRITE ONE BLOCK(82) until such time as the command is completed. When the acknowledge is received, the end write error, if any, is returned to the caller. The routine stays in total control until the process is completed or an error generated.

EXAMPLES:

This routine tries to write one block. If the returned error is not BUSY or NO DCB, the programmer checks if media is missing. Then the appropriate action can be taken.

```
        LD      A,4             ;write to disk one.
        LD      BC,0
        LD      DE,100          ;write block 100.
        LD      HL,12345                ;put the data found here.
        CALL    FG69            ;try and write.
        JR      Z,GOOD
        AND     127             ;strip the acknowledge bit if any.
        CP      1
        JR      Z,NODCB         ;oops.
        CP      2
        JR      Z,BUSY          ;device is busy on something else.
        CP      3
        JR      Z,IDLE          ;device is off line.
        LD      A,4
        CALL    FCE4            ;ask for device status.
        AND     0F              ;use lower nibble only.
        CF      4
        JR      Z,NODEVICE      ;there is no disk 1.
        OR      A
        JR      NZ,NOMEDIA      ;no disk in the drive.
```

DEVICE OPERATIONS READ/WRITE BLOCK

START WRITE ONE BLOCK

```
JUMP TABLE ADDRESS:    FCAB
ENTRY:  A       device
        BCDE    block number
        HL      RAM address to start writing from


EXIT:   Zero flag set means success
        A       error code if non zero
        others  preserved (including A if no error).
```

DESCRIPTION:

This routine starts by calling FIND DCB(67) and aborts if there is an error. It then checks if the device is off line (idle). It then writes the pertinent data into the DCB and issues a WRITE request. Control is then returned to the user so the device (tape or disk) can do its work while other activities are in progress.

Except for the WRITE request, this routine is identical to START READ ONE BLOCK(76).

EXAMPLES:

DEVICE OPERATIONS READ/WRITE BLOCK

END WRITE ONE BLOCK

```
JUMP TABLE ADDRESS:    FC4E

ENTRY:          A       device

EXIT:   Carry flag set means complete
        Zero flag set means no error if CARRY
        A       error code if non zero
        others  preserved
```

DESCRIPTION:

This routine starts by finding the DCB(67) and returns an error if not found. It then checks if the device is off line (idle). It then checks if the DCB has acknowledged the command (start write). If not, the CARRY FLAG is reset (no carry) and control returned to the caller. If the command has been acknowledged, the status is compared to 8OHEX, the carry flag set, and control returned to the caller.

This routine is identical to END READ ONE BLOCK(77). See that routine for more details. As a matter of fact, either END READ or END WRITE may be called to determine if a device is done its operation whether read or write.

EXAMPLES:

VIDEO RAM MANAGEMENT

SET VDP PORTS

```
JUMP TABLE ADDRESS:    FD11

ENTRY:  none

EXIT:   A       current memory configuration
        BC      destroyed
        DE      preserved
        HL      destroyed
```

DESCRIPTION:

This routine starts by remembering what the current memory configuration is. It then calls bank switch [SWITCH MEMORY BANKS](20) to switch in the 0S7. This is the mini operating system used by SmartWriter. It extracts the port addresses for the VDP, Games Controllers, Strobe and Sound ports. Finally, it restores the default memory configuration.

As a byproduct, this routine returns the current memory configuration. This may be handy to make sure the settings are correct for the desired operation.

While this routine is called by the EOS cold start [INITIALIZE EOS](11), there may be situations where the programmer may need to refresh the EOS data areas where the default control ports are stored.

While it is not essential to know what the port numbers are, following are the memory locations and port values for EOS 6. The addresses may be different in other revisions of EOS but the port values should be the same.

PORT	ADDRESS	VALUE	
VDP Control	FC29	191	
VDP Data	FC2A	190	
Joystick 0	FC2B	252	
Joystick 1	FC2C	255	IN ONLY
Strobe set	FC2D	128	
Strobe reset	FC2E	192	
Sound	FC2F	255	OUT ONLY

EXAMPLES:

VIDEO RAM MANAGEMENT

INITIALIZE VRAM TABLES

JUMP TABLE ADDRESS: FD29

```
ENTRY:          A               table number
        HL      table address

EXIT:                   all registers, including IX and IY are used
```

DESCRIPTION:

This is a very useful routine which can be used to set up any of the 5 VDP tables. While the routine correctly sets up the pointers to the tables, it does not check if the tables overlap each other. It is up to the programmer to ensure correct setting of the VDP tables. Once the tables are set up, subsequent EOS calls involving characters, patterns or sprites will be sent to the VDP according to the tables defined here.

The EOS cold start routine [INITIALIZE EOS](11) does NOT set up the VDP; you must set up your own tables. It is essential to set up EACH table which will be used. In order to facilitate the calculation of tables, the routine asks for a value in HL which is the VDP address where the table should start. The routine takes care of the encoding of the data.

```
        TABLE NUMBER    NAME                    SIZE
        0               Sprite Attribute        128 bytes
        1               Sprite Pattern          256 bytes
        2               Pattern name table      varies with VDP mode
        3               Pattern Generator       2048 bytes
        4               Pattern Color           varies with VDP mode
```

EXAMPLES:

This routine shows the requirements for 32 column displays. VDP memory area from 420 Hex to 7FF Hex is unused. Neither is the area form 1000 Hex to 37FF Hex. All area above 3980 Hex is also available. This configuration leaves plenty of room for bigger pattern and color tables for the other VDP modes.

VIDEO RAM MANAGEMENT

LOAD DEFAULT ASCII IN VDP

```
JUMP TABLE ADDRESS:    FD38

ENTRY:          A      none

EXIT:                  All registers including IX are used.
```

DESCRIPTION:

This routine gets the address of the pattern generator table as it was set up using INIT TABLES on the previous page. It then asks PUT ASCII on the next page, to place the first 128 characters in the start of the table. This routine is a handy way of bringing in all the ASCII characters without calculating the offsets.

The user MUST call this routine before printing any characters to the screen unless the screen was already initialized by another program.

EXAMPLES:

VIDEO RAM MANAGEMENT

put ASCII IN VDP

JUMP TABLE ADDRESS: FD17

```
ENTRY:  BC      number of characters (not bytes)
        DE      address in Video RAM to start at
        HL      character number to start at
```

EXIT: all registers used except IY

DESCRIPTION:

This routine copies a specific number of ASCII characters into VRAM. It begins by multiplying HL by 8 to get the offset position in the table. It also multiplies BC by 8 to get the total byte count. It then switches in the EOS ROM where the default ASCII characters are located. It gets the address of the ASCII character set and adds the offset to the first character. It then calls WRITE VRAM(88) to move the data. The last step is to restore the memory configuration to its previous state. When calling this routine, it is essential to indicate the proper offset into video RAM based on the tables which were set up using the routine on page 65.

EXAMPLES:

This subroutine restores one ASCII character (sent in A) to its default state. It gets the base VAAM address from memory where the programmer stored it when initializing the VDP. It might be used if a character was intentionally (or accidentally) altered for a special effect.

```
        LD      BC,1            ;restore only one.
        LD      DE, (PATTERN)   ;base address of pattern table.
        LD      H,0
        LD      L,A             ;the pattern to restore.
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL           ;times 8-offset.
        ADD     HL,DE
        EX      DE,HL           ;here's where to start in table.
        LD      H,0
        LD      L,A             ;put pattern number back in HL.
        CALL    FDl7            ;do it.
```

If you are going to make significant use of this routine, it is wise to set your pattern table at 0. In that case you only need to multiply the character by 8 to determine the offset:

```
        LD      BC,1
        LD      H,0
        LD      L,A
        ADD     HL,HL
        ADD     HL,HL
        ADD     HL,HL
        EX      DE,HL
```

```
        LD      H,0
        LD      L,A
        CALL    FDl7    ;much shorter.
```

VIDEO RAM MANAGEMENT

WRITE VRAM

```
JUMP TABLE ADDRESS:    FD1A

ENTRY:  BC      number of bytes to write
        DE      VRAM start address
        HL      where to get data (user buffer)

EXIT:   HL      points one past end of data moved
        others  destroyed
```

DESCRIPTION:

The routine starts by preparing the VDP for a write operation by feeding it the start address in DE. It then sends the bytes one at a time through the VDP data port. Since the port does not return any errors, the routine just presumes all went well and exits after sending the last byte.

EXAMPLES:

This routine sends a screen full of text to the name table. This can be a quick way to POP information into the screen (e.g. a window).

```
        LD      BC ,768         ;get a screen full worth of 32 column data.
        LD      DE, (NAME)      ;where did I put the name table?
        LD      HL, SCREEN1     ;this is the screen I want displayed.
        CALL    FD1A
```

READ VRAM

```
JUMP TABLE ADDRESS:    FD1D

ENTRY:  BC      number of bytes to read
        DE      VRAM start address
        HL      where to put data (user buffer)

EXIT:   HL      points one past end of data moved
        others  destroyed
```

DESCRIPTION:

This routine is the logical complement to WRITE VRAM on the previous page. It collects the specified number of bytes from the VDP and places them in the where they can be analyzed, modified, and perhaps sent back user's RAM area to the VDP.

EXAMPLES:

VIDEO RAM MANAGEMENT

PUT VRAM

```
JUMP TABLE ADDRESS:    FD2C

ENTRY:  A       table number
        DE      first entry to update
        HL      pointer to user buffer
        IY      number of entries to move

EXIT:   HL      points to one past end of data moved
        IX      preserved
        others  destroyed
```

DESCRIPTION:

This routine is more user friendly than WRITE VRAM(88). It takes care of all the arithmetic for you. All you need to do is specify which table you wish to update and let the EOS take care of offsets, table addresses, etc.

The routine checks which table to update (see list on page 65) and multiplies the entry numbers, offsets, etc. by 4 or 8 depending if sprite attributes or pattern tables. It then extracts the base VRAM address from its tables and adds in the offset to the start of the table. The last step is to call WRITE VRAM to actually move the data.

EXAMPLES:

This routine updates the sprite attributes for sprite numbers 10 to 19

```
        LD      A,0             ;do sprite attributes.
        LD      DE,10           ;start at sprite 10.
        LD      HL, SP+10*4     ;start 40 bytes in the sprite table.
        LD      IY, 10          ;update 10 entries.
        CALL    FD2C
```

This routine sends one full line of text to line 10

```
        LD      A,2             ;use the name table.
        LD      DE,320          ;start at line 10 (32 per line).
        LD      HL, LINE        ;move in this line.
        LD      IY, 32          ;move 32 entries.
        CALL    FD2C
```

GET VRAM

```
JUMP TABLE ADDRESS:    FD2F

ENTRY:  A      table number
        DE     first entry to get
        HL     pointer to user buffer
        IY     number of entries to move

EXIT:   HL     points to one past end of data moved
        IX     preserved
        others destroyed
```

DESCRIPTION:

This routine is the complement of PUT VRAM on the previous page. It extracts from VRAM a specified number of entries from the desired table. See additional explanations on previous page.

EXAMPLES:

This routine gets a full line of text from line 10 and shifts all the characters right by one. It then writes the data back; the last character on the right disappears.

```
        LD      A,2     ;use name table.
        LD      DE,320  ;start at line 10.
        LD      HL,LINE ;move it to here.
        LD      IY, 32  ;move 32 entries.
        CALL    FD2F    ;get it here.


        LD      A,2     ;use the name table.
        LD      DE,320  ;start at line 10 (32 per line).
        LD      HL, LINE - 1    ;start one behind it.
        LD      (HL),' '        ;put a space there.
        LD      IY, 32  ;move 32 entries.
        CALL    FD2C    ;scroll the line over.
```

VIDEO RAM MANAGEMENT

WRITE VDP REGISTER

```
JUMP TABLE ADDRESS:    FD20

ENTRY:          B      Register to write to
                C      Data to send

EXIT:   ABCDE   destroyed
        others  preserved
```

DESCRIPTION:

This routine extracts the VDP control port address from the EOS table, sends the DATA out the port and then sends the register number to which 128 has been added. If VDP register 0 or 1 is being written to, the data is stored so it knows what mode it is in when calculating tables. Writing to the VDP registers is the way to set up various VDP modes. Following is a breakdown of the VDP values; only the pertinent bits are shown, the others are zero.

```
REGISTER        BIT     Meaning if set
0               1       Graphics mode 2


1               7       Always 1
                6       0 to blank display, 1 to activate
                5       Non Maskable Interrupt enable
                4       TEXT mode on (40 column) \ set only one of these bits
                3       MULTI COLOR mode on    / or none for 32 column
                1       16x16 sprite enable
                0       double size sprite enable


2               3210    multiplied by 400      HEX is address of NAME table
3                       multiplied by 40       HEX is address of COLOUR table
4                       multiplied by 800      HEX is address of PATTERN table
S                       multiplied by 80       HEX is address of SPRITE
Attributes
6                       multiplied by 800      HEX is address of SPRITE
Pattern


7               7654            TEXT color in TEXT mode (40 column)
                3210            BORDER color
```

Bit 6 of register 1 can be cleared (off) while a screen is being PAINTED. The screen will be blanked but drawing will still take place. Setting Bit 6 will instantly TURN ON the entire screen... nice special effect.

Bits 0 and 1 of register 1 are not the same. Bit 1 uses 4 consecutive sprites (e.g. 0,1,2,3 to make a composite sprite). Bit 0 doubles the size of each bit to make a double sized sprite. Both bits can be combined for 32x32.

Registers 2 to 6 are handled automatically by INITIALIZE VRAM TABLES (84).

The TEXT mode uses no color table. SET (on) bits are the color specified by register 7. CLEAR (off) bits are always transparent and allow the border color (register 7) to shine through and become the background. The same is true of the 32 column mode when the CLEAR bits are set to transparent.

VIDEO RAM MANAGEMENT

READ VDP REGISTER

```
JUMP TABLE ADDRESS:    FD23


ENTRY:  none

EXIT:   A       VDP control value

        C       Control port number

                others  preserved
```

DESCRIPTION:

This routine reads in whatever value resides in the VDP control port, stores it internally, and returns the control value in A to the caller.

The VDP CONTROL port is also referred to as register 7 which is READ ONLY. Note that the registers on the previous page are WRITE ONLY. If you need to remember what you wrote to the registers, store it yourself. Reading the CONTROL port provides 4 items of data in the bit pattern:

```
bit 7           set on each raster scan (interrupt)

bit 6           set if 5 sprites on the same line

bit 5           set if two sprites overlap

bits 43210      fifth sprite if any.
```

Reading the CONTROL PORT clears any collisions or 5th sprites which were recorded. If more than one condition requires verification, be sure sand save the output value as it cannot be retrieved again. The sprite overlap (collision) flag does not tell you which two have collided, but it is an easy way to check if there is a collision. If no collision, then you can skip ahead. If there is one, then check each pair to determine who has collided with whom.

Have you noticed on some arcade games that some of the objects (sprites) flicker at times? When this happens, have a close look: there are 5 or more sprites on the same line. BURGER TIME is a good example. In order for the sprite not to totally disappear, programmers remove one of the higher priority sprites allowing the fifth sprite to show. They then replace the other sprite making the 5th disappear again. It can be a lot of work but 'keeps the sprites visible.

EXAMPLES:

VIDEO RAM MANAGEMENT

FILL VRAM

```
JUMP TABLE ADDRESS:    FD26

ENTRY:  A       value to write
        DE      repeat count
        HL      address to write to

EXIT:   All registers destroyed except IX IY
```

DESCRIPTION:

This routine sets up the VDP for a write operation to address HL. It then sends the byte in A to VRAM DE times. When used to fill the name table, it puts the same character throughout the specified area.

EXAMPLES:

This routine sets up the name table and fills 24 lines of 32 rows with a space. This clears the screen the hard way.

```
        LD      HL,0    ;use start of VRAM.
        LD      A,2     ;for the name table.
        CALL    FD29    ;WRITE VDP register
        LD      A ,     ;put a space.
        LD      DE,24*32        ;in the whole screen.
        LD      HL,0    ;starting at beginning.
        CALL    FD26
```

VIDEO RAM MANAGEMENT

CALCULATE PATTERN POSITION

```
JUMP TABLE ADDRESS:    FD32

ENTRY:  D      Y position
        E      X position

EXIT:   DE     absolute offset
        others preserved
```

DESCRIPTION:

This routine takes the drudgery out of figuring the proper offset related to a given set of X Y coordinates. It simply multiplies Y by 32 and adds X. Unfortunately, the routine is set up for a 32 column screen configuration. If you are using a 40 column set up, you are on your own. Once the offset is calculated, you can use PUT VRAM(70) to write data directly to a screen position without affecting the current cursor position. If you wish to position the cursor, you should use the control sequences in CONSOLE DISPLAY SPECIAL(23).

The routine is equipped to handle negative x,y coordinates but I have no idea why you would want to use them.

EXAMPLES:

This routine writes a message to a particular location on the screen.

```
        LD      E,10    ;goto line 10.

        LD      D,5     ;column 5.

        CALL    FD32    ;get offset.

        LD      A,2     ;name table.

        EX      DE,HL   ;put first entry into DE.

        LD      HL,MESSAGE      ;print this word.

        LD      IY,11   ;it is 11 bytes long.

        CALL    FD2C    ;put in in VAAM.
```

.....

MESSAGE:

```
               DB      'Please Wait'
...........
```

For 40 columns, following is a routine to calculate 40*y+x in DE

```
        LD      H,0
        LD      L,D     ;HL-y.
        ADD     HL,HL   ;*2.
        ADD     HL,HL   *4
        ADD     HL,HL   ;*8.
        LD      B,H
        LD      C,L     save *8.
        ADD     HL,HL   ;*16.
        ADD     HL,HL   ;*32.
        ADD     HL,BC   ;*40.
        LD      D,0
        ADD     HL,DE   ;40*y+x.
```

VIDEO RAM MANAGEMENT

POINT TO PATTERN POSITION

```
JUMP TABLE ADDRESS:    FD35

ENTRY:  DE      signed number

EXIT:   E       -DE/8 or 127 or -128 (80 HEX)
        others  preserved
```

DESCRIPTION:

This routine is used to determine the pattern number for a given offset. Since each pattern is 8 bytes, dividing DE by 8 does the job. The routine can handle negative offset and returns either 7F or 80. Again, I have no idea what the negative offsets are used for.

EXAMPLES:

VIDEO RAM MANAGEMENT

WRITE SPRITE TABLE

```
JUMP TABLE ADDRESS:    FD3B

ENTRY:  A       number of sprites to write
        DE      Sprite attribute table
                HL      Sprite priority table

EXIT:   IX      preserved
        IY      preserved
        others  destroyed
```

DESCRIPTION:

This routine writes the specified number of sprites to the VDP in the order specified by the priority table in (HL). This routine is one way to reorder the sprites without having to MOVE all the entries in the table. It can be used to help overcome the 5th sprite priority problem.

EXAMPLES:

This routine uses two priority tables, alternating on each successive call to send the sprites in a different order.

```
        LD              HL,ONE          ;maybe use first order.
        LD              A, (TOGGLE)     ;
        NEG                             ;toggle it.
        LD              (TOGGLE) ,A     ;save it back.
        JR              Z,USE1          ;let's use the first table.
        LD              HL,TWO          ;let's use the alternate.
        USE1:
        LD              DE ,ATTRIBUTE   ;get the attributes table.
        LD              A, 6            ;it contains 6 sprites.
        CALL            FD3B            ;do it.
        RET
```

....

```
ONE:    DB      0,1,2,3,4,5
TWO:    DB      5,4,3,2,1,0
```

GAME CONTROLLERS

READ GAME CONTROLLER

```
JUMP TABLE ADDRESS:    FD3E

ENTRY:          A      controller/spinner value
                       bit 0 controller 0
                       bit 1 controller 1
                       bit 7 enable spinner for selected controller

                       e.g. use 131 decimal or 83 HEX to read all

                IX     user decode table


EXIT:   IY     preserved
        others destroyed
```

DESCRIPTION:

This routine can read either joystick or both and optionally read their spinners as well. It checks if controller 0 was requested and skips to controller 1 if not. If neither controller is requested, the routine returns immediately. After reading controller 0, it checks if controller 1 was also requested. It then calls a debounce routine which must temporarily disable the interrupts while reading the joystick(s). It then gets the spinner count and resets it to zero. It then reads and decodes the fire buttons, keypad, and joystick data. If any of the data is the same as the last time it is placed in the user buffer. Thus, it takes two successive calls with identical data to register data and effect a debounce. If the spinner was enabled, the spinner count is added to the previous spinner value in the user's table.

The user's table is ordered in this fashion:

```
0       Joystick 0             KEYPAD VALUES          JOYSTICK
1       Right Fire button 0      1    2    3                  1
2       Left Fire button 0       4    5    6                9   |  3
3       Keypad value 0           7    8    9                  \  | /
4       Spinner 0                10    0   11               8----|----2
5       Joystick 1                                           /  | \
6       Right Fire button 1    12 purple button (*3)        12   |   6
7       Left Fire button 1     13 blue button  (#3)             4
8       Keypad value 1  15 blue and purple
9       Spinner I              15 no key pressed                    0
        is off
```

Note that the EOS has an error in decoding the blue/purple button combination. In EOS 6, you can change byte E205 to 14 to return the correct data when those buttons are pressed together.

EXAMPLES:

```
        LD      A,3     ;read both.
        LD      IX,TABLE        ;put data here.
        CALL    FD3E
        LD      IX,TABLE        ;restore pointer.
        LD      A, (IX+1)       ;get right button.
        OR      (IX+2)  ;OR with left button.
        JR      NZ,PUSHED       ;at lest one button pushed.
```

UPDATE SPINNER

```
JUMP TABLE ADDRESS:    FD41

ENTRY:  none

EXIT:   DE      preserved
        IX      preserved
        IY      preserved
        others  destroyed
```

DESCRIPTION:

This routine checks the spinner value in both joysticks and increments or decrements the internal spin table if either was active. The next time that READ GAME CONTROLLER(100) is called, the spinner's incremented value will be added to the previous value in the user table - IF AND ONLY IF the spinner bit was enabled when the call was made.

One spinner is the roller controller, when set in Roller Controller mode of operation. Unfortunately, the spinner spins so fast that you must call this routine very frequently in order to CATCH most of the spinner action. It seems the spinner can interrupt the Z-80 and branch to 38H. Thus, you should be able to set up an interrupt at 38H which branches to FD4l. In this way, you would be able to update spinner values at a fast rate and interpret the spinner update when you call READ GAME CONTROLLER. Unfortunately, I have not had any success in setting up an interrupt handling routine.

EXAMPLES:

This routine uses SmartBasic's PDL routines as a vehicle for reading and updating the spinner. There is no PDL function to return the SPINNER value, so it is necessary to PEEK directly into the joystick table. Unfortunately, BASIC reads the spinner at the wrong place and it is necessary to FIX its calling convention. Each time you CALL READ J0YSTICK (PDL in BASIC), the EOS spinner value is extracted and cleared. It is sent to the USER only if requested. This allows you to CLEAR the current SPINNER update by CALLING READ JOYSTICK with the SPINNER bit off.

```
100 POKE 26914, 3:POKE 26929, 3:REM turn off SPINNER read at wrong place

110 POKE 27115, 131:REM turn SPINNER read at correct place

115 p=PDL(0):REM read any joystick

120 PRINT PEEK(16787), PEEK(16782), CHR$(128):REM this is the new spinner

121 FOR x=l TO 1000:CALL 64833:NEXT:REM make several CALLS to UPDATE spinner

122 PRINT " "; PEEK(65112), PEEK(65113),:REM this is update since last time

130 GOTO 115:REM keep on repeating until ^C
```

SOUND ROUTINES

SOUND INITIALIZATION

```
JUMP TABLE ADDRESS:    FD50


ENTRY:  B       number of entries to create
        HL      pointer to list

EXIT:   all registers used
```

DESCRIPTION:

This routine sets up an internal pointer to the user's list of songs, or special effects. It then sets up the pointer to each of the 4 voices to an END-OF-SONG so they will all be synchronized when START SOUND(105) is called. The routine then falls through to SOUND OFF on the next page.

Sound CONTROL data is in coded binary format, with the high nibble representing the volume and frequency for the 4 voices:

```
1000    frequency voice 1       The lower nibble contains the data

1001    volume   voice 1        itself.  Either attenuation from 0 to 15

1010    frequency voice 2       or the two most significant bits of the

1011    volume   voice 2        frequency.  Following are the low nibble

1100    frequency voice 3       control codes for the SOUND voice:

1101    volume   voice 3        0100    white noise

1110    noise control           0000    periodic noise

1111    noise volume    They can be ORed with the frequency shift:

                                0000    high pitch

                                0001    medium pitch

                                0010    low pitch

                                0011    varied by voice 3
```

EXAMPLES:

SOUND ROUTINES

SOUND OFF

JUMP TABLE ADDRESS: FD53

```
ENTRY:  none

EXIT:   A       255

        C       sound port

                others  preserved
```

DESCRIPTION:

This handy routine turns the volume off on all voices which effectively kills any sound in progress. It should be used whenever an operation (which may have used sound) is aborted. It can also be useful from SmartBasic when a program crashes or ends abruptly leaving the sound blasting. Just CALL 64851 to turn it off.

EXAMPLES:

SOUND ROUTINES

START SOUND

```
JUMP TABLE ADDRESS:    FD56

ENTRY:  B       sound number to start playing

        HL      pointer to end of sound data


EXIT:   A       =0 if successful, non zero value means sound in progress
        HL      pointer to next note
        IX      pointer to sound data table
        all other registers used except IY
```

DESCRIPTION:

This routine gets the relevant vector for the requested sound into IX. It is the programmer's responsibility to save IX for ending the sound. It then checks to see, if a sound is currently in progress and simply returns if so. You must check for A=0 with an OR A instruction since the routine always returns with the zero flag set regardless of success or failure.

The next step is to record the sound number in the internal buffer. After that, the routine decrements the pointer to sound data to extract the address of the NEXT NOTE to play and places it in its PLAY buffer.

Note that no sound is generated by this routine, it just sets up the pointers used by PLAY SOUND on the next page.

EXAMPLES:

SOUND ROUTINES

PLAY SOUND

```
JUMP TABLE ADDRESS:    FD59

ENTRY:  none

EXIT:   Zero flag set means sound is over
        all registers used except IY
```

DESCRIPTION:

This routine goes through each of the 4 voice data areas associated with a SOUND. It sends the volume and frequency for each voice. It ends by checking if the SOUND is over and returns with the zero flag set if so. The idea of this routine is to process one set of sound values, return control to the user, and play the next set of values on the next call. Byte 3 is always a frequency value. Byte 4 may be either a voice or a volume commands. Bit 4 of the sound data determines which type of command is sent.

BYTE

```
   0     flag for active voice-channel number or 62(3E) if sound effect.

   1&2   address of sound effect handler if 62 in byte 0



        you may notice that the frequency data is
   3    f2  f3  f4  f5  f6  f7  f8  f9        not in the correct pattern to
send to the
                              sound port.  The sound manager takes care
   4    0    0   0   0   0    0   f0  f1      of shifting bits around.
        .voice ID.             see page 80 for details on voice.


   5    note length.  This value determines if a note is to be repeated
(unchanged) several times.  Set to 1 to play once.


   6    Sweep code.  If zero, the note is not to be swept.  The lower nibble
determines how many SWEEPS to perform and is decremented on each pass. Once
it becomes zero, it is reset by using the upper nibble value.


        7       Sweep value. If zero, the note is not swept.  Other values
range from 1 to 127 and -1 to -128.  This value is ADDED to the base
frequency (bytes 3 and 4) to vary the frequency on the next pass.  Sweeps
always work on fixed increments.


   8    Attenuation (volume) sweep.  If zero, then no sweep.  It is
decremented and reset as byte 6.
```

```
   9    The top four bits determine the VOLUME change to be applied to the
note.  A value of 1 increases attenuation by one and a value of 15 decreases
attenuation by 1.  Any carry (over 16) causes the volume to go from high to
low or reverse.
```

Sound programming is very complex and requires HOURS of analysis and experimentation.... I have not tried. Basically, you set up an NMI interrupt vector to pass control to the SOUND MANAGER on a regular basis. This allows you to go about your business while a sound is being played.

END OF SOUND

```
JUMP TABLE ADDRESS:    FD5C

ENTRY:  DE     pointer to sound number
        HL     pointer to next note
        IX     pointer to current sound table

EXIT:   All registers used except IX and IY
```

DESCRIPTION:

The input values for HL and IX are the values returned by START SOUND(105). The programmer should have saved these to call END SOUND. The routine saves the pointer to next note in the sound table, gets the sound number and turns off the requested sound.

If all this sounds complicated, it is! Besides, the play sound routines can handle complex instructions like frequency or volume sweeps through a variety of control codes. Several hours of decoding of the associated routines would be required in order to make full use of the sound routines.

Considering all of that, it is no surprise that this EOS area is commonly used by programmers to install special drivers like RAM DISK or HARD DRIVE interfaces.

EXAMPLES:

DECREMENT LOW NIBBLE

```
JUMP TABLE ADDRESS:    FD44

ENTRY:  HL     pointer to byte to be decremented

EXIT:   Zero flag set if nibble becomes zero
        Carry set if result negative
```

DESCRIPTION:

This routine can be quite useful when working with BINARY CODED DECIMAL numbers. HI. is used as a pointer to the byte containing two digits in question. A call to this routine reduces the least significant digit by one and sets the zero flag if it becomes zero. The carry flag will be set if it becomes negative. The programmer can then affect a borrow on the high nibble to correct the subtract error.

EXAMPLES:

```
        LD      HL,NUMBER               ;point to the data.
        CALL    FD44                    ;decrement it.
        JR      NC,NOBORROW             ;no need to borrow.
        CALL    FD47                    ;make a borrow.
        JR      NC, BORROW              ;so far so good.
```

;

;here you need to extend the borrow to the next most significant byte

;

BORROW:

```
                LD      A, (HL)         ;get the byte.

                AND     0F0H            ;keep the top nibble.

                ADD     A, 9    ;was originally zero so now 9 after borrow.

                LD      (HL) ,A ;write it back.
```

NOBORROW:

```
;carry on with routine.
```

DECREMENT HIGH NIBBLE

JUMP TABLE ADDRESS: FD47

```
ENTRY:  HL     pointer to byte

EXIT:   Zero flag set if nibble becomes zero
        Carry flag set if result negative
```

DESCRIPTION:

This routine works like the one on the previous page except that it decrements the upper half of the byte in HL. If a borrow is required, it must be taken from the next most significant byte (if any).

EXAMPLES:

SUBROUTINES

MOVE HIGH NIBBLE TO LOW NIBBLE

```
JUMP TABLE ADDRESS:    FD4A

ENTRY:  HL     address of byte to move

EXIT:   all registers preserved except A
```

DESCRIPTION:

This routine COPIES the most significant nibble of the byte in HL to the least significant nibble. Other than a precursor to a BCD divide by ten sequence, I can't think of a particularly useful purpose for this routine.

It is used, however, by the sound routines to RESET sweep values. Sweep values range from 0 to 15 and are initially placed in both halves of a byte. The low nibble is decremented until it becomes zero and then reset with the high nibble.

EXAMPLES:

```
        LD      HL, COUNTER     ;hi-lo nibble as described above
        CALL    FD44    ;decrement low nibble
        RET     NZ      ;not ready to take action yet
        CALL    FD4A    ;reset the counter in low nibble
```

;

;here take whatever action is required... one "pass" has been performed.

;

ADD A AND (HL)

```
JUMP TABLE ADDRESS:    FD4D


ENTRY:  A      signed value to add

        HL     pointer to 16 bit number


EXIT:   A      varies
        B      destroyed
        others preserved
```

DESCRIPTION:

This routine adds a signed value in A from +127 to -128 to a 16 bit number pointed to by register HL. The value is stored in the usual LSB, MSB fashion. It adds the value in A to location HL and writes it back. It then adds the carry and sign to the most significant byte.

EXAMPLES:

EOS DATA TABLES

Following is a description of some of the data tables in EOS 6. Most of them are the same in EOS 7.

```
FC17 to FC26   the numbers 0 to 15 representing the memory configurations.
FC27                  port number for bank switching.
FC28                  port number for network reset.
FC29                  VDP control port.
FC2A                  VDP data port.
FC2B                  Joystick 0 data port.
FC2C                  Joystick 1 data port.
FC2D                  Strobe set port (read second half of joystick).
FC2E                  Strobe reset port (read first half of joystick).
FC2F                  Sound port.
FC30 to FD5F   EOS Jump table.. .see next page.
FD60                  EOS revision number.
FD6l FD62             Current VDP mode (registers 0 and 1).
FD63                  VDP status (on last call).
FD64 FD65             Pointer to Sprite attribute table.
FD66 FD67             Pointer to Sprite pattern table.
FD68 FD68             Pointer to Name table.
FD6A FD6B      Pointer to Pattern table.
FD6C FD6D      Pointer to Colour table.
FD6E                  Current memory configuration.
FD6F                  Last storage device used.
FD70 FD7l             Address of current PCB.
FD75                  Last keypress register (after call to read keyboard).
FD76 FD85             Printer buffer.
FD86                  Media size for INIT.
FD87 FD8A      Current block of storage device.
FDA0 FDB9      Internal search buffer for directory work.
FDBA FDD3      Internal file control block.
FDD4                  Current file number in directory.
FDE0 FE1B      File Manager (see page 96).
FE1C FE57             Stack.
FE58 FE6l             Debounce table for joysticks.
FE62 FE6D             Temporary stack used when switching memory banks.
FE6E FE78             Sound data tables.
FE79                  Character under cursor.
FE7A                  Left margin.
FE7B                  Right margin.
FE7C                  Top margin.
FE7D                  Bottom margin.
FE7E FE9E             Screen buffer (only 33 characters).
FE9F                  Number of lines.
FEA0                  Number of columns.
FEA1 FEA2      Home cursor address.
FEA3 FEA4      Pointer to name table.
FEA5 FEA6      Current cursor position.
FEC0 FEC3      Processor Control Block (PCB).
FEC4 FFFF             Device Control Blocks (DCB); see page 94
```

EOS JUMP TABLE

This table lists only some of the jump vectors in the EOS. The ones which serve no purpose and those which are disabled have been omitted. The addresses are written in decimal and HEX. The second column indicates the actual address of the routine in EOS 6.

```
64560:FC3O     JP      F832:63538     initialize EOS
64563:FC33     JP      F627:63015     console display (no special characters)
64566:FC36     JP      F5DG:62940     console initialization
64569:FG39     JP      F60A:62986     display character on screen (process
specials)
64572:FC3G     JP      F95F:63839     delay after hard reset
64575:FC3F     JP      F5B8:62904     end print buffer
64578:FC42     JP      F57C:62844     end print character
64581:FC45     JP      FAE2:64226     end read one block
64584:FC48     JP      FBA5:64421     end read printer
64587:FC4B     JP      F4E0:62688     end read keyboard
64590:FC4E     JP      FBl3:64283     end write one block
64593:FC51     JP      FBEl:6448l     end write printer buffer
64596:FC54     JP      F446:62534     find DCB
64602:FC5A     JP      FA4C:64076     get PCB address
64605:FC5D     JP      F8F6:63734     hard INIT
64608:FC60     JP      F94B:63819     hard reset net
64611:FC63     JP      F515:62741     print buffer
64614:FC66     JP      F4FC:62716     print character
64617:FC69     JP      FA9E:64158     read one block
64620:FC6C     JP      F4BA:62650     read keyboard
64629:FC75     JP      FA87:64135     read device-return code
64635:FC7B     JP      FA2F:64047     relocate PCB
64638:FC7E     JP      F473:62579     request status
64641:FC8l     JP      F4CB:62667     keyboard status
64644:FC84     JP      F5D2:62930     printer status
64650:FC8A     JP      F9CB:63947     scan net for devices
64653:FC8D     JP      F922:63778     soft initialization
64656:FC90     JP      FA5D:64093     soft reset device
64659:FC93     JP      FA5l:64081     soft reset keyboard
64662:FC96     JP      FA55:64085     soft reset printer
64668:FC9C     JP      F580:62848     start print buffer
64671:FC9F     JP      F56D:62829     start print character
64674:FCA2     JP      FAC6:64198     start read one block
64677:FCA5     JP      FB86:64390     start read printer
64680:FCA8     JP      F4D0:62672     start read keyboard
64683:FCAB     JP      FAFF:64255     start write 1 block
64686:FCAE     JP      FBC2:64450     start write printer
64689:FCBl     JP      F970:63856     synchronize clocks
64692:FGB4     JP      FAB2:64178     write 1 block
64695:FCB7     JP      FB75:64373     write printer
```

EOS JUMP TABLE

```
64698:FCBA          JP      EEEA:61162      initialize file manager
64701:FCBD          JP      F323:62243      initialize directory
64704:FCC0          JP      EA00:59904      open file
64707:FCC3          JP      EB04:60164      close file
64710:FCC6          JP      EB6C:60268      reset file
64713:FCC9          JP      E690:59024      make file
64716:FCCC          JP      E61B:58907      find file 1
64719:FCCF          JP      E651:58961      update file in directory
64722:FGD2          JP      ECl7:60439      read file
64725:FCD5          JP      ED8F:60815      write file
64728:FCD8          JP      EEC5:61125      put date
64731:FCDB          JP      EED4:61140      get date
64734:FCDE          JP      F10F:61711      rename file
64737:FCE1          JP      Fl4E:61774      delete file
64740:FCE4          JP      F488:62600      get device status
64743:FCE7          JP      FA94:64148      exit to WP
64749:FCED          JP      F241: 62017     trim file
64752:FCF0          JP      F089:61577      find file in FCB
64755:FCF3          JP      Fl7B:6l819      read block
64758:FCF6          JP      FlE6:61926      write block
64761:FCF9          JP      FOD9:61657      mode check
64764:FCFC          JP      EF0B:61l95      look up file in directory
64767:FCFF          JP      E618:58904      find file 2
64785:FDll          JP      El91:57745      set VDP port data
64788:FDl4          JP      E185:57733      switch memory banks
64791:FD17          JP      E153:57683      put ASC in VDP
64794:FD1A          JP      E000:57344      write VRAM
64797:FDlD          JP      E0lA:57370      read VRAM
64800:FD20          JP      E034:57396      write VDP register
64803:FD23          JP      EO4F:57423      read VDP register
64806:FD26          JP      E059:57433      fill VRAM
64809:FD29          JP      E066:57446      initialize VRAM tables
64812:FD2C          JP      E0C9:57545      put VRAM
64815:FD2F          JP      E0CF:57551      get VRAM
64818:FD32          JP      E10A:57610      calculate pattern position
64821:FD35          JP      E129:57641      point to pattern position
64824:FD38          JP      E149:57673      load default ASCII to VDP
64827:FD3B          JP      E1C5:57797      write sprite table
64830:FD3E          JP      E253:57939      read game controller
64833:FD4l          JP      E2A4:58020      update spinner
64836:FD44          JP      E355:58197      decrement low nibble
64839:FD47          JP      E35F:58207      decrement high nibble
64842:FD4A          JP      E369:58217      move high nibble to low nibble
64845:FD4D          JP      E374:58228      add A + (HL)
64848:FD50          JP      E3AB:58283      sound initialization
64851:FD53          JP      E3D1:5832l      sound off
64854:FD56          JP      E3E7:58343      start sound
64857:FD59          JP      E406:58374      play sound
64860:FD5C          JP      E4B8:58552      end of sound
```

ERROR CODES

The EOS returns a variety of error codes. Several of these are errors returned by the Device Control Blocks. In order to interpret these from the table below, the high bit must be stripped with an AND 7F.

```
        1               DCB not found
        2               DCB busy
        3               DCB idle
        4               No date
        5               No file
        6               File exists or printer busy
        7               No available FCB
        8               Match error, file incompatible
        9               bad file number (greater than 2).
10      End of file (reading past end of file).
11      File too big
12      Directory full or no key pressed on keyboard read
13      Storage media full
14      File number error
15      Rename error
16      Delete error
17      Range error or bad mode
18      Synchronize error on clock
19      Synchronize error byte 2
20      Mode incompatible with access request; e.g. reading a read protect
file
21      Media status error
22      Device error, usually with tapes or disks
23      Program non existent
24      Storage medium fails directory validity check
27      Device time out
```

MEMORY BANKS

The ADAM can access 64K of memory at a given time. This memory, however can be switched in different combination of upper and lower 32K segments. These provisions allow the switching in of a GAME CARTRIDGE, the EOS ROM, Expansion memory$_1$ etc. Following are the memory configuration codes to send to the BANK

SWITCH ROUTINE(15):

CODE	LOWER 32K	UPPER 32K
0	SmartWriter ROM	Normal RAM
1	Normal RAM	Normal RAM
2	Expansion RAM	Normal RAM
3	Os 7	Normal RAM
4	SmartWriter ROM	Expansion ROM
5	Normal RAM	Expansion ROM
6	Expansion RAM	Expansion ROM
7	OS 7	Expansion ROM
8	SmartWriter ROM	Expansion RAM
9	Normal RAM	Expansion RAM
10	Expansion RAM	Expansion RAM
11	OS 7	Expansion RAM
12	SmartWriter ROM	Cartridge ROM
13	Normal RAM	Cartridge ROM
14	Expansion RAM	Cartridge ROM
15	Os 7	Cartridge ROM

The default set up for cartridge games is 15: use the OS 7 in lower memory and the cartridge in upper.

NORMAL RAM setting is used by SmartBasic and several other software. Generally, software which uses the blue and yellow SmartKey displays use OS 7 in lower memory and normal RAM in upper (3).

Expansion ROM was intended to be used for some ROM based software which could instantly be booted like SmartWriter when you power up the system.

SmartWriter uses configuration 0.

DEVICE CONTROL BLOCK STRUCTURE

The ADAM has more than one processor controlling its operations. The USER uses the Z-80 processor, and the network is controlled by several 6801 processors. The main 6801 accesses device control blocks in the EOS to and receive data according to the commands placed in the device control blocks. Each DCB is 21 bytes organized in the following fashion:

send

```
0               Command and/or status byte
1-2             Address where to put or get data
3-4             Size of device buffer (Keyboard-1, Printer-16)
5-8             Block number requested (for tapes and disks)
9               Secondary device (see below)
10-13           Unused
14-15           Retry count, how often before quitting
16              Device number
17-18           Maximum length of device buffer
19              Device type (0-character, 1-block)
20              Device dependent status flag (the one that detects missing
media)
```

Device commands are placed in DCB+0 and the device acknowledges by placing a value 80HEX or greater in DCB+0. If the value is greater than 80HEX, there is an error. There are only 5 commands:

```
        0               Idle, effectively disables the device
        1               Request status
        2               Reset device
        3               Write
        4               Read
```

The EOS has room for 15 devices. Following is a list of what COLECO had expected to supply as devices:

```
1               Keyboard                8       Tape 1
2                       Printer                 9       Tape 3
3               Copywriter              10      ?
4               Disk 1                  11      Modem (not the Adam Link)
5               Disk 2                  12      High Resolution Monitor
6               Disk 3                  13      Centronics Interface
7               Disk 4                  14      RS-232 Interface
                                        15      Gateway
```

Note that tape 2 is not listed. That is because tapes 1 and 2 share the same DCB. When requesting an operation on tape 2, the request is sent to the DCB for tape 1 with the number 18HEX in the secondary device ID in byte 9. The device status flag (20) is split in two nibbles. The upper represents the alternate device while the lower represents the main device. This status bit is similarly coded for the disk drives as well.

FILE CONTROL BLOCK STRUCTURE

The EOS has three file control blocks. The first is used exclusively for directory searches and cannot be utilized for a user file. The other two are used by the EOS in response to an OPEN FILE(38) request. Thus you can have two files opened at once (e.g. read form one and write to the other) provided they are on the same medium. A file control block has 36~bytes and is organized as follows:

```
0-11    File name
12      File attributes
13-16   Start block of file
17-18   Blocks allocated to the file
19-20   Blocks actually used
21-22   Bytes used in the last block
```

----------same as a directory entry

```
23      Device number corresponding to the file
24      File mode       0-unused (available)
        1-read
        2-write
        3-update
        4-fill rest of space on medium
25-28           Current block being used
29-32           Last block of file (or volume size if Directory FCB)
33-34           Byte position in current block
35              Length of FCB
```

While it may be interesting to know what the structure of an FCB is, the programmer normally does not need to worry about the details. The EOS file operation functions can take care of all the housekeeping related to a file.

FILE MANAGER STRUCTURE

The Fi1e Manager is an internal data storage area used by the EOS to process file commands. The various routines put and get information there when working with files. The File manager is 60 bytes long:

```
0       System Year
1       System Month
2       System Day
         ----------------------same as directory entry
3-14    File name
15      Attributes
16-19   Start Block
20-21   Blocks allocated to file
22-23   Blocks actually used
24-25   Bytes used in the last block
26      File Year
27      File Month
28      File Day
                      ------------------------same as directory entry
29-30   Pointer to File Control Block
31-32   Pointer to Directory buffer
33      File number
34-35   Bytes requested (read or write)
36-37   Bytes not processed, yet
38-39   User's buffer to send/receive data
40-41   Pointer to FCB buffer
42-43   Pointer to end of FCB buffer
44-47   Block number currently working on
48-49   Pointer to user's file name
50-53   Block number or volume size
54-57   Block number for start of BLOCKS LEF'T (after adding current file
size)
58-59   Size of BLOCKS LEFT; space remaining on medium
```

As with the file control blocks, the EOS routines take care of manipulating all the information described above.

SAMPLE PROGRAM

;This program illustrates several of the EOS functions described in this manual. It has been deliberately abbreviated and certain cosmetic improvements suppressed in order to keep it short. You will note that the bulk of the program is data in the form of messages. Although the entire package may appear lengthy, adding more functions will add very little to the existing code since most of the groundwork has already been laid. This is the advantage of creating modular routines which can be shared.

;

;The program has 5 functions accessed by pressing Smart Keys

```
;       I       string  input from the keyboard
;       with backspace editing
```

;

```
;       II      ordinary string printing using extra
;               line feeds for spacing
```

;

```
;       III     How to move the cursor around
;       and turn on inverse video
```

;

```
;       IV      writes data to a file
```

;

```
;       V       reads data from a file
```

;

;It is worthwhile to study the entire code as there are quite a few programming tricks included within the code itself. At the end of the code ;listing, you will find full HEX listing in order to POKE the data in. To make a self booting disk, a simple method is to copy blocks zero and one ;of your SmartBasic medium to a new medium. Then delete all the files form the directory EXCEPT the BASICPGM file. The next step is to ;write all the data starting at block 2 of the media. This can be easily done with CLONE.COM in TDOS. DO NOT change the file size in the ;directory unless you change the corresponding parameter in the boot block.

;

;If you would like a CP/M disk copy of the source code along with a BOOTABLE version of the DEMO, please send $5.00 to the author. All ;profits will be channeled back into ANN to fund other projects.

;

;Let's start by defining a few equates for the assembler

```
;
;**************************************
;*      JUMP TABLE VECTORS     *
;**************************************
;
F'CBDEVICE              EQU     0FD6FH          ;where EOS stores the default
drive
CONDISP                 EQU     0FC33H          ;print characters without
controls
INITCON                 EQU     0FC36H          ;set up the window
CONOUT                  EQU     0FC39H          ;print character with control~
END_RD_KYB              EQU     0FC4BH
START_RD_KYB            EQU     0FCA8H
OPENF                   EQU     0FCC0H  ;open file
CLOSEF                  EQU     0FCC3H
QUERYF                  EQU     0FCCCH  ;find exact file name (including type)
ISFILE                  EQU     0FCFFH          ;does file name exist
MAKEF                   EQU     0FCC9H
READF                   EQU     0FCD2H
WRITEF                  EQU     0FCD5H
GO_WP                   EQU     0FCE7H          ;exit to SmartWriter
WR'IVREG                EQU     0FD20H
FILLVRAM                EQU     0FD26H
INITTBL                 EQU     0FD29H          ;set up VDP table

LOADASCII               EQU     0FC38H          ;get ASCII in default location
PUTASCII                EQU     0FC17H          ;put ASCII in specified place
```

```
;
;****************************************
;*                        KEYBOARD KEYS   *
;****************************************
;
CTLC    EQU     03H     ;abort key CONTROL-C
LEFT    EQU     0A3H    ;keft arrow
SK1     EQU     81H     ;smart keys
SK2             EQU     82H
SK3             EQU     83H
SK4             EQU     84H
SK5             EQU     85H
SK6             EQU     86H

;
;****************************************
;*      OTHER EQUATES   *
;****************************************
;
ESC     EQU     1BH     escape
EOS     EQU     0H      end-of-string
BS      EQU     8H      ;back space
LF      EQU     0AH     ;line feed
CR      EQU     0DH     ;carriage return
CLRSCR  EQU     0CH     ;clear screen

;
;****************************************
;*      LET'S GET GOING*
;****************************************
;
```

```
        LD      HL,45EDH        ;a RETURN NMI
        LD      (66H),HL        ;disable NMI ASAP
        LD      SP,STACK        ;set up local stack
        LD      A,(FCBDEVICE)   ;get the boot device
        LD      (DRIVE),A       ;save boot drive
```

;

;SET UP SCREEN

;

```
        CALL    SETBORD0        ;set border colour
        LD      BC,0    ;send 0 to register 0
        CALL    WRTVREG
        LD      BC,0lC0H        ;send C0 to register 1
        CALL    WRTVREG ;set up VDP mode
        LD      HL, 0
        LD      A, 2    ;pattern name
        CALL    INITTBL ;set the table
        LD      HL,0340H
        LD      A,4     ;pattern colour
        CALL    INITTBL
        LD      HL,0800H
        LD      A,3     ;pattern generator
        CALL    INITTBL
        LD      HL,3800H
        LD      A,1     ;sprite pattern
        CALL    INITTBL ;though unused here, we must
        LD      HL,3880H        ;make sure they do not conflict
        XOR     A       ;=0 sprite attributes
        CALL    INITTBL
        CALL    SETNORM ;set colours
        CALL    SETINV
        LD      A,' '   ;use spaces
        LD      DE,0300H        ;fill whole screen
        LD      HL,0    ;at the pattern table
        CALL    FILLVRAM
        CALL    LOADASCII       ;gimme some characters
        LD      HL,0

        LD      BC, 80H

        LD      DE,0C0H

        CALL    PUTASCII        ;and some inverse

        LD      BC,1E17H        ;30 columns, 23 rows

        LD      HL, 0000        ;pattern name table

        LD      DE,l00H ;home position
```

```
        CALL    INITCON ;set it up

        CALL    START_RD_KYB    ;so we can use END READ to get
```

;

SAMPLE PROGRAM

```
;
;
;
;**********************************
;*      MAIN MENU                 *
;**********************************
;
MAIN:           CALL    CLS     ;clear off a
                LD      DE,0    ;send ~
                LD      HL,MAINMENU     ;print this message
                CALL    PRTLOC  ;print at (DE)
MAIN0:          CALL    CIN     ;get a character, wait for it
                CP      CTLC    ;is it abort?
                JP      Z,QUIT

                LD      DE,MAIN ;set to return to main

                PUSH    DE      ;after each function

                CP      SK1

                JP      Z,KEYIN ;demo keyboard

                CP      SK2

                JP      Z,PRTMSG        ;demo printing

                CP      SK3

                JP      Z,GOTOXY        ;let's move around

                CP      SK4

                JP      Z,MAKEFILE      ;make a file

                CP      SK5

                JP      Z,READFILE      ;let's peek at one

                CP      SK6
```

```
                JP      Z,QUIT
```

;

;get here if no matching menu was pressed

;

```
        POP     DE      ;level off stack
        JR      MAINO   ;try again
```

;

;**********

```
;*      FUNCTION I              *
;*              Demo of keyboard input  *
```

;**********

;

KEYIN:

```
        LD      HL, TYPEMSG     ,give user a prompt
        CALL    PRTSTR  ;print it
        LD      B,30    ;read no more than 30
        LD      HL, BUFFER      ;into this buffer
        CALL    C INBUFF        ;use our editor
        LD      HL, BUFFER      ;start at beginning
```

CASE:

```
        LD      A, (HL) ;get a character
        OR      A       ;is it zero?
        JR      Z,CDONE if end of string then we are done
        CALL    UPCASE  ;convert to uppercase if letter
        LD      (HL) ,A ;replace character even if same
        INC     HL      ;advance pointer
        JR      CASE    ;loop until we are done
```

CDONE:

```
        LD      HL,YOUSAID      ;let user know we are echoing
        CALL    PRTSTR
        LD      HL,BUFFER       ;point to buffer
```

```
        CALL    PRTSTR  ;print it
        CALL    CRLF    ;next line
        CALL    HITKEY  ;print ANY KEY message and get key
        RET             ;done this function
```

;

```
;
;***********************************
;*      FUNCTION II              *
;*      Printing  to Screen      *

;***********************************
;
PRTMSG:
        LD      HL,HEREMSG      ;print at current position
        CALL    PRTSTR
        CALL    HITKEY  ;wait for user to be ready
        LD      HL, SCROLL      ;this is a message which forces scroll
        CALL    PRTSTR
        CALL    HITKEY
        LD      HL,SKIPLINE     ;this message skips a few lines
        CALL    PRTSTR
        CALL    HITKEY
        LD      HL, LONG        ;this long messages wraps around
        CALL    PRTSTR
        CALL    HITKEY
        RET

;
;***********************************
;*              FUNCTION III            *
;*              Moving the cursor       *

;***********************************
;
GOTOXY:
        LD      HL, WATCHIT     ;print this message
        LD      DE,1005H        row 16 column 10
        CALL    PRTLOC  ;routine places cursor and prints
        LD      B,30    ;we'll move 30 times
        LD      HL,MOVET        ;using the table below

MOVEC:
```

```
        LD      A, (HL) ;get one
        CALL    COUT    ;send it
        LD      DE,33333        ;wait for this long
```

WAIT:

```
        DEC     DE
        LD      A, E
        OR      D       ;is DE=0?
        JR      NZ,WAIT ;let's wait some more
        INC     HL      ;advance pointer
        DJNZ    MOVEC   ;repeat until B=0
        LD      A, 24   ;clear to end of screen
        CALL    COUT    ;print the character
        LD      BC,65535
        CPIR            ;another non destructive way to wait
        CALL    IVON    ;set to print high bit
        LD      HL, IVMSG       ;print this message
        CALL    PRTSTR  ;will use alternate colour
        CALL    IVOFF   ;back to normal video
        CALL    HITKEY
        RET
```

MOVET:

```
        DB      160,160,160,163,163,163,160,163,160,163
        DB      160,160,160,163,163,163,160,163,160,161
        DB      160,163,160,161,160,163,160,161,160,163
```

;

;**********

```
;*              FUNTION IV              *
;*              Writing to file                 *
```

;**********

;

MAKEFILE:

```
        LD      HL, GETFNAME
        CALL    PRTSTR  ;ask user for a file name
        LD      HL, BUFFER      ;put it here
        LD      B,11    ;max 11 characters
        CALL    CINBUFF ;do it
        LD      HL, BUFFER      ;reset to start
        XOR     A       ;look for null
```

```
        LD      BC,11   ;we know there is a null
        CPIR            ;within 11 bytes for a name
        LD      (HL) , 3        ;mark end of file name
        DEC     HL
        LD      (HL) , 'A'      ;file type
```

;

;

```
        LD      A, (DRIVE)      ;look on default device

        LD      DE,BUFFER       ;match this name

        LD      HL, DIRENTRY    ;save here if found

        CALL    ISFILE  ;match type A or H of file

        JR      NZ,NOFILE       ;good, we're not killing anything

        LD      HL, EXISTS      ;tell user

        CALL    PRTSTR  ;the file is there

        JR      MAKEFILE        ;and try again
```

NOFILE:

```
        LD      DE,HEREMSG      ;start here
        LD      HL,WATCHIT      ;end here
        OR A            ;clear carry
        SBC     HL,DE   ;HL=size of file

        LD      B,H

        LD      C,L     ;copy to BC

        LD      HL, FILEBUFF    ;file start
```

MOVEDATA:

```
        LD      A, (DE)         ;get a byte

        CP      LF

        JR      Z,IGNORE        ;remove all formatting

        CP      CR

        JR      Z,PUTSPC        ;put space for <CR> for wrap
```

```
        CP      EOS
        JR      NZ,PUTCHR       ;if not END then put the character
        LD      A, CR   ;now put a <CR>
        JR      PUTCHR
```

PUTSPC:

```
        LD      A,' '   ;a space
```

PUTCHR:

```
        LD      (HL) ,A ;put in buffer
        INC     HL      ;increment buffer
```

IGNORE:

```
        INC     DE      ;increment raw source
        DEC     BC      ;decrement byte count
        LD      A,B
        OR C            ;is BC=0?
        JR      NZ ,MOVEDATA    ;not done yet
        LD      DE, FILEBUFF    ;this is the start
        XOR     A       ;clear carry
        SBC     HL,DE   ;this is the modified file size
        LD      BC , 0
        LD      D,H
        LD      E,L     ;BCDE=size
```

```
        PUSH    DE      ;save file size
        LD      A, (DRIVE)      ;get the default drive
        LD      HL,BUFFER       ;point to file name
        CALL    MAKEF   ;make the file
        JR      NZ, ERROR       ;oops
        LD      A, (DRIVE)      ;get drive back
        LD      B,2     ;WRITE mode
        CALL    OPENF   ;HL is still pointing to name
        JR      NZ, ERROR
        LD      (FNUM) ,A       ;save file number for close
        POP     BC      ;get back file size
        LD      HL, FILEBUFF    ;start writing here
        CALL    WRITEF  ;do it in one pass
        JR      NZ, ERROR
        LD      A, (FNUM)       ;get file number back again
        CALL    CLOSEF  ;close it off
        JR      NZ,ERROR
        LD      HL, DONEMSG     ;tell user
        CALL    PRTSTR  ;we are done
        CALL    HITKEY
        RET
```

ERROR:

```
        LD      HL, ERRORMSG    ;we're not interpreting errors here
        CALL    PRTSTR  ;just tell user
```

```
        CALL    HITKEY  ;and get out

        RET
```

;

SAMPLE PROGRAM

```
;

;************************************

;*      FUNCTION V                  *
;*      Reading from a file         *

;************************************

;

READFILE:

        LD              HL, GETFNAME    ;ask user
        CALL            PRTSTR  ;for file name
        LD              HL BUFFER       ;put it here
        LD              B,11    ;maximum length
        CALL            CINBUFF ;use our editor
        LD              HL, BUFFER      ;point to start
        XOR             A       ;look for null
        LD              BC,11   ;we know there is a null
        CPIR                    ;within 11 bytes for a name
        LD              (HL) , 3        ;mark end of file name
        DEC             HL
        LD              (HL), 'A'       ;file type
        LD      A, (DRIVE)      ;look on default device
        LD      DE, BUFFER      ;match this name
        LD      HL, DIRENTRY    ;save here if found
        CALL    QUERYF  ;exact match this time
        JR      Z, GOTFILE
        LD      HL, BADNAME     ;tell user
        CALL    PRTSTR  ;there is no file
        JR      READFILE        ;and try again

GOTFILE:

        LD      A, (DRIVE)      device
        LD      B,1     ;read mode
        LD      HL, BUFFER      ;file name
        CALL    OPENF   ;try and open
        JR      NZ, ERROR       ;oops
        LD      (FNUM) A        ;save number for close
        LD      BC,5006 ;maximum we want to read
        LD      HL, FILEBUFF    ;start of buffer
        CALL    READF   ;bring in 5000 or less bytes
        JR      Z NOERROR       ;file was at least 5000
        CP      i6      ;is it end of file?
        JR      NZ,ERROR        ;oops
```

```
;
```

;here you should flag that all file is not read yet

```
;
```

NOERROR:

```
        LD      A, (FNUM)       ;get back file number
        CALL    CLOSEF  ;we'll close it now though normally we
        JR      NZ,ERROR        ;would wait till completed.
```

```
;
```

;now we'll print the file with a word wrap.

```
;
```

```
        CALL    CRLF    ;advance to new line
        LD      DE, FILEBUFF    ;start of buffer
```

ECHOMORE:

```
        XOR     A       ;load A with 0
        LD      (DIDWRAP) ,A    ;reset the NEWLINE flag
        LD      HL, 30  ;screen width
        ADD     HL,DE   ;this is MAX end of line
```

FINDWRAP:

```
        LD      A, (HL)
        CP      ' '     ;is this character a space
        JR      Z,GOTWRAP       ;good, let's wrap here
        CP      CR      ;is it already a NEW LINE
        JR      Z,DOLINE        ;a <CR> forces wrap anyway
        DEC     HL      ;back up one character and try again
        JR      FINDWRAP        ;will crash if no space in30 characters
```

GOTWRAP:

```
        LD      (HL),255                ;mark with unique character
```

DOLINE:

```
        LD      A,(DE)  ;get character from original position
        CP      255     ;is it the NEW LINE
        JR      Z,DOWRAP        ;let's do it
```

```
        CP      CR      ;or if already newline?
        JR      NZ,SKIPWRAP     ;nope then don't wrap
```

;

```
;

;

;
```

DOWRAP:

```
                LD      (DIDWRAP) ,A    ;set the WE DID A WRAP
                CP      255             ;if (DE) was 255 then reset it
                JR      NZ, CHECKHL     ;nope, then check if (HL) was set
                LD
                LD      (DE) ,A
```

CHECKHL:

```
                LD      A, (HL)         ;always check (HL) regardless of (DE)
                CP      255             ;had we put a force there?
                JR      NZ , DOW1
                LD      (HL),' '                ;replace the stolen space
```

DOW1:

```
        LD      A, CR   ;now let's put a <CR>
        CALL    COUT
        LD      A,LF    ;followed by a <LF>
```

SKIPWPAP:

```
        CALL    COUT
        PUSH    HL      ;save HL
        LD      HL, 1000        ;wait a bit just for the heck of it
```

SLOW:

```
        DEC             HL
        LD              A,H
        OR      L       ;is HL zero?
        JR      NZ,SLOW ;wait some more
        POP             HL      ;restore HL
        INC             DE      ;advance pointer
        DEC             BC      ;one less byte in file
        LD              A,B
        OR      C       ;is file done?
        JR      Z, FILEDONE     ;then let's get out
        LD      A, (DIDWRAP)    ;did we wrap.
        OR      A       ;will be non-zero
```

```
        JR      Z,DOLINE        ;let's work on this line some more
        JR      NZ, ECHOMORE    ;let's start a new line
```

FILEDONE:

```
        CALL    CRLF    ;Skip line for clarity
        CALL    HITKEY
        RET
```

;

;*******************************

```
;*          EXIT ROUTINE               *
```

;*******************************

;

QUIT:

```
        LD      SP,OFFFFH       ;set stack in safe place
        JP      GO_WP   ;get outa here
```

;

;*******************************

```
;*         CHARACTER I/O               *
```

;*******************************

;

```
CRLF:   LD      HL,CRLFMSG      ;point to a <CR><LF>
        JR      PRTSTR  ;print it
```

;

GOTORC:

```
        PUSH    DE      ;save requested position
        LD      A,D     ;reverse DE
        LD      D,E     ;cause I like to goto XY
        LD      E,A     ;rather than GOTO YX
```

```
        INC     D
        LD      A, 1CH  ;goto xy prefix
        CALL            COUT    ;print the darn thing
        POP     DE      ;restore original request
        RET
```

;

PRTLOC:

```
        CALL    GOTORC  ;go to location in DE first
```

PRTSTR:

```
        LD              A, (HL) ;get a byte
        CP              EOS     ;is it END
        RET             Z       ;get outa here
        CALL            COUT    ;print one character
        INC             HL      ;advance pointer
        JR              PRTSTR  ;keep going til end
```

;

IVOFF:

```
        XOR     A       ;reset the IV mask
        JR      IV0
```

IVON:

```
        LD      A, 80H  ;set to add 128 in COUT routine
IV0:    LD      (INVERSE) ,A
        RET
```

UPCASE:

```
        CP      'a'     ;is it less than a
        RET     C       ;don't convert
        CP      'z'+1   ;is it bigger than z
        RET     NC      ;don't convert
        AND     5FH     strip the LOWERCASE bit
        RET
```

CLS:

```
        LD      A,CLRSCR        ;this is the clear screen code
        JP      COUT    ;print it
```

;

;reads B characters into (HL)

;space pads to end of buffer after <CR>

;

CINBUFF:

```
        LD              C,0     ;set counter
```

CIBUF1:

```
        CALL            CIN     ;get a character
        CP              CR      ;is it ENTER?
        JR              Z,CIBUF2        ;flush buffer if so
        CP              BS      ;was it BACKSPACE
        JR              Z,DESTBS        ;do it
        CP              LEFT    ;left arrow as well
```

```
        JR              Z,DESTBS
        CP              ESC     ;cancel request
        RET             Z       ;get out NOW
        GP              CTLC    ;was it abort
        JP              Z,MAIN          ;go back to MAIN
        CALL            COUT    ;echo the character
        LD              (HL) ,A         ;put it in buffer
        INC             C       ;one more character in
        INC             HL      ;advance buffer pointer
        DJNZ            CIBUF1  ;decrement MAX count & continue if safe
        CP              BS      ;was last character a BACKSPACE
        JR              Z,DESTBS        ;do it again
        CP              LEFT
        JR              Z,DESTBS
        CP              ESC     ;was last a cancel
        RET             Z
        CP              CTLC    ;was last an abort
        JP              Z ,MAIN
```

CIBUF2:

```
        LD              (HL) ,0         ;mark end of string
        LD              A,B
        OR              A
        RET             Z       ;we read in the maximum requested
```

CIBUF3:

```
        INC             HL      ;skip to next character
        LD              (HL),' '        ;space fill it
        DJNZ            CIBUF3  ;up to requested length
        RET
```

DESTBS:

```
        LD              A,C     ;what is the position
        OR              A
        JR              Z,CIBUF1        ;can't <BS> if at start
        LD              A,BS    ;print a backspace
        CALL            COUT
        LD              A ,' '  ;write a space over old character
        CALL            COUT
        LD              A, BS   ;backspace into hole
        CALL            COUT
        DEC             HL      ;reduce pointer
        DEC             C       ;and the character count
        INC             B       ;make one more available
        JR              CIBUF1  ;and try again
```

HITKEY:

```
        LD      HL,ANYKEYMSG    ;ask user to press key
        CALL    PRTSTR  ;print it and fall through
```

CIN:

```
                PUSH    BC      ;save all registers
                PUSH    DE      ;except A
                PUSH    HL
```

CIN1:

```
                CALL    END_RD_KYB      ;is there a character
                JR      NC ,CIN1        ;wait
                PUSH    AF      ;save character
                CALL    START_RD_KYB    ;restart the read
                POP     AF      ;restore character
                POP     HL      ;and all registers
                POP     DE
                POP     BC
                RET
```

COUT:

```
                PUSH    AF      ;save all registers
                PUSH    BC      ;including the character to print
                PUSH    DE
                PUSH    HL
                OR      A       ;is the character HIGH BIT
                JP      M,COUT0         ;if high bit set then control
                CP      ' '     ;is it less than 32
                JR      NC , COUT1      ;if >than space then print normal
```

COUT0:

```
        CALL    CONOUT  ;this one displays specials COUT2
        JR      COUT2
```

COUT1:

```
        OR      0
INVERSE                 EQU     $-1     ;add inverse if it is on
        CALL    CONDISP ;this one does not display specials
```

COUT2:

```
        CALL    END_RD_KYB      ;check for key waiting
        JR      NC ,COUT3       ;none so let's get out

        CALL    CIN     ;get the character

        CP      'S' -40H        ;is it CONTROL-S

        JR      NZ,COUT3        ;ignore if not

        CALL    CIN     ;wait for another character

        CP      'C' -40H        ;but abort if CONTROL-c

        JP      Z,MAIN
```

COUT3:

```
        POP     HL      ;restore all registers
        POP     DE
        POP     BC
        POP     AF

        RET
```

;

;*******************************

```
;*        SUBROUTINES           *
```

;*******************************

;

;set the border colour

;

SETBORD0:

```
        LD      A, (BORDCOL)    ;get the default border colour
        LD      C,A     ;into register C
        LD      B,07H   ;write to VDP register 7
        JP      WRTVREG ;do it and return
```

;set the normal and inverse colours

SETNORM:

```
        LD      A, (NCCOL)      ;this is the set colour
        SLA     A
        SLA     A
        SLA     A
        SLA     A       ;normal set colour *16
        LD      D,A
        LD      A,(NBCOL)       ;clear colour
        OR      D
        LD      DE,10H  ;write 16 bytes
        LD      HL, 0340H       ;starting here
        JP      FILLVRAM        ;for character 0-127
```

;set inverse colour, differently from above

SETINV:

```
        LD      A, (IBCOL)      ;clear colour
        LD      D,A
        LD      A, (ICCOL)      ;set colour
        SLA     A
        SLA     A
        SLA     A
        SLA     A
        OR      D
        LD      DE,10H  ;16 bytes
        LD      HL, 0350H       ;start here
        JP      FILLVRAM        ;for 128-255
```

;**********

```
;*      MESSAGES                        *
```

;**********

ANYKEYMSG:

```
        DB      'Press any key to continue...',EOS
```

MAINMENU:

```
        DB              'I       Keyboard input' ,CR,LF
        DB              'II      Print Message' ,CR,LF
        DB              'III     Cursor Movement' ,CR,LF
```

```
        DB              'IV     Make File',CR,LF
        DB              'V       Read File' ,CR,LF
        DB              'VI     Exit' ,CR,LF,LF,LF,LF,EOS
```

;

;used by function I

;

TYPEMSG:

```
        DB      'Type in something' ,CR,LF,LF,EOS
```

YOUSAID:

```
        DB      CR,LF,5fh,5fh,5fh,'This is what you said:',CR,LF,EOS
```

;

;used by function II

;

HEREMSG:

```
        DB      'Normally, messages are printed'
        DB      CR,LF,'at the current cursor position.'
        DB      CR,'It is up to you to remember';(last line forced wrap)
        DB      CR,LF, 'where you last left the cursor.'
        DB      CR,'If the last character of a line'
        DB      CR,'is on the margin a scroll is'
        DB      CR,LF, 'performed.  Be careful when'
        DB      CR,LF,' formatting your data.',CR,LF,LF,EOS
```

SKIPLINE:

```
        DB      CR,LF,LF, 'This message skips lines'
        DB      CR,LF,LF, 'To appear double spaced'
        DB      CR,LF,LF,'You only need an extra <LF>',CR,LF,LF,EOS
```

SCROLL:

```
        DB      CR,LF,LF, 'This -message is near the bottom'
        DB      CR, 'of the screen.  It will scroll'
        DB      CR,LF, 'smoothly when I reach the end'
        DB      CR,LF, 'of the screen.  The scroll is'
```

```
        DB      CR,LF, 'handled by the EOS.',CR,LF,LF,EOS
```

LONG:

```
        DB      CR,LF, 'This message is very long and has not been '
        DB      'formatted for the screen.  Although all the characters '
        DB      'are there, it is difficult to read because words are '
        DB      'truncated.  See the file read function for more.',CR,LF,EOS
```

```
;
;used by function III
;
WATCHIT:
                DB      'Column 5 row 16',CR,LF
                DB      'Watch the cursor move around' ,EOS

IVMSG:
        DB      CR,LF,' This message is printed     '
        DB      CR,LF,'  In inverse video. It is        '
        DB      CR,LF,'  accomplished by setting   '
        DB      CR,LF,'  the color table for the       '
        DB      CR,LF,'  high bit characters  to       '
        DB      CR,LF,'  make them another color  ',CR,LF,LF,EOS

;
;used by function IV
;
ERRORMSG:
                DB      CR,LF,LF, 'I/O error' ,EOS

DONEMSG:
                DB      CR,LF,LF, 'Operation Complete' ,CR,LF,EOS

EXISTS:
                DB      CR,LF,LF, 'File Exists, use another name' ,EOS

GETFNAME:
                DB      CR,LF,LF, 'Input File Name: ',EOS

;
```

```
;used by function V
;
BADNAME:
                DB      CR,LF,'File Not Found, try again',EOS
;
;*********************************
:*       DATA & STORAGE                 *
;*********************************
;
CRLFMSG:
                DB      CR,LF,EOS
BORDGOL:
                DB      07H     ;border colour saved here
NCCOL:
                DB      01H     ;character SET colour
NBCOL:
                DB      0FH     ;character CLEAR colour
ICCOL:
                DB      01H     ;inverse SET colour
IBCOL:
                DB      07H     ;inverse CLEAR colour
```

DRIVE:

```
                DS      1       ;default drive
```

FNUM:

```
                DS      1       file number for READ/WRITE
```

DIDWRAP:

```
                DS      1       ;word wrap flag for file print
```

;

DIRENTRY:

```
        DS      26              ;store directory entry here
```

BUFFER:

```
        DS      100             ;room for keyboard input
        DS      100             ;stack space
```

STACK:

```
FILEBUFF:                       ;use this area for file I/O

        END
```

SAMPLE PROGRAM HEX CODE FIRST BLOCK

21 ED 45 22 66 00 31 82 OA 3A 6F FD 32 9D 09 CD 87 04 01 00

00 CD 20 FD 01 C0 01 CD 20 FD 21 00 80 3E 02 CD 29 FD 21 40

03 3E 04 GD 29 FD 21 00 08 3E 03 CD 29 FD 21 00 38 3E 01 CD

29 FD 21 80 38 AF CD 29 FD CD 90 04 CD A9 04 3E 20 11 00 03

21 00 00 CD 26 FD CD 38 FD 21 00 00 01 80 00 11 00 0C CD 17

FD 01 17 1E 21 00 00 11 00 01 CD 36 FC CD A8 FC CD E8 03 11

00 00 21 DF 04 CD C9 03 CD 47 04 FE 03 CA B2 03 11 74 O1 D5

FE 81 CA AD 01 FE 82 CA DC 01 FE 03 CA 01 02 FE 84 CA 56 02

FE 85 CA FA 02 FE 86 CA B2 03 D1 48 D3 21 53 05 CD CC 03 06

1E 21 BA 09 CD ED 03 21 BA 09 7E B7 28 07 CD DF 03 77 23 18

F5 21 68 O5 GD CC 03 21 BA 09 CD CC 03 CD B8 03 CD 41 04 C9

21 86 05 CD CC 03 CD 41 04 21 CE 06 CD CC 03 CD 41 04 21 77

06 CD CC 03 CD 41 04 21 66 07 CD CC 03 CD 41 04 G9 21 32 08

11 05 10 GD G9 03 06 1E 21 38 02 7E CD 58 04 11 35 82 1B 7B

B2 20 FB 23 10 F1 3E 18 CD 58 04 01 FF FF ED B1 CD D9 03 21

60 08 CD CC 03 CD D6 03 GD 41 04 G9 A0 A0 A0 A3 A3 A3 A0 A3

A0 A3 A0 A0 A0 A3 A3 A3 A0 A3 A0 A1 A0 A3 A0 A1 A0 A3 A0 A1

A0 A3 21 64 09 CD CC 03 21 BA 09 06 OB CD ED 03 21 BA 09 AF

01 OB 0O ED B1 36 03 2B 36 41 3A 9D 09 11 BA 09 21 AO 09 CD

FF FC 20 08 21 43 09 CD CC 03 18 CE 11 86 05 21 32 08 B7 ED

52 44 4D 21 82 0A 1A FE 0A 28 10 FE 0D 28 08 FE 00 20 06 3E

0D 18 02 3E 20 77 23 13 0B 78 B1 20 E5 11 82 0A AF ED 52 01

00 00 54 5D D5 3A 9D 09 21 BA 09 CD C9 FC 20 28 3A 9D 09 06

02 CD C0 FG 20 1E 32 9E 09 C1 21 82 OA CD D5 FC 20 12 3A 9E

09 CD G3 FC 20 0A 21 2B 09 CD CC 03 CD 41 04 C9 21 1E 09 CD

CC 03 CD 41 04 C9 21 64 09 CD CC 03 21 BA 09 06 0B CD ED 03

21 BA 09 AF 01 OB 00 ED B1 36 03 2B 36 41 3A 9D 09 11 BA 09

21 A0 09 CD CC FC 28 08 21 79 09 CD CC 03 18 CE 3A 9D 09 06

O1 21 BA 09 CD C0 FC 20 B7 32 9E 09 01 88 13 21 82 0A CD D2

FG 28 04 FE 0A 20 A5 3A 9E 09 CD C3 FC 20 9D CD B8 03 11 82

0A AF 32 9F 09 21 1E 00 19 7E FE 20 28 07 FE 0D 28 05 2B 18

F4 36 FF 1A FE FF 28 04 FE 0D 20 18 32 9F 09 FE FF 20 03 3E

20 12 7E FE FF 20 02 36 20 3E 0D CD 58 04 3E 0A CD 58 04 ES

21 E8 03 2B 7G B5 20 FB E1 13 0B 78 B1 28 08 3A 9F 09 B7 28

C6 20 AE CD B8 03 CD 41 04 C9 31 FF FF C3 E7 FC 21 95 09 18

0F D5 7A 53 5F 14 3E lC CD 58 04 D1 C9 CD BD 03 7E FE 00 C8

CD 58 04 23 18 F6 AF 18 02 3E 80 32 6A 04 C9 FE 61 D8 FE 7B

D0 E6 5F C9 3E 0C C3 58 04 0E 00 CD 47 04 FE 0D 28 28 FE 08

28 2F FE A3 28 2B FE 1B C8 FE 03 CA 74 01 CD 58 04 77 0C 23

10 E1 FE 08 28 17 FE A3 28 13 FE 1B C8 FE 03 CA 74 01 36 00

78 B7 C8 23 36 20 10 FB C9 79 B7 28 G2 3E 08 CD 58 04 3E 20

CD 58 04 3E 08 CD 58 04 2B 0D 04 18 AE 21 C2 04 CD CC 03 C5

D5 E5 CD 4B FC 30 FB F5 CD A8 FC F1 E1 D1 C1 C9 F5 C5 D5 E5

B7 FA 64 04 FE 20 30 05 CD 39 FC 18 05 F6 00 CD 33 FC CD 4B

FG 30 0F CD 47 04 FE 13 20 08 CD 47 04 FE 03 CA 74 01 E1 D1

C1 F1 C9 3A 98 09 4F 06 07 C3 20 FD 3A 99 09 CB 27 CB 27 CB

27 CB 27 57 3A 9A 09 B2 11 10 00 21 40 03 C3 26 FD 3A 9C 09

57 3A 9B 09 CB 27 CB 27 CB 27 CB 27 B2 11 10 00 21 50 03 C3

26 FD 50 72 65 73 73 20 61 6E 79 20 6B 65 79 20 74 6F 20 63

6F 6E 74 69 6E 75 65 2E 2E 2E 00 20 20 49 20 20 20 4B 65 79 <--data

62 6F 61 72 64 20 69 6E 70 75 74 0D 0A 20 49 49 20 20 20 50 start

72 69 6E 74

SAMPLE PROGRAM HEX CODE SECOND BLOCK

20 4D 65 73 73 61 67 65 0D 0A 20 49 49 49 20 20 43 75 72 73

6F 72 20 4D 6F 76 65 6D 65 6E 74 0D 0A 20 49 56 20 20 20 4D

61 6B 65 20 46 69 6C 65 0D 0A 20 20 56 20 20 20 52 65 61 64

20 46 69 6C 65 0D 0A 20 56 49 20 20 20 45 78 69 74 0D 0A 0A

0A 0A 00 54 79 70 65 20 69 6E 20 73 6F 6D 65 74 68 69 6E 67

0D 0A 0A 00 0D 0A 5F 5F 5F 54 68 69 73 20 69 73 20 77 68 61

74 20 79 6F 75 20 73 61 69 64 3A 0D 0A 00 4E 6F 72 6D 61 6C

6C 79 2C 20 6D 65 73 73 61 67 65 73 20 61 72 65 20 70 72 69

6E 74 65 64 0D 0A 61 74 2O 74 68 65 20 63 75 72 72 65 6E 74

20 63 75 72 73 6F 72 20 70 6F 73 69 74 69 6F 6E 2E 0D 49 74

20 69 73 20 75 70 20 74 6F 20 79 6F 75 20 74 6F 20 72 65 6D

65 6D 62 65 72 0D 0A 77 68 65 72 65 20 79 6F 75 20 6C 61 73

74 20 6C 65 66 74 20 74 68 65 20 63 75 72 73 6F 72 2E 0D 49

66 20 74 68 65 20 6C 61 73 74 20 63 68 61 72 61 63 74 65 72

20 6F 66 20 61 20 6C 69 6E 65 0D 69 73 20 6F 6E 20 74 68 65

20 6D 61 72 67 69 6E 20 61 20 73 63 72 6F 6C 6C 20 69 73 0D

0A 70 65 72 66 6F 72 6D 65 64 2E 20 20 42 65 20 63 61 72 65

66 75 6C 20 77 68 65 6E 0D 0A 66 6F 72 6D 61 74 74 69 6E 67

20 79 6F 75 72 20 64 61 74 61 2E 0D 0A 0A 00 0D 0A 0A 54 68

69 73 20 6D 65 73 73 61 67 65 20 73 6B 69 70 73 20 6C 69 6E

65 73 0D 0A 0A 54 6F 20 61 70 70 65 61 72 20 64 6F 75 62 6G

65 20 73 70 61 63 65 64 0D 0A 0A 59 6F 75 20 6F 6E 6G 79 20

6E 65 65 64 20 61 6E 20 65 78 74 72 61 20 3C 4C 46 3E 0D 0A

0A 00 0D 0A 0A 54 68 69 73 20 6D 65 73 73 61 67 65 20 69 73

20 6E 65 61 72 20 74 68 65 20 62 6F 74 74 6F 6D 0D 6F 66 20

74 68 65 20 73 63 72 65 65 6E 2E 20 20 49 74 20 77 69 6C 6G

20 73 63 72 6F 6C 6C 0D 0A 73 6D 6F 6F 74 68 6C 79 20 77 68

65 6E 20 49 20 72 65 61 63 68 20 74 68 65 20 65 6E 64 0D 0A

6F 66 20 74 68 65 20 73 63 72 65 65 6E 2E 20 20 54 68 65 20

73 63 72 6F 6C 6C 20 69 73 0D 0A 68 61 6E 64 6C 65 64 20 62

79 20 74 68 65 20 45 4F 53 2E 0D 0A 0A 00 0D 0A 54 68 69 73

20 6D 65 73 73 61 67 65 20 69 73 20 76 65 72 79 20 6C 6F 6E

67 20 61 6E 64 20 68 61 73 20 6E 6F 74 20 62 65 65 6E 20 66

6F 72 6D 61 74 74 65 64 20 66 6F 72 20 74 68 65 20 73 63 72

65 65 6E 2E 20 20 41 6C 74 68 6F 75 67 68 20 61 6C 6G 20 74

68 65 20 63 68 61 72 61 63 74 65 72 73 20 61 72 65 20 74 68

65 72 65 2C 20 69 74 20 69 73 20 64 69 66 66 69 63 75 6C 74

20 74 6F 20 72 65 61 64 20 62 65 63 61 75 73 65 20 77 6F 72

64 73 20 61 72 65 20 74 72 75 6E 63 61 74 65 64 2E 20 20 53

65 65 20 74 68 65 20 66 69 6C 65 20 72 65 61 64 20 66 75 6E

63 74 69 6F 6E 20 66 6F 72 20 6D 6F 72 65 2E 0D 0A 00 43 6F

6C 75 6D 6E 20 35 20 72 6F 77 20 31 36 0D 0A 57 61 74 63 68

20 74 68 65 20 63 75 72 73 6F 72 20 6D 6F 76 65 20 61 72 6F

75 6E 64 00 0D 0A 20 20 20 54 68 69 73 20 6D 65 73 73 61 67

65 20 69 73 20 70 72 69 6E 74 65 64 20 20 20 0D 0A 20 20 20

49 6E 20 69 6E 76 65 72 73 65 20 76 69 64 65 6F 2E 20 49 74

20 69 73 20 20 20 0D 0A 20 20 20 61 63 63 6F 6D 70 6G 69 73

68 65 64 20 62 79 20 73 65 74 74 69 6E 67 20 20 20 0D 0A 20

20 20 74 68 65 20 63 6F 6G 6F 72 20 74 61 62 6G 65 20 66 6F

72 20 74 68 65 20 20 20 0D 0A 20 20 20 68 69 67 68 20 62 69

74 20 63 68 61 72 61 63 74 65 72 73 20 20 74 6F 20 20 20 0D

0A 20 20 20

SAMPLE PROGRAM HEX CODE THIRD BLOCK

6D 61 6B 65 20 74 68 65 6D 20 61 6E 6F 74 68 65 72 20 63 6F

6C 6F 72 20 20 20 0D 0A 0A 00 0D 0A 0A 49 2F 4F 20 65 72 72

6F 72 00 0D 0A 0A 4F 70 65 72 61 74 69 6F 6E 20 43 6F 6D 70

6C 65 74 65 0D 0A 00 0D 0A 0A 46 69 6C 65 20 45 78 69 73 74

73 2G 20 75 73 65 20 61 6E 6F 74 68 65 72 20 6E 61 6D 65 00

0D 0A 0A 49 6E 70 75 74 2O 46 69 6C 65 20 4E 61 6D 65 3A 20

00 0D 0A 46 69 6C 65 20 4E 6F 74 20 46 6F 75 6E 64 2C 20 74

72 79 20 61 67 61 69 6E 00 0D 0A 00 07 01 0F 01 07 0C

EOS DIRECTORY STRUCTURE

When looking at BLOCK 1 (The Directory), group the bytes into groups of 26 bytes. The first 26 bytes is the VOLUME-RECORD. The next groups of 26 bytes are the FILE-RECORDS. The last FILE-RECORD has a filename of BLOCKS LEFT. This indicates the end and indicates how much is left on the tape/disk.

VOLUME-RECORD

VOLUME-NAME 12 BYTES

Inited name that shows up on catalog as VOLUME.

DIRECTORY-SIZE 1 BYTE

VOLUME ATTRIBUTE BIT 7

VOL. DIRECTORY SIZE BITS 6 – 0

A 1 in the attribute indicates delete protected.

The directory size is shown as # of blocks.

DIRECTORY-CHECK 4 BYTES

Unique code 55 AA 00 FF indicates a directory exists on tape/disk.

VOLUME-SIZE 4 BYTES

Total # of blocks allocated.

Unused ? 2 BYTES

DATE (CREATION ?) 3 BYTES

1 each for Year, Month and Day.

FILE-RECORD

FILE-NAME 12 BYTES

Name of file, usually followed by file type (A, A, H, H), then a HEX 03.

FILE-ATTRIBUTE 1 BYTE

1 in the bit position indicates true conditions as:

7 – Perm. (DELETED) Protected
6 – Write Protected
5 – Read Protected
4 – User File
3 – System File
2- File flagged as deleted
1 – Execute protected (1 is no execute)

0 – Not a file (See BLOCKS-LEFT entry)

START-BLOCK 4 BYTES

Starting block number for file. Left (Hi-order) bytes usually has the value, other 3 are 00.

ALLOCATED-SIZE 2 BYTES

Number of blocks actually allocated.

USED-SIZE 2 BYTES

Number of blocks used (Full + Partial)

BYTES-IN-LAST-BLOCK 2 BYTES

Number of bytes in last block. Max value in here is 00 04. This would indicate 1024 bytes (full).

DATE (CREATED ?) 3 BYTES

1 for Year, Month and Day.

NOTE: For START-BLOCK, ALLOC/USED SIZE and BYTES-LAST.

The 2 bytes are always shown as lo-value/hi-value.

The value = left byte + (right byte * 256).

I.e. 00 04 is 0 + (4 * 256) = 1024

This coding is fairly standard in Z*) coding.

Things to look for:

DIRECTORY-SIZE most are 81 (1000 0001) which indicates delete protect, 1 block directory.

VOLUME-SIZE Tape normally FF 00 or 00 01. Disk A0. If using PACKCOPY, the copied disk from tape gets the tape FF 00.

FILE-NAME Sometimes you see parts of the name BLOCKS-LEFT remaining at the right end. This is ignored as the system reads up to the first HEX 03 (left to right). If it is a USER-FILE, the character to the left of the HEX 03 is the FILE-TYPE. Some system files have a HEX 02 here. This shows up under DISK-MGR as a circle with bars in it.

FILE-ATTRIBUTE This is where you need to break the hex into the bit patterns. This is what the system determines what files are printed under CATALOG. Deleted files still exist but not accessible from BASIC. You can BLOCK-READ then and dump them out. LOCKED files set on the first 3 protects.

START-BLOCK This will tell you what block to start reading from. Remember to do a HEX convert.

DATE (CREATE) This may contain values.

BLOCKS-LEFT This is the last filename entry. Its is marked as ATTRIBUTE (01) not a file. The START-BLOCK shows the next available block for storage. ALLOCATED-SIZE shows the number of available blocks left.

Interleave Chart

Type	Size	Heads	Tracks	Sectors/Track	Interleave
Standard 5.25"	160K	1	40	8	5
Double sided 5.25"	320K	2	40	8	5
Quad size 5.25"	640K	2	80	8	4
Double sided 3.5"	720K	2	80	9	4
High density 3.5"	1.44M	2	80	18	4

160K Disk Interleave Example

Track 0

0	200	400	600	800	A00	C00	E00
Block 0 low	Block 2 high	Block 1 low	Block 3 high	Block 2 low	Block 0 high	Block 3 low	Block 1 high
Sector 1	Sector 2	Sector 3	Sector 4	Sector 5	Sector 6	Sector 7	Sector 8

Track 1

1000	1200	1400	1600	1800	1A00	1C00	1E00
Block 4 low	Block 6 high	Block 5 low	Block 7 high	Block 6 low	Block 4 high	Block 7 low	Block 5 high

Track 2

2000	2200	2400	2600	2800	2A00	2C00	2E00
Block 8 low	Block 10 high	Block 9 low	Block 11 high	Block 10 low	Block 8 high	Block 11 low	Block 9 high

Track 3

3000	3200	3400	3600	3800	3A00	3C00	3E00
Block 12 low	Block 14 high	Block 13 low	Block 15 high	Block 14 low	Block 12 high	Block 15 low	Block 13 high
		directory	file			file	

Track 4

4000	4200	4400	4600	4800	4A00	4C00	4E00
Block 16 low	Block 18 high	Block 17 low	Block 19 high	Block 18 low	Block 16 high	Block 19 low	Block 17 high
file	file	file		file	file	file	file

5000	5200	5400	5600	5800	5A00	5C00	5E00
Block 20 low	Block 22 high	Block 21 low	Block 23 high	Block 22 low	Block 20 high	Block 23 low	Block 21 high

file		file		file	file		file

ASCII CHART

DEC	HEX	CHARACTER	MEANING	DEC	HEX	CHARACTER	MEANING
0	00	CONTROL-@		39	27	'	
1	01	CONTROL-A		40	28	(	
2	02	CONTROL-B		41	29	)	
3	03	CONTROL-C		42	2A	*	
4	04	CONTROL-D		43	2B	+	
5	05	CONTROL-E		44	2C	'	
6	06	CONTROL-F		45	2D	-	
7	07	CONTROL-G	BELL	46	2E	.	
8	08	CONTROL-H	BACKSPACE	47	2F	/	
9	09	CONTROL-I	HORIZ TAB	48	30	0	
10	0A	CONTROL-J	LINE FEED	49	31	1	
11	0B	CONTROL-K		50	32	2	
12	0C	CONTROL-L	CLEAR SCREEN	51	33	3	
13	0D	CONTROL-M	RETURN	52	34	4	
14	0E	CONTROL-N		53	35	5	
15	0F	CONTROL-O		54	36	6	
16	10	CONTROL-P	DUMP TO PRT	55	37	7	
17	11	CONTROL-Q		56	38	8	
18	12	CONTROL-R		57	39	9	
19	13	CONTROL-S	PAUSE	58	3A	:	
20	14	CONTROL-T		59	3B	;	
21	15	CONTROL-U		60	3C	<	
22	16	CONTROL-V		61	3D	=	
23	17	CONTROL-W		62	3E	>	
24	18	CONTROL-X		63	3F	?	
25	19	CONTROL-Y		64	40	@	
26	1A	CONTROL-Z		65	41	A	
27	1B	CONTROL-[	ESCAPE/WP	66	42	B	
28	1C	CONTROL-\		67	43	C	
29	1D	CONTROL-]		68	44	D	
30	1E	CONTROL-^		69	45	E	
31	1F	CONTROL-_		70	46	F	
32	20	SPACE		71	47	G	
33	21	!		72	48	H	
34	22	"		73	49	I	
35	23	#		74	4A	J	
36	24	$		75	4B	K	
37	25	%		76	4C	L	
38	26	&		77	4D	M	

DEC	HEX	CHARACTER	MEANING	DEC	HEX	CHARACTER	MEANING
78	4E	N		110	6E	n	
79	4F	O		111	6F	o	
80	50	P		112	70	p	
81	51	Q		113	71	q	
82	52	R		114	72	r	
83	53	S		115	73	s	
84	54	T		116	74	t	
85	55	U		117	75	u	
86	56	V		118	76	v	
87	57	W		119	77	w	
88	58	X		120	78	x	
89	59	Y		121	79	y	
90	5A	Z		122	7A	z	
91	5B	[		123	7B	{	
92	5C	\		124	7C	|	
93	5D	]		125	7D	}	
94	5E	^		126	7E	~	
95	5F	_		127	7F	DELETE	
96	60	'		128	80	HOME	
97	61	a		129	81	FUNCTION I	
98	62	b		130	82	FUNCTION II	
99	63	c		131	83	FUNCTION III	
100	64	d		132	84	FUNCTION IV	
101	65	e		133	85	FUNCTION V	
102	66	f		134	86	FUNCTION VI	
103	67	g		135	87		
104	68	h		136	88		
105	69	i		137	89	SHIFT FUNCTION 1	
106	6A	j		138	8A	SHIFT FUNCTION II	
107	6B	k		139	8B	SHIFT FUNCTION III	
108	6C	l		140	8C	SHIFT FUNCTION IV	
109	6D	m		141	8D	SHIFT FUNCTION V	

DEC	HEX	CHARACTER	MEANING
		SHIFT FUNCTION	
141	8D	V	
		SHIFT FUNCTION	
142	8E	VI	
143	8F		
144	90	WILDCARD	
145	91	UNDO	
146	92	COPY	
147	93	GET	
148	94	INSERT	
149	95	PRINT	
150	96	CLEAR	
151	97	DELETE	
152	98	SHIFT WILDCARD	
153	99	SHIFT UNDO	
154	9A	SHIFT MOVE	
155	9B	SHIFT STORE	
156	9C	SHIFT INSERT	
157	9D	SHIFT PRINT	
158	9E	SHIFT DELETE	
159	9F		
160	A0	ARROW UP	
161	A1	ARROW RIGHT	
162	A2	ARROW DOWN	
163	A3	ARROW LEFT	

COLOR PALETTE

0	Transparent	**8**	Medium Red
1	Black	**9**	Light Red
2	Medium Green	**10**	Dark Yellow
3	Light Green	**11**	Light Yellow
4	Dark Blue	**12**	Dark Green
5	Light Blue	**13**	Purple
6	Dark Red	**14**	Gray
7	Cyan	**15**	White

MEMORY BANK SWITCHES

The Z80 can only address 64K (2^{16}) memory locations but can change blocks of memory by a technique known as bank switching. The ADAM contains about 40K of ROM (Read Only Memory) with Smartwriter and two operating system, EOS and OS7 that can be switched into the upper or power 32K of memory space. Expansion RAM (if you have it) and game cartridge ROM can also be switched in and out of the active memory space. Memory is normally all RAM in BASIC, the OS having been copied from ROM to RAM.

The bank switch is an OUT 7F,x command, where the lower nibble of x selects the following options:

Lower 32 K	**D1**	**D0**	**Upper 32K**	**D3**	**D2**
Smartwriter, EOS	0	0	32K RAM	0	0
32K RAM	0	1	Expansion ROM	0	1
Expansion RAM	1	0	Expansion RAM	1	0
OS7+24K RAM	1	1	Cartridge ROM	1	1

For example, to select normal RAM for both the upper and lower blocks the number in binary is 0001, or 1. To select 32K of RAM on the bottom and cartridge ROM on the top, the number is 1101, or 13 (dec). A D1 and D) of 0 will access wither Smartwriter or the EOS ROM, depending upon another in/out command. Performing OUT 3F,2 before the OUT 7F,0 will access the EOS ROM. OUT 3F,0 causes OUT 7F,0 to access the Smartwriter ROM.