DOS/65 SYSTEM INTERFACE GUIDE

VERSION 2.1

180 Ridge Road
Cimarron, CO 81220

TABLE OF CONTENTS

SECTION 1 - INTRODUCTION

DOS/65 is a powerful and flexible operating system for the 65XX series of microprocessors. Most of its power is realized as a result of the ease with which user programs can use the console and file oriented I/O features of the PRIMITIVE EXECUTION MODULE (PEM) portion of the operating system. The flexibility on the other hand is realized through use of a software structure which places all of the hardware peculiar device control routines in a SYSTEM INTERFACE MODULE (SIM) which can be modified by the user to conform to his peculiar hardware and software environment. SIM allows definition both of the disk characteristics through use of a DISK CONTROL BLOCK (DCB) and the console characteristics through use of a CONSOLE DEFINITION BLOCK (CDB).

PEM and SIM are thus the two key interfaces in DOS/65 which the programmer must understand in order to effectively use DOS/65. Each of these two interfaces will be discussed in the following sections.

SECTION 2 - PRIMITIVE EXECUTION MODULE (PEM)

2.1 GENERAL CONCEPT

Each time PEM is entered it performs a single function and upon completion of that function executes an RTS. Thus, PEM is normally entered as a subroutine. The specific function performed is determined by the number contained in the X register of the 6502.

In addition to the contents of the X register, the value of the A, and in some cases also the Y, register upon entry determines not what is done but rather to what or with what the function specified by the X register is performed. Similarly, upon exit the A register contains the result if the result is a single byte quantity such as an ASCII character, or for 16 bit results the A and Y register together contain the result.

Figure 2-1 summarizes the register conditions at entry to PEM and upon return. Each command is discussed in the following sections.

2.2 CHARACTER ORIENTED I/O COMMANDS

Those commands which involve transfer of single ASCII characters are described in the following sections. The contents of the CONSOLE DEFINITION BLOCK referred to in this section are defined in Section 3.6.

2.2.1 X = 1 (READ CONSOLE INPUT WITH ECHO)

This command returns a single ASCII character in A from the console input device. If a null ($00) then the Z flag is set. If the character is a printable character ($20 through $7F) it will be sent to the console output. The CR ($0D), LF ($0A), and HT (ctl-i or $09) characters will also be output, however, the HT echo will be expanded using blanks ($20) to the next modulo-8 column. The actual character returned by PEM in this case will be the HT, not the blanks resulting from the expansion. No other control characters are echoed. While SIM is supposed to provide characters having the MSB set to zero, PEM neither checks for zero nor sets that bit to zero. If the LIST ECHO flag is set, the character (after expansion) will also be sent to the list device.

2.2.2 X = 2 (CONSOLE OUTPUT)

This command causes the single ASCII character in A to be sent to the console output device. All characters ($00 to $FF) will be output exactly as input to PEM except for HT

($09) which is expanded using blanks ($20) to the next modulo-8 column. Note that the MSB of the character being output is not checked for zero nor is it automatically set to zero. If the LIST ECHO flag is set, the character (after expansion) will also be sent to the list device.

Each time a character is sent to the console the READ CONSOLE STATUS routine in SIM is executed. If a key has been pressed it is read by PEM and saved for later use by a READ CONSOLE INPUT command (X = 1 or X = 6). However, if the key is a (ctl-s) PEM does not save the (ctl-s) but waits for a second key to be pressed. If that second key is a (ctl-c), a WARM BOOT is executed by jumping to SIM+3. All other keys are not saved but return control to the calling routine. This last feature allows the output to be "held" and thus is useful for viewing lengthy outputs on a non-permanent console output device such as a CRT.

	INPUT		OUTPUT	
REGISTER	SINGLE BYTE PARAMETER	TWO BYTE PARAMETER	SINGLE BYTE RESULT	TWO BYTE RESULT
A	Parameter (if any)	Low Byte	Result (if any)	Low Byte
Y	X	High Byte	X	High Byte
X	Function Number	Function Number	X	X
S	See note 2	See note 2	Unchanged	Unchanged
P	X	X	Z=1 IFF A=0 N==1 IFF A<0 D=0 See note 3	Z=1 IFF A=0 N==1 IFF A<0 D=0 See note 3

NOTES

1. X = “don’t care” or indeterminate.
2. While S upon exit from PEM is unchanged (except for the action of the RTS) its value at input should be large enough to ensure that no stack underflow results or that the stack intrudes into any page one reserved areas such as the default buffer. While it is estimated that PEM uses less than 32 bytes of stack space, the total which should be allowed depends upon the design of SIM. The 88 bytes available above the default buffer should be adequate for most user programs.
3. The setting of the I flag upon exit from PEM is purely a function of SIM. PEM itself does not alter that flag. All flags not noted (C, B, and V) have indeterminate values upon return.
4. This table is valid only for X greater than zero and less than or equal to the maximum value. For all values of X greater than the maximum allowable only the S and P output conditions are valid. For X equal to zero there is no "return".

Figure 2-1 PEM Register Usage

2.2.3 X = 3 (READ FROM READER)

This command causes a single ASCII character to be read from the reader device. As was the case for the console the MSB is neither checked nor altered by PEM.

2.2.4 X = 4 (WRITE TO PUNCH)

This command causes a single ASCII character to be sent to the punch device. Again

the MSB is neither checked nor altered by PEM.

2.2.5 X = 5 (WRITE TO LIST DEVICE)

Similar to the action of the previous output commands, this command causes a single ASCII character to be sent to the list device. The MSB is neither checked nor altered by PEM. All characters (including the HT) are output exactly as input to PEM.

2.2.6 X = 6 (READ CONSOLE INPUT WITHOUT ECHO)

This command functions exactly like the X = 1 command except that the character is not echoed either to the console output or list device.

2.2.7 X = 9 (PRINT BUFFER)

This command causes the ASCII string pointed to by the A (low) and Y (high) registers to be sent to the console output device. The string may be up to 256 characters long. Output will only be terminated when a "$" is encountered in the string or after the 256th character. As was the case for the single character output commands, the HT character will be expanded to a modulo-8 column. The output string will also be sent to the list device if the LIST ECHO flag is set.

2.2.8 X = 10 (READ BUFFER)

This command causes an entire line of data to be read from the console input device, stored in a buffer and echoed to the console output device. The read is terminated by entry of a carriage return (CR) at the console. Upon entry the A (low) and Y (high) registers point to the start of the buffer. The buffer is organized in a unique way. Upon entry the byte pointed to by A and Y contains the maximum number of characters in the buffer. That value is not changed during execution. Upon exit the second byte contains the number of characters in the buffer. The characters themselves begin at the third byte. A typical buffer at exit would be as shown in Figure 2-2.

POSITION	CONTENTS	COMMENTS
1	32	At entry contains maximum length (32 in this use). Is the byte pointed to by A & Y at entry
2	3	At exit contains number of characters input
3	I	First character
4	N	Second character
5	C	Last character

Figure 2-2 READ BUFFER

Several special characters cause specific buffer editing actions to be executed if entered from the console during execution of this command. Those characters and the

resulting action are:

(ctl-i) Horizontal Tab

As was the case for the single character commands, the echo consists of enough blanks to move the cursor to a modulo-8 column.

(ctl-r) Line Repeat

This character is not entered into the buffer nor is it echoed. It causes a (CR) to be sent to the console followed by enough FORWARD SPACE (SIM+57) characters to skip over any characters printed on the current line before the READ BUFFER command was executed. A CLEAR TO END OF LINE (SIM+56) character is then sent to the console followed by the complete contents of the buffer. All characters are output literally except control characters and the (ctl-i) which is expanded to a modulo-8 column using blanks. All other control characters are output as an INVERT ATTRIBUTES (SIM+59) character followed by the character resulting from "oring" the character with an ASCII @ and then followed by a NORMAL ATTRIBUTES (SIM+58) character. This command is most useful to check that buffer contents are correct or to see what the buffer contents are after character deletes.

(delete) Character Delete

This character is not entered into the buffer nor is it echoed. The (delete) causes the last character in the buffer to be deleted. The (delete) also causes one or more BACKSPACE (SIM+55), $20, BACKSPACE (SIM +55) sequences to be sent to the console to erase the deleted character.

(ctl-x) Line Cancel

This character is not entered into the buffer nor is it echoed. It causes a (CR) to be sent to the console followed by enough FORWARD SPACE (SIM+57) characters to skip over any characters printed on the current line before the READ BUFFER command was executed. A CLEAR TO END OF LINE (SIM+56) character is then sent to the console. It then causes the buffer to be emptied. This command is most useful if an input line is in error but is too long to be conveniently corrected using the (delete).

(ctl-e) Physical (CR) and (LF)

This character is not entered into the buffer nor is it echoed. It causes a (CR)(LF) combination to be sent to the console. This command is most useful with console devices having limited line lengths and no automatic end of line (CR)(LF) feature.

(ctl-p) Toggle LIST ECHO

This character is not entered into the buffer nor is it echoed. It causes the

LIST ECHO flag to be toggled (on to off or off to on).

(ctl-c) WARM BOOT

If this character is the first character in the buffer, it causes a WARM BOOT to be executed. If not the first character in a line, then it is handled as a normal character and is entered into the buffer.

2.2.9 X = 11 (CONSOLE READY)

This commands checks to see if a console input is waiting. If an input is ready A is set to a non-zero value and Z is cleared. If no character is ready then A is set to zero and Z is set. As was the case for the CONSOLE OUTPUT (X = 2) function, a (ctl-s) will "hold" execution until another character is entered. In this instance, the second key will cause a "not-ready" condition upon return. If the second key is a (ctl-c) a WARM BOOT will be executed.

2.2.10 X = 12 (READ LIST STATUS)

This commands checks to see if the list device can accept another output character. If a character can be accepted then A is set to a non-zero value and Z is cleared. If a character can not be accepted then A is set to zero and Z is set.

2.2.11 X = 30 (SET LIST ECHO STATUS)

The contents of the A register are stored in the LIST ECHO flag. If the MSB of that value is 1, then characters output using the CONSOLE OUTPUT (X = 2) function also will be sent to the list Device. If the MSB is a 0 then the characters will not be output to the list device. Note that the LIST ECHO flag does not affect output using the LIST OUTPUT (X = 5) function.

2.2.12 X = 31 (READ LIST ECHO STATUS)

Upon return, the A register contains the current value of the LIST ECHO flag. The N flag will reflect the value of the MSB. Thus if N = 1 (i.e., minus), then the list echo is enabled. If N = 0 (i.e., plus), then the list echo is disabled.

2.3 SYSTEM CONTROL COMMANDS

These commands control several general aspects of the system. These commands do not involve actual disk I/O operations.

2.3.1 X = 0 (WARM BOOT)

The warm boot command is unique in that it does not return to the calling program. Its

primary function is to reload CCM and PEM from drive A and then execute CCM. The default drive will remain unchanged. This command is most often used to reinitialize the system after a transient program has completed execution. The JMP at $100 will also cause a warm boot to be executed, however, that jump goes directly to SIM+3 without using PEM.

2.3.2 X = 7 (READ I/O STATUS)

This command returns the contents of location $106 in the A register. DOS/65 does not use this byte, thus it can be used by the user to hold an I/O device mapping vector or for any other purpose. The standard version of SIM does set this byte to zero when a cold boot is executed.

2.3.3 X = 8 (SET I/O STATUS)

This command stores the contents of the A register at location $106. As discussed above, neither this action nor the value of the contents of $106 are significant to DOS/65.

2.3.4 X = 32 (READ CLOCK)

This command calls the real time clock routine at SIM+48 (see section 3.5) and saves the three values returned by SIM in an internal PEM storage area. Upon exit from PEM, the A register will contain the low byte of the three byte real time clock counter and Y will be zero if a clock is present or will be 128 if no clock is present. If Y is 128 then the contents of the A register are meaningless.

2.3.5 X = 33 (READ HIGH CLOCK)

This command returns the middle byte of the real time clock in the A register and the high byte in Y. Since this command does not issue a new call to SIM it must be used only after a READ CLOCK (X = 32) command has been executed. If the last READ CLOCK (X=32) command returned with Y equal to 128 then the return values from this command are meaningless.

2.4 DISK I/O COMMANDS

These commands control the disk I/O operations of DOS/65. Two concepts are central to use of these commands. The first is that of a FILE CONTROL BLOCK (FCB), A FCB is a 33 byte region which is initialized by the user and then is used by PEM to control disk I/O. The function of each byte in the FCB is discussed in detail in the System Description and is presented in summary form in Figure 2-3.

Byte	Mnemonic	Explanation

0	D	DRIVE number, i.e., 0 through 7
1	N	Byte 1 is the first character of file NAME where full name is of the form NNNNNNNN.TTT Name is blank filled to byte 8 if actual name is less than eight characters long.
2	N	
3	N	
4	N	
5	N	
6	N	
7	N	
8	N	
9	T	Byte 9 is the first character of the file name extension or TYPE, i.e., the three characters after the ".". This is also blank filled if needed.
10	T	
11	T	
12	E	EXTENT
13		Not used
14		Not used
15	R	NUMBER of RECORDS in binary
16	B	BLOCK NUMBERS
17	B	
18	B	
19	B	
20	B	
21	B	
22	B	
23	B	
24	B	
25	B	
26	B	
27	B	
28	B	
29	B	
30	B	
31	B	
32	X	NEXT RECORD in binary

Figure 2-3 FCB Contents

Each disk has a directory which contains a user defined number of directory entries (normally 64 for most floppy diskette formats) numbered starting with 0. Each directory entry corresponds to one or more extents of a file. Each extent references up to 16K (K=1024) bytes as individual 128 byte records. The data is organized in IK, 2K, 4K, 8K or 16K byte blocks as a function of the DCB contents in SIM. The number of each block assigned to a directory entry is stored in that directory. Since $00 is not a legal block number for user files its appearance in a directory entry means that no block is assigned for that particular portion of the directory.

The aspects of the use of the FCB that should be remembered are:

- User must fill in

  DRIVE (Byte 0) $0 = default drive (1 to 8 selects designated drive minus 1)

  NAME (Bytes 1-8) Upper Case ASCII

  TYPE (Bytes 9-11)

  EXTENT (Byte 12) (0 to $1F)

  before searching for, opening, closing, deleting, or creating a file. Special actions for FCB use when renaming a file are discussed under that command. While not absolutely necessary, all unused bytes should be set to 0 before any functions are executed.

- DOS/65 fills in

  NUMBER RECORDS (Byte 15)

  BLOCK NUMBERS (Bytes 16-31)

  when a file is opened and maintains those values during read or write operations and updates the disk directory if a new extent is automatically opened or when a file is closed.

- User must set

  NEXT RECORD (Byte 32)

  before any read or write. Sequential sector read or writes do not require any user action as the NEXT RECORD will automatically be incremented after each read or write operation.

The second key concept is that of a sector buffer. This 128 byte block is the region in which DOS/65 does all disk reads and writes. This includes all directory reads required during operation as well as data reads or writes. The fact that two types of uses are made of the buffer is significant since the buffer contents may be altered by normal DOS/65 actions other than user commanded reads or writes.

2.4.1 X = 13 (INITIALIZE SYSTEM)

This command executes the following sequence:

- Clear log-in status so that no drive is considered on-line.

- Log-in drive A and select it as the default drive.

- Set drive A status to read/write

The effect of these actions is to clear the disk allocation maps and directory checksums which DOS/65 maintains but then to set them to match the disk currently in Drive A. Those maps and checksums are key elements in allocating disk space when writing new data and in verifying that the disk in the drive being addressed has not been changed since it was last logged-in.

2.4.2 X = 14 (SELECT DRIVE)

The drive specified by the contents of the A register ($00 through $07) is selected as the default drive. If that drive is not logged-in, it is logged-in and set to a read/write mode.

CAUTION

If the specified drive is already logged-in, no check is made during execution of this command to ensure that the disk has not been changed. (See Section 2.4.1.) However if any write operation is attempted to a drive whose directory checksum map entries do not match the checksums of the directory entries on the disk, then a PEM error message will be sent to the console and appropriate user action requested unless the DCB for that drive has disabled directory checksum verification. Any read operation to such a drive will be allowed.

2.4.3 X = 15 (OPEN FILE)

The file matching the DRIVE, NAME, TYPE and EXTENT fields of the FCB pointed to by A (low) and Y (high) will be initialized by DOS/65. If the file is a valid file and is successfully opened, the A register upon return will contain the directory number modulo 4 (0 to 3) of the file and N will be O. If the file is not successfully opened, the value returned in A will be 255 ($FF) and N will be 1. This command will fill in the NUMBER RECORDS byte and the BLOCK NUMBER FIELDS in the FCB.

2.4.4 X = 16 (CLOSE FILE)

The file matching the DRIVE, NAME, TYPE, AND EXTENT fields of the FCB pointed to

by A (low) and Y (high) will be closed by DOS/65.

CAUTION

The file must be "open" for this command to execute properly.

This command updates the directory entry on the disk and if successful returns the directory number modulo 4 (0 to 3) in the A register and clears N to 0. If not successful, the contents of the A register upon return will be 255 ($FF) and N will be set.

2.4.5 X = 17 (SEARCH FIRST)

The file extent matching the DRIVE, NAME, TYPE, and EXTENT fields of the FCB pointed to by A (low) and Y (high) will be searched for in the directory. If found, the directory number modulo 4 (0 to 3) will be returned in the A register and N cleared to 0. A '?' in any position of the NAME and TYPE fields will match any character in the directory. The contents of the buffer will contain the directory entry beginning at location:

BUFFER + ((A) *32)

Example : IF (BUFFER = $128) and (A = 1)
Then directory begins at $148

If not found, the returned value in A will be 255 ($FF) and N will be set to 1.

2.4.6 X = 18 (SEARCH NEXT)

The next file extent matching the DRIVE, NAME, TYPE, and EXTENT fields of the FCB pointed to by A (low) and Y (high) will be searched for. If found, the directory number modulo 4 (0 to 3) will be returned in A and N will be cleared. As discussed above, the directory is located in the BUFFER at a position determined by the directory number. If not found, A will be set to 255 ($FF) and N will be set to a 1.

NOTE

This command is only guaranteed to be meaningful if the last previous disk I/O command was X = 17 (SEARCH FIRST).

2.4.7 X = 19 (DELETE FILE)

The file matching the DRIVE, NAME, and TYPE fields of the FCB pointed to by A (low) and Y (high) will be deleted if it exists. The return value from the command will always be 255 ($FF) and N will always be set.

2.4.8 X = 20 (READ RECORD)

The record specified by the NEXT RECORD field of the FCB pointed to by A (low) and Y (high) will be read from the disk into the current BUFFER. The file must be open. The following return codes are used to signify the results of the read operation.

A Value	Meaning for READ
0	Read successful
1	Physical end of file
2	Attempt to read data from an unwritten block. This return will usually only occur when the user is performing random reads from a file and the block entry specified by the value of NEXT RECORD is empty, i.e., $00
255	Error

The NEXT RECORD field will be automatically incremented by DOS/65 after the read and the next extent opened if necessary.

Random reads will require the user to open and close the proper extent using explicit open and close commands. The following equations are used to calculate the extent and record number of the nth random record:

EXTENT = INTEGER.PART.OF(n/128)
RECORD = n-(EXTENT*128)

2.4.9 X = 21 (WRITE RECORD)

The record specified by the NEXT RECORD field of the FCB pointed to by A (low) and Y (high) will be written to the disk from the current BUFFER. The file must be open. The following return codes are used to signify the results of the write operation where MAX is as defined in section 2.4.8.

A Value	Meaning for WRITE
0	Write successful
1	Extending error, i.e., next record is > 127
2	Disk full
255	Error

As was the case for the READ RECORD function, the NEXT RECORD field will be automatically incremented after the write and if necessary a new extent will be automatically opened.

CAUTION

If a new extent is automatically opened by DOS/65 after a write

operation, the contents of the write buffer will be destroyed.

2.4.10 X = 22 (CREATE FILE)

The file extent matching the DRIVE, NAME, TYPE and EXTENT fields of the FCB pointed to by A (low) and Y (high) will be created and the directory marked as empty. If successful, the directory number modulo 4 (0 to 3) will be returned in the A register and N cleared to 0. If the attempt is unsuccessful, the A register will be set to 255 and N set to 1.

CAUTION

Creation of a file using a FCB which contains illegal characters or lower case alphabetic characters could result in a file which cannot be executed or manipulated using CCM.

2.4.11 X = 23 (RENAME FILE)

This command is a unique command in that the normal FCB format is modified. For this command the file corresponding to the DRIVE, NAME, and TYPE fields of the FCB pointed to by A (low) and Y (high) is renamed to correspond to the NAME and TYPE contained at location FCB+16. If the following FCB were pointed to by A and Y upon execution of this command, the file OLDNAME.ASM would be renamed to NEWFILE.BAK.

```
$0 O L D N A M E $20 A S M $0 $0 $0 $0

$0 N E W F I L E $20 B A K $0 $0 $0 $0
```

CAUTION

If the old file name or type portions of the FCB contain a ?, all files which match the resulting AFN will be changed to match the new file name and type . The consequences of that action could include having identical names. For example, FILE.A and FILE.B would be renamed to FILE.A if the following FCB were used:

```
$0 F I L E $20 $20 $20 $20 ? $20 $20 $0 $0 $0 $0

$0 F I L E $20 $20 $20 $20 A $20 $20 $0 $0 $0 $0
```

2.4.12 X = 24 (READ LOG-IN STATUS)

This command returns the drive log-in status byte in the A register. Bit 0 corresponds to drive A, bit 1 to B, bit 2 to C, and so on up through bit 7 which corresponds to H. If a given bit is zero, the corresponding drive is not logged-in, if a one the drive is logged-in.

CAUTION

If the value of a bit is a 1, it does not mean that the disk in the corresponding drive has not been changed. For additional details see sections 2.4.1 and 2.4.2.

2.4.13 X = 25 (READ CURRENT DRIVE)

This command returns in the A register the drive number (0 = A, 1 = B, 2 = C,..., or 7 = H) corresponding to the current default drive.

2.4.14 X = 26 (SET BUFFER ADDRESS)

This command sets the sector buffer starting address to the value in the A (low) and Y (high) registers. Caution must be exercised in using this command to ensure that portions of DOS/65 (PEM and SIM especially) are not destroyed by incorrect placement of the buffer.

2.4.15 X = 27 (READ ALLOCATION VECTOR)

This command returns in the A (low) and Y (high) registers the address of the beginning of the block allocation map for the default drive. DOS/65 maintains a map for each drive which indicates the allocation status of each block. The blocks are numbered from 0 to MAXBLK and are mapped into the following bit and byte of the map.

BYTE = INTEGER.PART.OF(BLOCK/8)

BIT = 7-(BLOCK-(BYTE*8))

NOTE

Directory blocks will always show as allocated.

CAUTION

Alteration of the contents of the map could result in over writing of the contents of the affected blocks.

2.4.16 X = 28 (SET READ/WRITE STATUS)

This command causes the byte contained in the A register to be transferred to the read/write status byte maintained by PEM. Bit 0 corresponds to drive A, bit 1 to B, bit 2 to C, and so on up through bit 7 which corresponds to H. If the value of the corresponding bit is a 0 then the corresponding drive will be set to a read/write mode, if a 1; then the drive is set to a read only mode.

2.4.17 X = 29 (READ READ/WRITE STATUS)

This command returns the read/write status byte in the A register. The value of each bit is determined as discussed in section 2.4.16.

2.4.18 X = 34 (READ DCB ADDRESS)

This command returns in the A (low) and Y (high) registers the address of the DCB for the default drive.

2.4.19 X = 35 (TRANSLATE SECTOR)

This command calls the TRANSLATE SECTOR (SIM+51) routine in SIM using the values in the A (low) and Y (high) registers as the logical sector number. This command then returns in the A (low) and Y (high) registers the physical sector calculated by SIM for the default drive.

SECTION 3 – SYSTEM INTERFACE MODULE (SIM)

3.1 GENERAL CONCEPT

Unlike PEM which executes a function determined by the contents of the X register, SIM executes eighteen different functions based upon the address used to enter SIM, provides a block of data that defines the users console characteristics (the CONSOLE DEFINITION BLOCK or CDB), provides a block of data which defines the characteristics of each disk in the system (the DISK CONTROL BLOCK or DCB), and contains the disk allocation and checksum maps maintained by PEM. Figure 3-1 lists the function entry points. Each function, the DCB, the CDB, and the allocation and checksum maps are described in the following sections.

The X register has no meaning upon entry to SIM and except for the READ CLOCK function also has no meaning upon exit from SIM. The A register is the primary SIM input data register except for those functions requiring a sixteen bit parameter for which the A register holds the low byte of the parameter and the high byte is in the Y register. The A register is also the primary SIM output data register but just as was the case for inputs, both the A and Y registers are used for sixteen bit parameters.

During execution of all commands SIM may use all CPU registers and the stack. Exit from SIM is accomplished by execution of a RTS (except for COLD and WARM BOOTS) hence each function operates like a subroutine. For those functions which require data to be returned, the only thing that SIM must guarantee is the value in the data register(s) since the CPU flags are not checked by PEM or any other calling program but rather the value in the register(s) is checked. Thus, for example, if a return value in A should be 0 to indicate that no error occurred, PEM does not rely upon the Z flag being set but will test A to see if its contents are zero.

3.2 SYSTEM INITIALIZATION FUNCTIONS

These two functions do not return to the calling program but instead set up the system and then execute CCM.

3.2.1 EXECUTE COLD BOOT INITIALIZATION (SIM)

The function is executed by BOOT after CCM, PEM, and SIM are loaded into memory by BOOT. Its primary function is to set-up the JMP's at $100 to SIM+3 and at $103 to PEM, to set the buffer to the default location ($128), and to set the default drive as drive A. It must also initialize the stack, CPU flags (particularly I), and the system I/O devices.

The exit from this function is a JMP to CCM with the number corresponding to the default drive (normally 0 for drive A) in the A register. Note that while the SIM listing shows an elaborate process of prompts and user inputs which allows the user to specify the number of drives, this is not necessary since the number of drives could be included in the assembly code. The advantage of having a user input is that SIM does not have to change if the number of drives changes.

3.2.2 EXECUTE WARM BOOT (SIM+3)

This function which is normally entered either via the JMP at $100 or via execution of the WARM BOOT (X = 0) command in PEM, must accomplish the following steps:

Set Interrupt Vector/JMP (if used)
Set Stack and CPU Flags
Read CCM and PEM from Drive A
Set up JMPs at $100 and $103
Set Buffer to Default ($128)
Jump to CCM with Default Drive Number in A

It is important to note that this function does not read SIM into memory from the disk and does not initialize the I/O devices.

Entry Address	Function
SIM	EXECUTE COLD BOOT
SIM+3	EXECUTE WARM BOOT
SIM+6	READ CONSOLE STATUS
SIM+9	READ FROM CONSOLE
SIM+12	WRITE TO CONSOLE
SIM+15	WRITE TO LIST
SIM+18	WRITE TO PUNCH
SIM+21	READ FROM READER
SIM+24	HOME SELECTED DRIVE
SIM+27	SELECT DRIVE
SIM+30	SET TRACK
SIM+33	SET SECTOR
SIM+36	SET BUFFER ADDRESS
SIM+39	READ SECTOR
SIM+42	WRITE SECTOR
SIM+45	READ LIST STATUS
SIM+48	READ CLOCK
SIM+51	TRANSLATE SECTOR

Figure 3-1 SIM Functions

3.3 CHARACTER I/O FUNCTIONS

The following functions handle all single character I/O operations.

3.3.1 READ CONSOLE STATUS (SIM+6)

This function tests the console input device and if an input is ready then a non-zero byte is returned in A. If no input is ready the returned value in A is set to zero. This function does not actually read the input.

3.3.2 READ FROM CONSOLE (SIM+9)

This function reads a single ASCII character from the console input device. If no input is ready, this routine must wait until an input is ready. The MSB of the character (Bit 7) is set to zero and the character is returned in the A register. This character must not be echoed by SIM or by the software or hardware called by SIM.

3.3.2 WRITE TO CONSOLE (SIM+12)

This function writes the single ASCII character in the A register to the console output device. The A register contents need not be preserved.

CAUTION

PEM does not filter out any control characters except for HT ($9 or ctl-i) and all characters whose MSB is 1 are also not filtered. SIM or the routines called by SIM must filter out any potentially dangerous characters.

3.3.4 WRITE TO LIST (SIM+15)

This function writes the single ASCII character in the A register to the list device. This function should behave exactly like the WRITE TO CONSOLE function at SIM+12. The one difference is that if this routine is called from the PEM WRITE TO LIST DEVICE (X = 5) routine, then the HT ($9 or ctl-i) character will not have been expanded. If on the other hand this routine has been called because the LIST ECHO flag was set and characters are being output from the PEM CONSOLE OUTPUT (X = 2) routine, then the HT characters will have been expanded prior to the call to WRITE TO LIST (SIM +15). While standard DOS/65 programs do not require that this routine include a HT expansion routine, it is recommended that such a capability be included.

3.3.5 WRITE TO PUNCH (SIM+18)

This routine transfers a single byte to the users punch device. Note that this may actually be almost any device but is usually a serial output device such as a cassette

recorder or a modem. Note that no standard DOS/65 program uses this entry and PEM does not alter the byte passed through it to this SIM function.

3.3.6 READ FROM READER (SIM+21)

This routine returns a single byte to PEM in the A register. Like the punch output routine, this function is not used in any standard DOS/65 programs. PEM also does not alter the byte when returned to user through PEM.

3.3.7 READ LIST STATUS (SIM+45)

This routine returns a zero if the list device can not accept another input and a non-zero value if it can accept another input. This routine is intended for use in such programs as spoolers where it is desired to output data to the list device but not to suffer the delays possible if the WRITE TO LIST DEVICE routine is called and then must wait for the device to be ready to accept a character. This entry is not used by CCM or the standard system transients.

3.4 DISK I/O AND CONTROL FUNCTIONS

The following functions control the disk subsystem in response to PEM file oriented commands.

3.4.1 HOME SELECTED DRIVE (SIM+24)

This routine causes the currently selected drive to immediately move to the home position (Track 0). This routine has no return value requirements.

3.4.2 SELECT DRIVE (SIM+27)

This routine sets the drive for all following operations to the value of the A register (0 for drive A, 1 for B, 2 for C, and so on up to 7 for H). It is recommended that the drive not be physically selected until a home, read, or write command is received by SIM. This routine must return a value that is the address of the DCB for the selected drive. This address must use the standard DOS/65 convention which places the low byte of the address in A and the high byte of the address in Y. If the selected drive is not present in the system both A and Y should be set to zero.

The following example shows one way to code SELECT DRIVE for a system having two drives:

```
SELDSK    CMP       #2          see if too big
          BCS       NTVALD      it is!!!
          STA       DISKNO      save for later use
```

```
            ASL       A           multiply by two
            TAX                   make an index
            LDA       DCBTBL,X    get address low
            LDY       DCBTBL+1,X  and high
            RTS
NTVALD      LDA       #0          set ay to 0
            TAY
            RTS
DCBTBL      .WOR      DCB0,DCB1   addresses of dcbs
```

3.4.3 SET TRACK (SIM+30)

This routine should set the track for all following disk I/O operation to the value in the A (low) and Y (high) registers. This value will range from 0 to a number determined by PEM as a function of the DCB contents. This routine has no return value requirements.

3.4.4 SET SECTOR (SIM+33)

This routine sets the sector number (logical record number) for all following disk I/O operations to the value in the A (low) and Y (high) registers. This value is only a physical sector number if the physical sector size is one record (128 bytes) long. The range of logical record numbers is determined by the DCB for the selected drive and will range from 0 to records.per.track-1 (NRECRD-1). The value passed to this function is also affected by the action of the TRANSLATE SECTOR function at SIM+51. For example, an eight inch single sided, single density disk uses a translation table that converts the logical sector number to a physical sector number in the range of 1 to NRECRD while also incorporating interleave for performance enhancement.

CAUTION

Some transients are designed only for the IBM standard single density format and use direct physical, 128 byte sector I/O. If general use is planned they may need to be converted to use of the full capability of SIM including the de-blocking described in APPENDIX E.

3.4.5 SET BUFFER ADDRESS (SIM+36)

This routine sets the sector buffer starting address for all following disk I/O operations to the value in A (low) register and Y (high) register. This routine has no return value requirements.

3.4.6 READ SECTOR (SIM+39)

This routine reads a single sector (128 bytes) from the current drive, track and sector and transfers it to the buffer beginning at the current buffer address. None of these

parameters (drive, track, sector, or buffer address) are affected by this operation. If this function is successful, then the value of the A register upon return to PEM should be set to 0. If the function is not successful, then the value of the A register upon return to PEM should be set to a non-zero value.

3.4.7 WRITE SECTOR (SIM+42)

This routine writes a single sector (128 bytes) to the current drive, track, and sector from the buffer beginning at the current buffer address. None of these parameters (drive, track, sector, or buffer address) are affected by this operation. If this function is successful, then the value of the A register upon return to PEM should be set to 0. If the function is not successful, then the value of the A register upon return to PEM should be set to a non-zero value.

At entry to this function, PEM sets A equal to one of the following values:

0	if the write is to an allocated record
1	if the write is to a directory record
2	if the write is to an unallocated record

Use of these values described in APPENDIX E – DEBLOCKING.

3.4.8 TRANSLATE SECTOR (SIM+51)

This routine translates a logical record number in the A (low) and Y (high) registers into a physical sector number in the A (low) and Y (high) registers for the currently selected drive. For many floppy disk formats this logical sector number can be used as an index into a translation table as shown in the following examples:

```
;translate - all drives identical & <=255 sectors per track
XLATE     TAX                       A to index & ignore Y
          LDA       XLTTBL,X        get table value
          RTS                       return with physical

;translate - two different drives & <=255 sectors per track
XLATE     TAX                       A to index & ignore Y
          LDA       NXTDRV          get drive number
          BNE       SETS2           branch if not drive 0
          LDA       XLTT0,X         else get value
          RTS                       exit
SETS2     LDA       XLTT1,X         get value
          RTS                       and return with it
```

NOTE

This function is normally only used when the physical sector is one record (128 bytes) long. For disks having physical records > 128 bytes long this

function should return the logical record number unaltered.

3.5 READ CLOCK (SIM+48)

This entry provides a standard means to return a real-time clock value to transients. The return time should be in 1/60ths of a second for a 24 hour clock. Thus the return value should be between $000000 and $4F19FF (5183999) inclusive. Since the return value requires more than two bytes, the normal DOS/65 parameter passing standards have been extended in this case to use the X register. Thus for example a time of $4F19FF would be returned as

A=$FF Y=$19 X=$4F

If the user does not have a real-time clock then the most significant bit of X should be set to 1. In such a case the code at the entry point could be

```
LDX       #128
RTS
```

An example of how to convert this capability to a clock time in a BASIC-E/65 subroutine is as follows:

```
tea=512
sim=tea+12
x.reg=tea+17
poke x.reg,48
time=call(sim)
if peek(x.reg) > 127\
     then\
          print "NO CLOCK!":\
          return
time=peek(x.reg)*65536+time
hours=int(time/216000)
time=time-(hours*216000)
minutes=int(time/3600)
time=time-(minutes*3600)
seconds=int(time/60)
print hours;":";minutes;":";seconds
return
```

Neither CCM, PEM nor any standard transient uses this entry, however future versions may use the real time clock for such functions as calculation of assembly or compile time and rate. In that case the clock will only be used if the X value indicates that the clock is actually present.

3.6 CONSOLE DEFINITION BLOCK (SIM+54)

The man-machine interface provided by the console uses a block of data which defines the characteristics of the users console output device. That block of data begins at SIM +54 and is defined by the following assembly language (the data shown is for example only). Each individual entry is discussed in Section 3.6.1.

```
*          =          sim+54
           .byt       0          scratch byte
           .byt       8          backspace
           .byt       5          clear to end of line
           .byt       $1c        forward space
           .byt       $f         normal attributes
           .byt       $e         invert attributes
           .byt       24         lines per screen
           .byt       80         char per line
           .byt       $c         formfeed
           .byt       1          home
           .byt       2          clear to end of screen
```

NOTE

If the console actually requires a multiple character sequence to accomplish the function described then the single character in this block should function as a "trigger" which causes the full sequence to be generated by the console output routine in SIM.

3.6.1 DATA DEFINITIONS

The following descriptive terms are used in many of the data descriptions and are defined here for clarity.

Control Character

One of the ASCII characters having a hexadecimal value of from $00 through $1F. Since the CR ($0D) and LF ($0A) are already used as output characters by DOS/65 they can not normally be used as to accomplish any of the special functions defined below. They can be used if the resulting action is an acceptable substitute for a function which is not otherwise supported by the users console.

Non-Printing

This term means that the character does not result in placement of a character on the screen and does not cause the cursor to move except as specified.

3.6.1.1 SCRATCH (SIM+54)

This byte is used by PEM as a scratch byte. It need not be initialized to any specific

value by SIM but must not be altered by SIM or any transient.

3.6.1.2 BACKSPACE (SIM+55)
This byte should be the non-printing, control character which when sent to the console causes the cursor to move one position to the left. If the cursor is at the left edge of the screen when this character is sent to the console no action should be taken. Other than the change in display attributes (e.g., blink, underline, reverse) associated with the cursor, the character at the old and new cursor positions should not be altered as a result of this character.

3.6.1.3 CLEAR TO END OF LINE (SIM+56)
This byte should be the non-printing, control character which when sent to the console will cause every position after the cursor and in the same line as the cursor and the position at the cursor to be cleared. If the console does not support such a function a linefeed ($0A) may be substituted. In this case some line editing inputs (ctl-r and ctl-x) will cause the console output to scroll up to display a blank line rather than clearing the current line as the CLEAR TO END OF LINE character should.

3.6.1.4 FORWARD SPACE (SIM+57)
This byte should be the non-printing, control character which when sent to the console causes the cursor to move one position to the right. If the cursor is at the right edge of the screen when this character is sent to the console no action should be taken. Other than the change in display attributes (e.g., blink, underline, reverse) associated with the cursor, the character at the old and new cursor positions should not be altered as a result of this character. The action of this character is the reverse of the BACKSPACE character. If this function is not supported by the console an ASCII space ($20) may be substituted. The only adverse effect is that input prompts (e.g., the normal CCM prompt) will be destroyed when a ctl-r or ctl-x is used during editing.

3.6.1.5 NORMAL ATTRIBUTES (SIM+58)
This byte should be the non-printing, control character which when sent to the console causes all following characters sent to the console to be displayed in "normal" mode. The definition of "normal" is up to the user but would typically mean normal video, not underlined, not blinking or similar attributes. If this function and the INVERT ATTRIBUTES function are not supported by the console then this character should be set to the NULL ($00) and the output routine in SIM should ignore the NULL.

3.6.1.6 INVERT ATTRIBUTES (SIM+59)
This byte should be the non-printing, control character which when sent to the console causes all following characters sent to the console to be displayed in "inverted" mode. The definition of "inverted" is up to the user but would typically mean inverted video, underline, blinking or similar attributes. If this function and the NORMAL ATTRIBUTES function are not supported by the console then this byte should be set to an ASCII "Up Arrow" ($5E).

3.6.1.7 LINES PER SCREEN (SIM+60)

This byte should be the number of lines in one full screen of data on the console. Typical values are 16 and 24. Hardcopy terminals should use 24 in order to preclude excess output lengths for programs which use this parameter to determine the allowable number of lines to output.

3.6.1.8 CHARACTERS PER LINE (SIM+61)

This byte should be the number of characters in a single line on the console output device. Typical values are 20, 32, 40, 64, 72, 80.

3.6.1.9 FORMFEED (SIM+62)

This byte should be the non-printing, control character which when sent to the console will cause the screen to be cleared and the cursor to be positioned at the upper left hand (home) position.

3.6.1.10 HOME (SIM+63)

This byte should be the non-printing, control character which when sent to the console will cause the cursor to be positioned at the upper left hand (home) position. Other than the change in display attributes (e.g., blink, underline, reverse) associated with the cursor, the character at the old and new cursor positions should not be altered as a result of this character.

3.6.1.11 CLEAR TO END OF SCREEN (SIM+64)

This byte should be the non-printing, control character which when sent to the console causes all positions after the cursor and the position at the cursor to be cleared. The cursor location should not be altered as a result of this character.

3.6.2 USAGE

Version 2.1 of DOS/65 does not use all of the data in the console definition block. The following table shows which elements of the block are used by each program. Many other transients (e.g., RUN.COM) use the buffered input capabilities of PEM and thus indirectly use the data in the console definition block.

	PEM Input	PEM Output	CCM	EDIT	DEBUG	ASM
BACKSPACE	X	X		X		
CLEAR TO EOL	X			X		
FORWARD SPACE	X	X				
NORMAL ATTRIBUTES	X		X	X		
INVERT ATTRIBUTES	X		X	X		
LINES PER SCREEN					X	
CHAR PER LINE			X		X	
FORMFEED						X
HOME						
CLEAR TO EOS						

NOTE

The Normal and Invert Attributes characters are only used to echo control characters.

3.6.3 STANDARD CHARACTERS

The CR ($0D), LF ($0A) and DELETE ($7F) are used extensively by DOS/65 and are assumed to result in the following action when sent to the console:

CR

Places cursor at left edge of screen but does not scroll the screen or in any other way cause the cursor to move from the original line. No characters are altered by the CR other than the attributes associated with the cursor itself.

LF

Moves cursor to the next line. If the cursor is already on the last line of the screen the screen should scroll up and clear the new bottom line. The position of the cursor in the line should not be altered as a result of the action of this character.

DELETE

Prints some character on the screen. If the users console does not print a character when it receives a DELETE, then the following code should be inserted in the WRITE TO CONSOLE routine in SIM:

```
CMP       #$7F
BNE       *+4
LDA       #' '
```

3.7 DCB CONTENTS

Version 2.1 allows a wide variety of disk formats to be used. In order to implement that flexibility the user must define several parameters and place them in a DISK CONTROL BLOCK (DCB) organized as follows (numbers shown are example only):

```
MAXBLK    .WOR 242     maximum block number
NRECRD    .WOR 26      number of 128 byte records/track
NSYSTR    .WOR 2       number of system tracks
BLKSCD    .BYT 0       allocation block size code
MAXDIR    .WOR 63      maximum directory number
ALCMAP    .WOR $F345   address of allocation map
CHKFLG    .BYT 0       checksum flag
CHKMAP    .WOR $F365   address of checksum map
```

Note that each drive must have a unique DCB. Each of the DCB parameters is discussed in the following sections:

NOTE

All references to records refer to 128 byte, DOS/65 sectors or records. If a user wishes to use hardware sectors of some other length all blocking/deblocking is the users responsibility as described in APPENDIX E. All data transfers to and from PEM or any other program which calls SIM must be in the form of 128 byte records. The buffer length in DOS/65 is set at 128 therefore that buffer can only be used as the hardware sector buffer when the hardware sector length is 128. Any larger sector size will require use of a separate buffer in SIM and transfer of data to/from the current DOS/65 buffer location as logical records of 128 bytes. For example a Kaypro IV double-sided, double density 5.25 inch disk would appear to DOS/65 Version 2.1 to consist of 40 records per track. In that case the users SIM must handle the blocking & unblocking associated with handling the 512 byte hardware sectors of the disk and getting the correct 128 bytes out of the hardware sector or putting the 128 bytes back into the right position.

CAUTION

While Version 2 allows each disk in the system to be different and even allows the disk characteristics to be dynamically altered, once a given disk (i.e., the magnetic media) is used with a specific DCB parameter set it can only be used with that same DCB parameter set.

3.7.1 NUMBER OF SYSTEM TRACKS (NSYSTR)

For disks which must contain the operating system in a form which can be created and loaded using the normal approach or which are to be used for interchange with other users, sufficient space must be reserved on the disk for BOOT, CCM, PEM and SIM. This space must be allocated as whole tracks. It is possible to create a data-only disk by setting NSYSTR to zero.

As a minimum of 5504 bytes are required for BOOT, CCM, PEM and SIM, the number of system tracks which must be allocated can be calculated as follows:

bytes.per.track=(sectors.per.track)*128

system.tracks=5504/(bytes.per.track)

EXAMPLE:

sectors.per.track=18

bytes.per.track=18*128=2304
system.tracks=5504/2304=2.4

therefore

NSYSTR=3

3.7.2 NUMBER OF RECORDS (NRECRD)

The NRECRD parameter is the number of 128 byte, DOS/65 records in a track. No calculation is usually needed to determine this number except in cases where the hardware sector size is larger than 128 bytes. In that case

NRECRD=(number.of.hardware.sectors)*(hardware.sector.size)/128

3.7.3 ALLOCATION BLOCK SIZE CODE (BLKSCD)

Disk space is allocated in blocks as a function of the block size code (BLKSCD). The value of BLKSCD for each of the allowable allocation block sizes is shown in the following table:

BLOCK SIZE (K=1024)	BLKSCD
1K	0
2K	1
4K	2
8K	3
16K	4

3.7.4 MAXIMUM BLOCK NUMBER (MAXBLK)

The disk allocation blocks on a disk are numbered from zero through the value of the parameter MAXBLK. This parameter is calculated as follows:

data.bytes=(total.tracks-system.tracks)*(sectors.per.track)*128

maximum.block.number=integer.part.of(data.bytes/block.size)-1

EXAMPLE:

total.tracks=40
system.tracks=2
sectors.per.track=26
data.bytes=(40-2)*(26)*128=126464
block.size=1024
number.of.blocks=int(126464/1024)=123

therefore

MAXBLK=122

NOTE

If the maximum block number is greater than 65535 with the selected block.size then the block size must be increased and MAXBLK recalculated or MAXBLK must be set to 65535. In the later case some disk capacity will be wasted. If the maximum block number is greater than 65535 with a block size of 16K then MAXBLK must be set to 65535. Some disk capacity will be wasted in that case.

3.7.5 MAXIMUM DIRECTORY NUMBER (MAXDIR)

The first few records of the data area on a disk are used for the directory. MAXDIR defines the maximum directory number for the disk and is one less than the number of directory entries allowed for the drive. Note that since 4 directory entries are stored in each DOS/65 record the number of records devoted to the directory can be calculated as follows:

number.directory.records=integer.part.of(MAXDIR/4)+1

While all SIF formats use 64 directory entries and hence use a MAXDIR of 63, the user can use other values for internal use as desired. Disks having a large capacity will be most likely to require additional directory space.

3.7.6 ADDRESS OF ALLOCATION MAP (ALCMAP)

SIM must contain a space devoted to the disk allocation map for each drive. Each drive must have its own unique space. ALCMAP is the address of the first byte of that space. The length of that space is calculated as follows:

alloc.map.length=integer.part.of(MAXBLK/8)+1

EXAMPLE:

MAXBLK=345
alloc.map.length=(345/8)+1=43+1=44

In this example the space in SIM would be reserved with a line in the SIM assembly source that looked like:

```
AMAP0       *=      *+44
```

The entry in the DCB for ALCMAP would be:

```
.wor    AMAP0
```

3.7.7 CHECK FLAG (CHKFLG)

As discussed in section 2.4.1, DOS/65 maintains a set of checksums for each directory record on each disk. Whenever a write operation is performed on that disk, the checksums stored by PEM are compared to the checksums on the disk for which the write is being attempted. If those checksums do not match, the disk is marked as READ ONLY (R/O) and the write is aborted. While this is a good check for removable disks, it is not necessary for non-removable disks and would slow down I/O operations. If CHKFLG is zero, then checksums will be tested. If the user desires to eliminate testing of checksums then CHKFLG should be set to 128 ($80).

3.7.8 ADDRESS OF CHECKSUM MAP (CHKMAP)

As discussed in section 2.4.1 and 3.7.7, DOS/65 maintains a set of checksums for each directory record on each disk. As was the case for the allocation map, the set of checksums are actually located in SIM but are maintained by PEM. A unique space must be allocated for each drive. The length of that space is calculated as follows:

check.map.length=integer.part.of(MAXDIR)/4+1

EXAMPLE:

MAXDIR=63
check.map.length=(63/4)+1=15+1=16

In this example the space in SIM would be reserved with a line in the SIM assembly source that looked like:

```
CMAP0       *=      *+16
```

The entry in the DCB for CHKMAP would be:

```
.wor    CMAP0
```

APPENDIX A - SYSTEM MODULE LOCATION ON DISK

MODULE	LENGTH BYTES	SECTORS	FIRST TRACK/SECTOR	LAST TRACK/SECTOR
SIF A & D				
BOOT	128	1	0/1	0/1
CCM	2048	16	0/2	0/17
PEM	3072	24	0/18	1/15
SIM	256 & up	2 & up	1/16	
SIF B & C				
BOOT	128	1	0/1	0/1
CCM	2048	16	0/2	0/17
PEM	3072	24	0/18	2/5
SIM	256 & up	2 & up	2/6	
SIF E				
BOOT	128	1	0/1	0/1
CCM	2048	16	0/2	1/1
PEM	3072	24	1/2	2/9
SIM	256 & up	2 & up	2/10	
SIF F & G				
BOOT	128	1	0/1	0/1
CCM	2048	16	0/2	0/17
PEM	3072	24	0/18	1/11
SIM	256 & up	2 & up	1/12	

APPENDIX B - DOS/65 MEMORY USAGE

B.1 DOS/65 PECULIAR LOCATIONS

As illustrated in Figure B-1, some DOS/65 locations vary as a function of the version while others are fixed. Both of these categories will be discussed separately.

B.1.1 FIXED LOCATIONS

All versions of DOS/65 use the lower part of page 1 as shown in Figure B-2 for certain critical items. Specifically in all versions the assembly code which describes page one is:

```
*         =         $100
WBOOT     JMP       SIM+3     Warm Boot
PEMJMP    JMP       PEM       Jump to PEM
IOBYTE    .BYT      0         I/O status byte (unused)
DFLFCB    *=        *+33      Default fcb
DFLBUF    *=        *+128     Default buffer
```

$100-102 Jump to DOS/65 Warm Boot
A warm boot reads back into memory CCM and PEM (but not SIM) and then runs CCM.

$103-105 Jump to PEM

$107-$127 Default FCB
This is set up by CCM when a transient is executed.

$128-$1A7 Default Disk Buffer

Since the default disk buffer and fcb can be moved almost anywhere, no real stack space is lost. Moreover, the jump addresses for PEM and Warm Boot could both be captured by the transient and thus free all of Page one for unrestricted use by the transient. While the fcb and buffer can be moved almost anywhere, the use of the 128 byte default buffer (one record or sector) is straightforward. A read will fill in the buffer with data from the disk while a write will transfer the data in the buffer to the disk. At least two schemes are useable in reading or writing large amounts of data - either move the data to the buffer or move the buffer to the data. Moving the data to the buffer is the safest approach but is also probably the slowest. Moving the buffer to (really through) the data as successive records are written has one danger. During a write operation

which requires opening of a new extent, the contents of the buffer are destroyed. Thus, a user program which used that approach would find part of the data in memory to be in error after the write. If more than 16K bytes must be written, than do not use the moving buffer approach.

B.1.2 VARIABLE LOCATIONS

The only location which varies as a function of the version is the TEA start address.

SIM – variable length
PEM – 3072 bytes
CCM – 2048 bytes
TEA – variable length
PAGE 1
PAGE 0

NOTES

1. TEA start address is a function only of version (e.g., S=$200, K=$2000) but is always on a page boundary. TEA length is a function of version, MEMORY SIZE, and SIM length. Transients are loaded at start of TEA and entered at start of TEA for execution.
2. CCM, PEM and SIM start addresses are a function of MEMORY SIZE and SIM length but are always on a page boundary.
3. Lower half of Page 0 ($00 - $7F) is used by BOOT for some versions. BOOT is not permanently RAM resident thus page zero is free after BOOT completes execution.
4. First sixteen bytes in Page 0 have some restrictions. See text for details.
5. SIM is at least one page (256 bytes) long. It usually occupies the highest memory not used for essential I/O or ROM-based software.
6. See Figure B-2 for details of Page 1 memory usage.

Figure B-1 DOS/65 Memory Map

$1A8 to $1FF Stack
$128 to 1A7 Default Buffer
$107 to $127 Default FCB
$106 IO Status Byte
$103 to $105 JMP PEM
$100 to $102 JMP SIM+3

Figure B-2 Page One Memory Map

B.2 RESTRICTED LOCATIONS

Because of the way in which PEM saves, uses, and restores page zero; the first 16

bytes in page zero ($00 through $0F) cannot be used as part of fcbs, disk buffers, console input buffers or output strings. This does not mean that those locations are modified by DOS/65 - they are not modified nor is any other part of page zero. They can be used for any purpose other than those mentioned above with absolutely no problems. One additional consequence of the save-use-restore approach used by PEM is that a 6502 Reset may result in a scrambled page zero. So if the system hangs up, a 6502 Reset will normally require a DOS/65 cold start to be executed.

Upon entry to SIM from PEM it is possible that the first sixteen bytes of page zero will contain critical PEM parameters rather than the data stored there by a transient. Transients should not use those locations for storage of data to be used by SIM unless the data is passed to SIM through PEM or prior to calling PEM.

While DOS/65 itself (i.e., CCM, PEM, or SIM) does not require any part of page zero to be dedicated to its use, most of the transients supplied with DOS/65 do use some of page zero. In the current version, no program uses more than the region from $02 through $BF. Allowing for future growth it is suggested that the user follow these guidelines for use of page zero by SIM or the routines called by SIM:

1. Use page zero beginning at the top (i.e., $FF)

2. Use an absolute minimum number of page zero locations (preferably 32 or less)

APPENDIX C - FLAGS AND INTERRUPTS

C.1 CPU FLAGS

Calls to PEM will always result in the decimal flag (D) being cleared since the first thing that happens in PEM is a CLD instruction. The interrupt flag is not altered by CCM or PEM. If the interrupt option for SIM is used, then SIM will disable interrupts during disk I/O operations but will always exit with interrupts enabled (I = 0). The use of interrupts by the system or applications S/W is discussed more fully below. The state of the overflow (V) flag is indeterminate upon return from PEM as is the state of the carry flag (C). The zero and negative flags (Z and N) are set or cleared as a function of the return value in A.

C.2 INTERRUPTS

While DOS/65 (and PEM in particular) is not designed to use interrupts it can be used in an interrupt environment. The major limitation is how page zero is used by PEM and SIM (CCM uses no page zero memory!). Since SIM and PEM do some swapping of user and DOS/65 data in and out of page zero, the state of page zero (i.e., is the user data there or is it DOS/65 data) upon execution of a random interrupt is not easily determined. If the interrupt routine does not use or alter the affected parts of page zero and if the CPU state is fully restored upon return to DOS/65 (as it should be anyway), there would be no problem.

CAUTION

Do not attempt to use PEM as a reentrant routine in an interrupt environment. It is not designed to be reentrant and will not work properly.

APPENDIX D - STANDARD INTERCHANGE FORMATS

The following Standard Interchange Formats (SIF) are available for DOS/65 Version 2.1. Some of these require special action and are only available by special order. Those are designated by the yellow highlighting in the table. Other formats, e.g., C64 CP/M 1541, are under development.

Code	Media	Tracks	MAXBLK	NSYSTR	NRECRD	Format Notes	Translation Table
STD-8 (SIF-A)	8 “	77	242	2	26	1,2,3,7	1
SIF-B	5.25 “	35	71	3	18	1,2,4,7	2
SIF-C	5.25 “	80	172	3	18	1,2,7	2
OSI-8 (SIF-D)	8 “	77	239	3	26	5,7	1
OSI-5 (SIF-E)	5.25 “	40	71	4	16	5,6	None
SIF-F	5.25 “	40	141	2	30	2,8,7	2
SIF-G	5.25 “	80	145	2	30	2,8,9	2
K-IV	5.25 “	40	196	1	40	10	None

FORMAT NOTES

1. Diskettes must be single sided, single density, soft sectored diskettes with hardware sectors of 128 bytes.

2. Diskettes should be formatted for compatibility with the Western Digital series of LSI controller circuits.

3. "Out-of-the-box" IBM compatible single density eight inch diskettes will also be satisfactory if they meet all other requirements.

4. It is recommended that all diskettes be certified and formatted for 40 tracks so that users having a 40 track capable drive can use the full diskette surface by changing MAXBLK to 82.

5. Diskettes shall be formatted for compatibility with the standard OSI disk controller hardware using the standards defined in the OSI NOTES.

6. Physical sector number shall be one more than the logical sector number.

7. Diskettes shall use BLKSCD=0 (1K) and MAXDIR=63 (64 directory entries).

8. Diskettes must be single sided, double density, soft sectored diskettes with hardware sectors of 128 bytes.

9. Diskettes shall use BLKSCD=1 (2K) and MAXDIR=63 (64 directory entries).

10. Kaypro IV formatted diskettes (i.e., double-sided, double-density, 40 track, 5.25 inch diskettes) can be used and freely interchanged with Kaypro IV or compatible CP/M machines with one possible exception. That exception is the result of the slightly different way in which DOS/65 defines directory space allocation. CP/M provides separate means of specifying the number of directory entries and the number of blocks allocated to those entries. For the Kaypro IV format this feature was used to specify 64 directory entries but to allocate two 2048 byte blocks for the directory rather than the single 2048 byte block actually needed to store the directory. DOS/65 does not provide separate ways to define the number of directory entries and the number of blocks allocated to the directory. For the Kaypro IV format DOS/65 would only allocate one block to the directory. That would mean that data would start at the wrong block. As a result, DOS/65 says that a Kaypro IV diskette can store 128 directory entries so that two blocks will be allocated. This makes the diskettes fully compatible unless the DOS/65 user writes more than 64 entries in the directory. While that is possible it is unlikely. However, as a consequence of this anomaly the user should always ensure that no more than 64 directory entries are used if diskettes are actually to be exchanged with Kaypro IV compatible CP/M systems.

SECTOR TRANSLATION TABLE 1

LOGICAL	PHYSICAL
0	1
1	7
2	13
3	19
4	25
5	5
6	11
7	17
8	23
9	3
10	9
11	15
12	21
13	2
14	8
15	14
16	20
17	26

18	6
19	12
20	18
21	24
22	4
23	10
24	16
25	22

SECTOR TRANSLATION TABLE 2

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

APPENDIX E - DEBLOCKING

E.1 GENERAL

DOS/65 Version 2.1 includes the logic needed to optimize read and write performance with disks having physical sector sizes larger than the 128 byte DOS/65 record. This is achieved by passing parameters to SIM as well as by the addition of code to SIM.

E.2 PERFORMANCE

This algorithm provides the following performance improvements

BASELINE (8" SSSD) = 42 sec.

8" SSDD (512 bytes/sector) = 27 sec.

when executing a typical assembly. It is interesting to note that the improvement noted for DOS/65 was comparable to that noted for similar de-blocking under CP/M.

E.3 SAMPLE CODE

```
;deblock
;sector deblocking algorithms for dos/65 2.1
;released:          24 november 1985
;last revision:
;          15 march 2008
;                   converted to TASM 3.x format
;dos/65 to host disk constants
;these parameters will need to be set to the values
;appropriate for the system in question.
;data shown is for example only.
blksiz    =         2048               ;dos/65 allocation size
hstsiz    =         256                ;host disk sector size
hstspt    =         32                 ;host disk sectors/trk
hstblk    =         hstsiz/128         ;dos/65 sects/host sector
d65spt    =         hstblk*hstspt      ;dos/65 sectors/track
secmsk    =         hstblk-1  ;sector mask
secshf    =         hstblk/2  ;shift to get host sector
;pem constants on entry to write
wrall     =         0                  ;write to allocated
wrdir     =         1                  ;write to directory
wrual     =         2                  ;write to unallocated
;page zero definitions
;the starting address should be set to an
;appropriate location in upper part of
;page zero.
pzstrt    =         $e0                ;first free page zero
;note that four page zero bytes are needed.
          *=        pzstrt
```

```
dmaadr    *=        *+2               ;dos/65 buffer location
mvepnt    *=        *+2               ;host buffer loaction
;code after here is shown at $f000 as an example only
;it is normally in the sim location as defined
;by the sim addressing.
          *=        $f000
;The sim entry points given below show the
;code which is relevant to deblocking only.
boot
          ;insert normal boot code here
wboot
          ;insert normal warm boot code here
;initialize key variables
;this code would normally be in the setup section.
          lda       #0                ;clear a
          sta       hstact            ;host buffer inactive
          sta       unacnt            ;clear unalloc count
          ;normal code to get ready to go to ccm goes here
;home the selected disk
home      lda       hstwrt            ;check for pending write
          bne       homed             ;there is so skip ahead
          sta       hstact            ;clear host active flag
homed     ;normal code for home starts here
          rts
;select disk
seldsk    sta       sekdsk            ;seek disk number
          ;insert normal dcb address capture
          rts
;set track given by registers ay
settrk    sta       sektrk            ;save seek track
          sty       sektrk+1
          rts
;set sector given by registers ay
setsec    sta       seksec            ;save seek sector
          sty       seksec+1
          rts
;set dma address given by ay
setdma    sta       dmaadr            ;save address
          sty       dmaadr+1
          rts
;translate sector number
sectrn    rts                         ;do nothing
;the read entry point takes the place of
;the previous sim definition for read.
;read the selected dos/65 sector
read      ldx       #0                ;x <-- 0
          stx       unacnt            ;clear unallocated count
          inx                         ;x <-- 1
          stx       readop            ;say is read operation
          stx       rsflag            ;must read data
          inx                         ;x <-- wrual
          stx       wrtype            ;treat as unalloc
          jmp       rwoper            ;to perform the read
;The write entry point takes the place of
;the previous sim defintion for write.
;write the selected dos/65 sector
write     sta       wrtype            ;save param from pem
          ldx       #0                ;say is
```

```
          stx       readop              ;not a read operation
          cmp       #wrual              ;write unallocated?
          bne       chkuna              ;check for unalloc
;write to unallocated, set parameters
          lda       #blksiz/128         ;next unalloc recs
          sta       unacnt
          lda       sekdsk              ;disk to seek
          sta       unadsk              ;unadsk <-- sekdsk
          lda       sektrk
          ldy       sektrk+1
          sta       unatrk              ;unatrk <-- sectrk
          sty       unatrk+1
          lda       seksec
          ldy       seksec+1
          sta       unasec              ;unasec <-- seksec
          sty       unasec+1
;check for write to unallocated sector
chkuna    lda       unacnt              ;any unalloc remain?
          beq       alloc               ;skip if not
;more unallocated records remain
          dec       unacnt              ;unacnt <-- unacnt-1
          lda       sekdsk
          cmp       unadsk              ;sekdsk = unadsk?
          bne       alloc               ;skip if not
;disks are the same
          lda       unatrk              ;sektrk = unatrk?
          cmp       sektrk
          bne       alloc               ;no so skip
          lda       unatrk+1
          cmp       sektrk+1
          bne       alloc               ;skip if not
;tracks are the same
          lda       unasec              ;seksec = unasec?
          cmp       seksec
          bne       alloc               ;no so skip
          lda       unasec+1
          cmp       seksec+1
          bne       alloc               ;skip if not
;match, move to next sector for future ref
          inc       unasec              ;unasec = unasec+1
          bne       nounsc
          inc       unasec+1
nounsc    lda       unasec              ;end of track?
          cmp       #<d65spt  ;count dos/65 sectors
          lda       unasec+1
          sbc       #>d65spt
          bcc       noovf               ;skip if no overflow
;overflow to next track
          lda       #0                  ;unasec <-- 0
          sta       unasec
          sta       unasec+1
          inc       unatrk              ;unatrk <-- unatrk+1
          bne       noovf
          inc       unatrk+1
;match found, mark as unnecessary read
noovf     lda       #0                  ;0 to accumulator
          sta       rsflag              ;rsflag <-- 0
          beq       rwoper              ;to perform the write
```

```
;not an unallocated record, requires pre-read
alloc     ldx       #0                  ;x <-- 0
          stx       unacnt              ;unacnt <-- 0
          inx                           ;x <-- 1
          stx       rsflag              ;rsflag <-- 1
;common code for read and write follows
;enter here to perform the read/write
rwoper    lda       #0                  ;zero to accum
          sta       erflag              ;no errors (yet)
          lda       seksec              ;compute host sector
          ldy       seksec+1
          sta       sekhst
          sty       sekhst+1
          ldx       #secshf             ;get shift count
shflpe    lsr       sekhst+1  ;do high
          ror       sekhst              ;then low
          dex
          bne       shflpe              ;loop if more
;active host sector?
          lda       hstact              ;host active flag
          pha                           ;save
          inx                           ;x <-- 1
          stx       hstact
          pla                           ;get flag back
          beq       filhst              ;fill host if not active
;host buffer active, same as seek buffer?
          lda       sekdsk
          cmp       hstdsk              ;same disk?
          bne       nmatch
;same disk, same track?
          lda       hsttrk              ;sektrk = hsttrk?
          cmp       sektrk
          bne       nmatch              ;no
          lda       hsttrk+1
          cmp       sektrk+1
          bne       nmatch
;same disk, same track, same sector?
          lda       sekhst              ;sekhst = hstsec?
          cmp       hstsec
          bne       nmatch              ;no
          lda       sekhst+1
          cmp       hstsec+1
          beq       match               ;skip if match
;proper disk, but not correct sector
nmatch    lda       hstwrt              ;host written?
          beq       filhst              ;skip is was
          jsr       writeh              ;else clear host buff
;may have to fill the host buffer
;so set host parameters
filhst    lda       sekdsk
          sta       hstdsk
          lda       sektrk
          ldy       sektrk+1
          sta       hsttrk
          sty       hsttrk+1
          lda       sekhst
          ldy       sekhst+1
          sta       hstsec
```

```
          sty       hstsec+1
          lda       rsflag              ;need to read?
          beq       noread              ;no
          jsr       readh               ;yes, if 1
noread    lda       #0                  ;0 to accum
          sta       hstwrt              ;no pending write
;copy data to or from buffer
match     lda       #0                  ;clear move pointer
          sta       mvepnt
          sta       mvepnt+1
          lda       seksec              ;mask sector number
          and       #secmsk             ;least signif bits
          tax                           ;make a counter
          beq       nooff               ;done if zero
clcpnt    clc
          lda       mvepnt
          adc       #128
          sta       mvepnt
          lda       mvepnt+1
          adc       #0
          sta       mvepnt+1
          dex
          bne       clcpnt              ;loop if more
;mvepnt has relative host buffer address
nooff     clc                           ;add hstbuf
          lda       #<hstbuf
          adc       mvepnt
          sta       mvepnt
          lda       #>hstbuf
          adc       mvepnt+1
          sta       mvepnt+1
;at this point mvepnt contains the address of the
;sector of interest in the hstbuf buffer.
          ldy       #127                ;length of move - 1
          ldx       readop              ;which way?
          bne       rmove               ;skip if read
;write operation so move from dmaadr to mvepnt
          inx                           ;x <-- 1
          stx       hstwrt              ;hstwrt <-- 1
wmove     lda       (dmaadr),y
          sta       (mvepnt),y
          dey
          bpl       wmove               ;loop if more
          bmi       endmve              ;else done
;read operation so move from mvepnt to dmaadr
rmove     lda       (mvepnt),y
          sta       (dmaadr),y
          dey
          bpl       rmove               ;loop if more
;data has been moved to/from host buffer
endmve    lda       wrtype              ;write type
          cmp       #wrdir              ;to directory?
          bne       nodir               ;done if not
;clear host buffer for directory write
          lda       erflag              ;get error flag
          bne       nodir               ;done if errors
          sta       hstwrt              ;say buffer written
          jsr       writeh
```

```
nodir     lda       erflag
          rts
;writeh performs the physical write to
;the host disk, readh reads the physical disk.
writeh
          ;hstdsk = host disk #, hsttrk = host track #,
          ;hstsec = host sect #. write "hstsiz" bytes
          ;from hstbuf and return error flag in erflag.
          ;return erflag non-zero if error
          rts
;
readh
          ;hstdsk = host disk #, hsttrk = host track #,
          ;hstsec = host sect #. read "hstsiz" bytes
          ;into hstbuf and return error flag in erflag.
          rts
;uninitialized ram data areas
sekdsk                                    ;seek disk number
          *=        *+1
sektrk                                    ;seek track number
          *=        *+2
seksec                                    ;seek sector number
          *=        *+2
hstdsk                                    ;host disk number
          *=        *+1
hsttrk                                    ;host track number
          *=        *+2
hstsec                                    ;host sector number
          *=        *+2
sekhst                                    ;seek shr secshf
          *=        *+2
hstact                                    ;host active flag
          *=        *+1
hstwrt                                    ;host written flag
          *=        *+1
unacnt                                    ;unalloc rec cnt
          *=        *+1
unadsk                                    ;last unalloc disk
          *=        *+1
unatrk                                    ;last unalloc track
          *=        *+2
unasec                                    ;last unalloc sector
          *=        *+2
erflag                                    ;error reporting
          *=        *+1
rsflag                                    ;read sector flag
          *=        *+1
readop                                    ;1 if read operation
          *=        *+1
wrtype                                    ;write operation type
          *=        *+1
hstbuf                                    ;host buffer
          *=        *+hstsiz
```