CHAPTER X

TPM-II SYSTEM CALLS

In the next few pages, we'll get to the core of the process of interfacing your program with TPM-II. The main mechanism involved is the system call. System calls provide a consistent and well-defined access point to the common programming tasks executed by TPM-II. We'll cover the concept and protocol of these system calls first, then examine each call in detail.

SYSTEM CALLS IN GENERAL

A system call is nothing more than the issuing of a command to TPM-II. A program can issue a broad range of system calls, encompassing character I/O, disk file tasks, and several other commands such as requesting and receiving the time of day.

Depending on the command the program wants executed, it can issue one of several types of system calls. Let's review these generic types:

Non-data commands: These system calls instruct TPM-II to execute a command thatdoesn't involve the input or output of data. An example would be the warm boot system call which signals to TPM-II that a program has terminated execution, and that a warm boot should be executed.

Byte I/O: System calls in this category pass a single character between the calling program and TPM-II. In most cases, these calls involve one of the character-oriented logical devices, such as the Console.

Block I/O: These system calls are primarily disk- oriented, and involve the passing of blocks of data or command parameters between the calling program and TPM-II.

System Call Protocol

Certain elements of the system call protocol are common to all three types of system calls. First, the call number (or function number) must be passed to TPM-II. Each of the system calls has an assigned number, and the calling program must inform TPM-II as to which call it wants executed. Secondly, all system calls must be initiated via a Z80 CALL instruction to memory location 0005H.

Finally, any data which is necessary to initiate the system call, or which results from it, must be transferred between TPM-II and the calling program. Byte I/O system calls pass data one byte at a time. These calls use the internal registers of the Z80 as the transfer medium. Block I/O calls, on the other hand, use system memory to transfer the block of data and parameters that are necessary for, or result from, their operations. With these system calls, the address of the block of memory is passed in the internal registers of the Z80.

Setting up a system call is quite simple. The number of the desired system call is placed in register C, and the data transfer mechanism is set up. If the system call is a Block I/O type call, the beginning address of the parameter or data block is placed in register set DE. If the call involves Byte I/O, the register used to pass the character is dependent on the direction the byte is being passed. If the calling program is passing a byte to TPM-II (as in the Console Output call), the byte is placed in register E. If the program is getting a byte from TPM-II (Console Input call), the value is returned in register A.

To summarize, executing a system call is a four-step process. The function call number is placed in register C, the byte or block address (if any) is set, a Z80 CALL 0005H instruction is executed, and finally, the status, and data (in the case of byte or block input functions) is checked by the calling program.

TPM-II's Use Of Registers

Most TPM-II functions return a result in the A register (usually the byte input in an I/O operation or an error code indicating success or failure of a disk operation). Word results are usually returned in the BC register.

When TPM-II is called, the HL, DE, X and Y registers are saved and restored unaltered upon return to the program, unless they are used to return results.

The BC and PSW registers are always cleared, unless they are used to hold function results. The prime registers (reached via the EXX instruction) are not used by TPM. However, other programs may use them, so any interrupt routines that make use of such registers should be adjusted accordingly.

System calls have been broken down into three groups: system calls dealing with input/output of characters to/from the four logical devices (Console, Punch, Reader, and List); file operations; and all functions not included in the first two groups, e.g., setting time-of-day, getting the version number, warm boot.

TPM-II'S DUAL MODE

Before we begin, one topic must be mentioned. TPM-II was originally written prior to the introduction of CP/M version 2.2 and its subsequent revisions (the version preceding 2.2 was 1.4). CP/M 1.4 supported only 28 system calls (0 through 27). These function calls have been faithfully repeated in TPM-II. However, extensions to the basic CP/M 1.4 repertoire were included in the original TPM, in which function calls 28 through 36 were used. When CP/M 2.2 was introduced, a conflict arose, since the new version of CP/M also used these function calls, but for different functions.

This conflict has been reconciled in TPM-II through a dual set of function calls. In the TPM mode, function calls 28 through 36 correspond to the TPM extensions, whereas in the CP/M mode the CP/M version 2.2 calls are executed. A program may select either mode, or switch between the two modes at any time.

The filetype or extension determines the default mode. If the program has a filetype .SYS, the TPM mode is selected when the program is invoked, whereas a .COM filetype will select the CP/M mode. Function 12 (Get Version Number) has been enhanced to allow a program to change modes, "in midstream" so to speak. If the DE register pair contain the value 0CCCCH, the CP/M mode is activated. A value of 0AAAAH sets up the TPM mode. Any other value in DE leaves the mode unchanged. This allows a program the best of both worlds: standard CP/M function calls, as well as TPM-II extensions.

(Text continues on next page)

**** TABLE 10 - 1 ****

TPM-II SYSTEM CALLS
TPM-II AND CP/M MODES

CALL NUMBER	FUNCTION
0	Reset System (Warm Boot)
1	Read Console Character w/echo
2	Write Console Character
3	Read Character from Reader
4	Write Character to Punch
5	Write Character to List Device
6	Get Serial # and Direct Console I/O
7	Get IOBYTE
8	Set IOBYTE
9	Write Buffer To Console
10	Read Console Buffer
11	Get Console Status
12	Return Version and Set Mode
13	Reset Disk System
14	Log In Disk Drive
15	Open Disk File
16	Close Disk File
17	Search For Disk File
18	Search For Next Disk File
19	Erase Disk File
20	Read Disk File Record
21	Write Disk File Record
22	Create New Disk File
23	Rename Disk File
24	Return Login Vector
25	Identify Logged-In Disk Drive
26	Set Disk I/O Address
27	Get Disk Parameters

(Table continues on next page)

CALL NUMBER	CP/M FUNCTION	TPM-II FUNCTION
28	Write Protect Disk	Read Console without echo
29	Get Read/Only Vector	Get System Date
30	Set/Reset File Attributes	Get Time
31	Get Disk Parameter Block	Trap Control
32	Set/Get User Group Number	Set Date/Time
33	Read Random Record	Chain Program
34	Write Random	Get TPM Version Number
35	Computer File Size	Do Direct Disk I/O
36	Set Random Record	Create File Control Block
37	Reserved For Future Use	Return Time In Seconds
38	Unused	Set/Reset File Attributes
39	Unused	Graphics Driver Support
40	Write Random w/Zero Fill	Multibank & Interrupt Support

****

BASIC I/O SYSTEM CALLS

The Basic I/O calls provide input and output for the four logical character I/O devices, namely the Console, Reader, Punch, and List. All employ byte I/O conventions, with the exception of the two buffered Console functions, which employ block I/O.

READ CONSOLE CHARACTER

Function number: 1
I/O mode: byte
CP/M-TPM mode: Both

Reads an input character from the console keyboard and returns it in the A register. Control characters CTRL-C, CTRL-P, CTRL-Q, CTRL-S, CTRL-B, and CTRL-\ are trapped by TPM and are not returned to the program. Each non-control character entered at the keyboard is echoed on the console screen. Default tabs are set in every eighth column.

READ CONSOLE CHARACTER WITHOUT ECHO

Function number: 28
I/O mode: byte
CP/M-TPM mode: TPM

Identical to Function 1, except that the characters entered at the keyboard are not echoed on the console screen.

DIRECT CONSOLE I/O

Function number: 6
I/O mode: byte
CP/M-TPM mode: CP/M

Allows characters to be input or output to the Console device without intervention from TPM-II. The input is not echoed, nor are the various control characters trapped.

The value in register E determines the direction of operation. If E contains 0FFH, TPM-II assumes an input operation, and the character is returned in register A. Any other value in E is considered an output value and will be output to the Console.

This function is the one exception to complete TPM-II compatibility with CP/M function calls 0 - 27. In the TPM-II mode, this call will return the base address of a 6-byte block of

memory containing the TPM-II serial number. In the TPM-II mode use function call 28 for console input without echo, and function 2 to output to the console.

WRITE CONSOLE CHARACTER

Function number: 2
I/O mode: byte
CP/M-TPM mode: Both

Writes the character in E to the console. The tab character (09H) is expanded to enough spaces to reach the next default tab stop (default tab stops are defined every eight characters). All other characters are written directly; your program is therefore responsible for initiating any special control functions provided by the terminal in response to the keyboard entries. No result is returned by this function.

READ CONSOLE BUFFER

Function number: 10
I/O mode: block
CP/M-TPM mode: Both

Used by TPM to read console commands. Characters input at the keyboard are buffered and echoed on the console screen until a carriage return <CR> is typed. The usual TPM-II editing control characters are supported.

The characters are stored in a buffer that starts at the address pointed to by the DE register. The first byte of this buffer must state the maximum buffer length (up to 255 bytes allowed); on return, the second byte contains the current buffer length. (These two bytes are not included in the count.) The input characters start in the third buffer position. The carriage return does not appear in the buffer; however if you enter a line-feed (0AH) to perform a physical carriage return without terminating the input, the line-feed character will appear in the buffer and be counted.

If the buffer fills before you enter a carriage return, no additional characters will be accepted and the console bell will ring. Any unused portion of the buffer will remain empty.

WRITE BUFFER TO CONSOLE

Function number: 9
I/O mode: block
CP/M-TPM mode: Both

Writes the buffer pointed to by the DE register to the console. The buffer is terminated by a null (00H) or dollar sign ($, 24H), neither of which is displayed. You can also terminate the buffer by setting the high-order bit of the last byte to be displayed. Your program is responsible for console control characters such as carriage return and line feed.

INTERROGATE CONSOLE STATUS

Function number: 11
I/O mode: byte
CP/M-TPM mode: Both

Determines whether console input is waiting to be read by your program. If input is waiting, the A register is set to non-zero; otherwise, a 00H is returned.

WRITE CHARACTER TO LIST DEVICE

Function number: 5
I/O mode: byte
CP/M-TPM mode: Both

Character in the E register is written to the list device. The characters are output in a literal manner. All special print characters, such as Tab, Linefeed, and Formfeed, , are passed directly to the list device, and no attempt is made to interpret them, as in the case of the Console output (function 2).

READ CHARACTER FROM READER

Function number: 3
I/O mode: byte
CP/M-TPM mode: Both

Reads a single character from the device currently defined as the logical Reader (usually a modem on the QX-10's serial port). The character is returned in the A register.

WRITE CHARACTER TO PUNCH

Function number: 4
I/O mode: byte
CP/M-TPM mode: Both

Character in the E register is written to the logical Punch device (usually a modem on the QX-10's serial port). No result is returned.

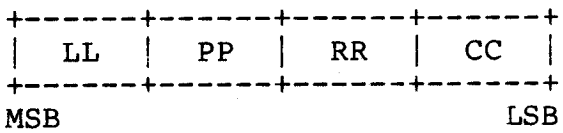
READ I/O ASSIGNMENT BYTE

Function number: 7
I/O mode: byte
CP/M-TPM mode: Both

Returns the I/O assignment byte in the A register. The I/O assignment byte consists of four fields of two bits each, with each field describing the current assignment of one of the four logical I/O devices, as shown below:

(Table on next page)

**** TABLE 10 - 2 ****



I/O BYTE

LOGICAL DEVICE	PHYSICAL ASSIGNMENT
LIST (LL)	00 CRT 01 TTY 10 LIST 11 USER *
PUNCH (PP)	00 CRT 01 TTY 10 USER 1 * 11 USER 2 *
READER (RR)	00 CRT 01 TTY 10 USER 1 * 11 USER 2 *
CONSOLE (CC)	00 CRT 01 TTY 10 USER * 11 N/A

* Denotes user-supplied driver. See Chapter 11 for details.

****

In the QX-10, the Zapple resident monitor handles I/O manipulations for the console (CRT), for both disk drives, for the printer (LIST), and for the RS-232C serial port (TTY). The physical devices marked with an asterisk (*) require you to provide your own I/O drivers if you wish to add them to the QX-10.

MODIFY I/O ASSIGNMENT BYTE

Function number: 8
I/O mode: byte
CP/M-TPM mode: Both

Replaces the I/O assignment byte with the value you place in the E register.

DISK I/O SYSTEM CALLS

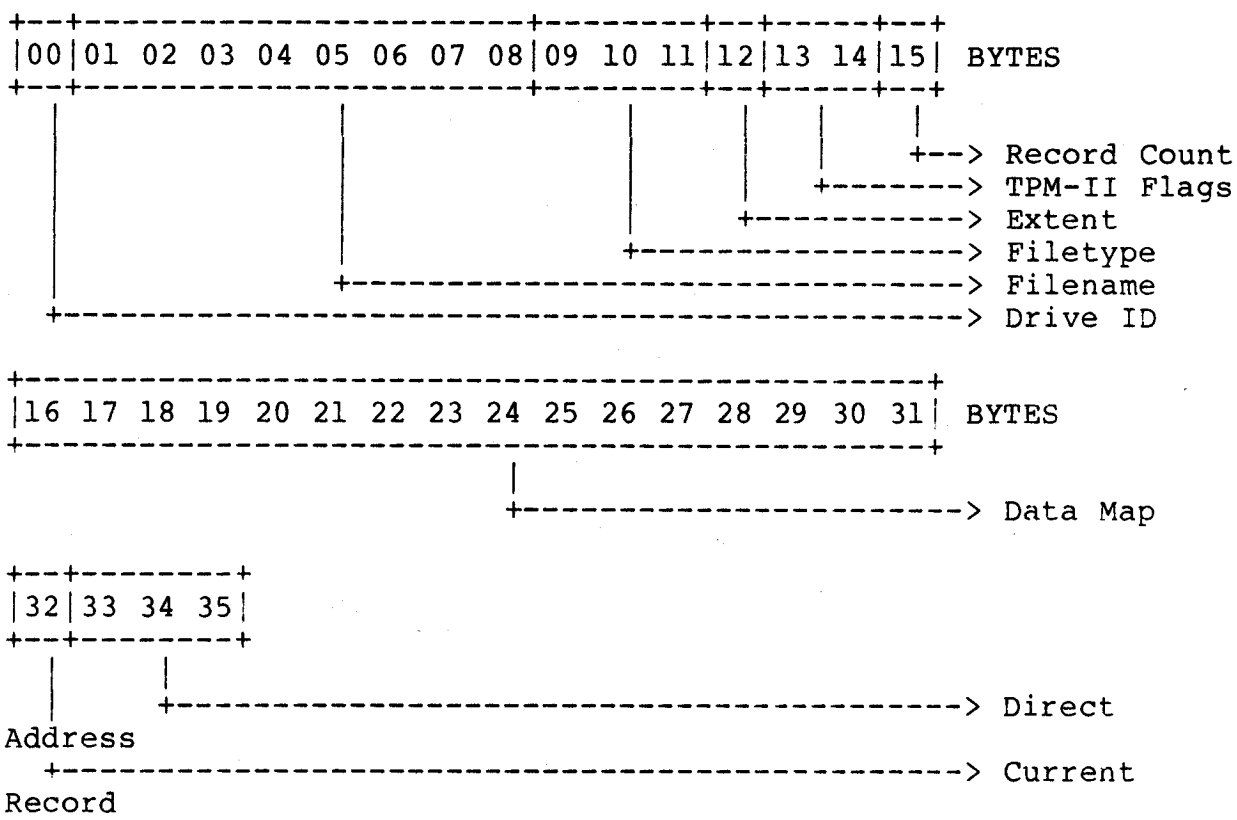
The disk I/O system calls are similar in structure to the basic I/O calls. The call number is placed in the C register and the TPM-II entry point (0005H) is called. However, with the basic I/O calls, we usually deal with only one character at a time and can therefore pass all of the necessary data to and from TPM-II in the internal registers (with the exception of the buffered Console input and output calls). With the disk I/O we are dealing with much larger blocks of data -- 128 bytes of data and 36 bytes or more of parameters. Obviously, the internal registers of the Z80 can't be used for data transfer.

TPM-II sets up two memory blocks to pass data and parameters back and forth. The first block, the disk data buffer, is a 128-byte block of memory that is used for all disk read and write operations. The second block is the File Control Block (FCB), a 36-byte block of memory used to pass parameters which control the disk I/O to and from TPM-II.

File Control Block (FCB) And Disk Data Buffer

The FCB and the disk data buffer can be located anywhere in memory that doesn't conflict with one of the TPM-II modules or a reserved memory location. TPM-II provides a "default" location for the FCB and the disk data buffer. The default FCB is located at memory location 005CH, and the default data buffer at 0080H. The exact location of the data buffer is set via a system call (function 26) if you wish to change it, and the address of the FCB must be contained in register pair DE during each disk I/O system call. Each field in the FCB must be filled in properly before a disk I/O system call is issued. In general, the first 13 and the last 4 bytes are the responsibility of the programmer to maintain. Bytes 13 through 31 contain allocation information and are maintained by TPM-II. They may be read by your program, but you should not attempt to change them. The FCB format is shown below:

**** TABLE 10 - 3 ****



FILE CONTROL BLOCK

File Maintenance System Calls

In order to properly keep track of the files on a disk, TPM-II requires that some preliminary system calls be issued by a program before actual read/write operation.

Creating, Deleting, and Renaming files:

If a file doesn't already exist on a disk, it must be created with the Create system call, which makes a directory entry on the disk for the file. After that, it can then be accessed by your program. Note that a file need only be created once. Once a file has been created, it can be erased with the Delete File system call, or given a new name with the Rename call.

Opening and Closing Files:

Once a file has been created, there will be one or more directory entries on the disk for that file. In order to read or write any information, TPM-II must know where the file resides on the disk. It must, therefore, get the allocation information contained in the directory into memory, which is accomplished with the Open File call. Open reads the information from the file's directory into the FCB contained in memory. Prior to issuing the Open File system call, the FCB for the file will contain the file name in the first 13 bytes and zeros in all other fields. After the Open call, the remaining fields will be filed with data corresponding to the allocation map on the disk for our particular file.

TPM-II will update the allocation unit map in memory as it reads and writes new data to the file. Thus, when your program has finished updating the file, the new allocation unit map must be written back into the directory so that it will be permanently stored. This is accomplished with the Close File system call. While reading a file does not change the allocation unit map in any way (no records are being added or deleted, therefore the allocation unit map in the directory is unchanged), it is good programming practice to Close all files when you are done reading. If you write anything to a file you must always Close the file, or the new data may be lost.

Searching For A File:

TPM-II provides a method, the Search File system call, for determining whether or not a file is on a disk. In order to support ambiguous filenames, it is possible to place wildcards in the filename contained in the FCB. If one or more question marks, or an asterisk, are encountered in the filename, the Search File system call will return the first filename that is a

match. In order to find out if there are other files that match, the Search Next call is used.

CREATE NEW DISK FILE

Function number: 22
I/O mode: block
CP/M-TPM mode: Both

Used to create a new disk file having the name stored in the FCB. It terminates the program if the file name turns out to be invalid.

If the file name is valid, the file is created and left open so that write operations may begin. If there is no room for the file on the specified disk, the function returns FFH in the A register. Any pre-existing file with that name and protection level is erased.

ERASE DISK FILE

Function number: 19
I/O mode: block
CP/M-TPM mode: Both

Deletes any files matching the file name, extension, user code, and protection level supplied in the FCB (providing the rules concerning protection levels have been observed). Ambiguous file names, extensions, and user codes may be placed in the FCB and erased.

RENAME DISK FILE

Function number: 23
I/O mode: block
CP/M-TPM mode: Both

Changes file names on the disk from within a program. The program must load the old file name into the first 16 bytes of the FCB (for example, at 5CH) and the new file name in the second 16 bytes, beginning at 6CH (four bytes following the first file name are not used). No result is returned.

You can rename files having ambiguous file names, but they must be ambiguous in the same relative positions to avoid naming conflicts during the renaming operation.

Using this routine, your application program can also set a new protection level on the renamed file, as follows:

If the low order bit of the first byte of the second file name (i.e., the disk drive ID) is set to one, the protection code is set to the high-order three bits of that byte. The disk drive ID of the first file name is assumed for both.

OPEN DISK FILE

Function number: 15
I/O mode: block
CP/M-TPM mode: Both

Opens the file specified in the FCB, i.e., it prepares the file for read and write operations. All disk files must be opened with Function 15 or Function 22 (create new file) before read and write operations can be performed. If the file to be opened doesn't exist, function 15 returns FFH in the A register.

CLOSE DISK FILE

Function number: 16
I/O mode: block
CP/M-TPM mode: Both

Allows you to close a disk file from within a program by specifying the file name in the FCB. Of course, the file must have been opened or created before it can be closed. If the specified file name is not present in the disk directory, the routine returns FFH in the A register. This error indication is usually caused by a disk or disk file being write-protected; sometimes, however, it can mean that the operator switched disks before the application had a chance to close all files.

Disk files that have not been written to need not be closed. However, files that have been written to should always be closed, since the disk directory entries describing the space in use by the file are updated at closing.

SEARCH FOR DISK FILE

Function number: 17
I/O mode: block
CP/M-TPM mode: Both

Determines whether or not a specified file name, having particular attributes, exists in the directory of the logged in disk

drive. When Function 17 is called, TPM returns, in the A register, the address of the disk directory entry for the first FCB that matches the specified file name and attributes. The address of the directory entry is used to allow the user program to examine the directory entry. If no match is found, FFH is returned.

As part of this function, TPM reads a 128-byte directory block (four FCB entries) into the disk I/O buffer. If an FCB match is found within the block, its number within the block is determined by taking the residue of the A register module 4 (i.e., the low order two bits). Each FCB occupies 32 bytes, so the following code would place the address of the desired directory entry into the HL register:

```
LXI  H,<disk I/O address>
ANI  03  ;mod 4
RRC      ;multiply A by 32
RRC
RRC
MOV  E,A
MVI  D,0
DAD  D
```

You can use Function 17 to search for an ambiguous file name, but you must then use Function 18, Search Next (see below) to get subsequent directory entries.

Normally, only the disk FCB entry for the first extent of a file is returned by the search function; however, placing a question mark (?) in the FCB extent field will return directory entries for all extents.

SEARCH FOR NEXT DISK FILE

Function number: 18
I/O mode: block
CP/M-TPM mode: Both

Similar to Function 17 (see above), but returns the byte address of the next disk directory FCB that matches the specified file name and attributes. No intermediate TPM disk I/O calls are allowed between functions 17 and 18 or between function 18 and the next call to 18. The address of the FCB must not be changed between calls.

File Read And Write System Calls

Once a file has been opened or created, reading and writing data is fairly simple. The last two bytes of the FCB are reserved for a Random Access Record Number, which can be anywhere from 1 to 65,535. Set these two bytes to indicate the record you wish to access prior to issuing either a read or write system call. Once the FCB has been set up properly, either a read or write operation can be initiated.

In the TPM mode, the Random Access Record Number should be set to the absolute record number you want to access. TPM will automatically perform the absolute record to extent and record conversion. After each read (function 20) or write (function 21) operation, the Random Access Record Number will be incremented by one, allowing easy sequential access. To perform random access, the Random Access Record Number is simply set before each read or write operation. For sequential access, it is only set on the first such operation.

In the CP/M mode, two read/write modes are used: random access and sequential access. Sequential access is far less flexible than the random access method. We won't discuss the details here (consult your CP/M documentation), except to mention one important point; sequential access uses the extent and record number to indicate the record, while random access uses the Random Access Record Number. Also, there are two sets of read and write commands. The sequential access mode uses function calls 20 (read) and 21 (write) while the random access mode uses 33 (read) and 34 (write). The sequential read and write operations are much faster than the random operations. Therefore, unless you need the random access capabilities, sequential is recommended. We won't discuss the details here, but sequential access causes the system to read the next sequential 128-byte record from the specified disk file into the disk I/O address. The file must previously have been opened or created. (See section on Function 26: Set Disk I/O Address.)

A byte describing the result of the read operation is returned in the A register, as follows:

00 - Successful read.
01 - End of File.
02 - Attempted to read unwritten records.

WRITE DISK FILE RECORD

Function number: 21
I/O mode: block
CP/M-TPM mode: Both

Function 21 writes the record stored in the 128 byte disk I/O area to the indicated disk file at the next sequential location. The file must be named in the FCB and must have previously been opened or created. The result returned in the A register has one of the following meanings:

00 - Successful write.
01 - File exceeds maximum size.
02 - No more space on disk for the file.
03 - Write protection violation. (See protection code rules in Chapter 3.)
FFH - No more space in disk directory.

READ RANDOM RECORD

Function number: 33
I/O mode: block
CP/M-TPM mode: CP/M

Lets you read directly from an open file. You specify the required record in bytes 21H, 22H, and 23H of the FCB, from which the LIOS determines the required extent number, calculates the correct allocation block within the extent, and looks up the entry in the extent data map. It then finds the record on the disk, reads it into the current file buffer, and returns 00 in A to indicate a successful read.

If the current extent isn't the one required, LIOS updates the directory entry for the current extent (if it has been modified) then copies the correct extent number into the FCB.

One of the virtues of random access is that you can alternately read and write to the same file. Unfortunately, random access files usually have holes in them, unless you've taken precautions. If the block you're reading has ever been part of another file, it could be filled with garbage, so watch out.

Function 33 returns a result in A, as follows:

00 - Successful read completed.
01 - Attempted to read unallocated record.
03 - Cannot close/update current FCB.
04 - Attempted to read unallocated extent.
06 - Direct address larger than allowed.

WRITE RANDOM RECORD

Function number: 34
I/O mode: block
CP/M-TPM mode: CP/M

Lets you write directly to an open file. You specify the required record in bytes 21H, 22H, and 23H of the FCB, from which the LIOS determines the required extent number, calculates the correct allocation block within the extent, and then writes the record. If the record already exists, the LIOS overwrites it with the new data. If no block has been allocated in this extent, LIOS assigns one. If the current extent is not the one required, LIOS updates the directory entry for the current extent (if it has been modified) and then copies the correct extent number into the FCB.

Unfortunately, you can easily create holes in which there is no useful data with random access files. The directory program doesn't know that such data is garbage, so if the block has ever been written to before, the directory will view it as full, even if you've only written to its first record. (See Function 35, below.)

Function 34 returns the same result codes as 33 in A, but adds the following:

05 - Directory overflow; new extent not created.

GET RANDOM RECORD ADDRESS

Function number: 36
I/O mode: block
CP/M-TPM mode: CP/M

Calculates (from the extent and record number) the Random Access Record Number. Please refer to your CP/M documentation regarding the use of this function to aid on switching between sequential and random access modes.

COMPUTE FILE SIZE

Function number: 35
I/O mode: block
CP/M-TPM mode: CP/M

Computes the size of the file pointed to by the DE register so that you can add data sequentially to its end, read its last record directly (Function 33), and so on.

Of course, if the file whose size you're checking was written directly, Function 35 returns only the identity of the highest numbered record in the directory FCB. As indicated above, the LIOS has no way to distinguish between meaningful data and nonsense in the record, so a random file with holes in it will generally appear larger than it actually is.

Miscellaneous Disk I/O Function Calls

RESET DISK SYSTEM

Function number: 13
I/O mode: non-data
CP/M-TPM mode: Both

Resets the disk system. Drive A is logged in, and the disk I/O address is set to its default, 80H (see section below on setting disk I/O address). Any waiting console input is flushed.

IDENTIFY LOGGED-IN DISK DRIVE

Function number: 25
I/O mode: byte
CP/M-TPM mode: Both

Returns a hexadecimal value representing the number of the currently logged-in disk drive. The value is returned in the A register. 00H means drive A, 01H drive B, and so on. Note that this is not the same convention as for the drive number in the FCB where 00H = current drive, 01H = A, etc.

LOG IN DISK DRIVE

Function number: 14
I/O mode: byte
CP/M-TPM mode: Both

Lets you log in a particular disk drive by placing a hexadecimal value in the E register. 00H logs in drive A, 01H drive B, and so on. Note that this is not the same convention as for the drive number in the FCB where 00H = current drive, 01H = A, etc.

SET DISK I/O ADDRESS

Function number: 26
I/O mode: byte
CP/M-TPM mode: Both

Every disk write or read requires the transfer of 128 bytes to or from the disk I/O buffer. At every system initialization, whether cold start or warm boot, the address of this buffer is set to 80H (the default I/O buffer). You can set the I/O buffer address to a different value by using Function 26.

If the application requires that several disk files be opened simultaneously, we recommend that you allocate a separate 128-byte buffer for each file, and call Function 26 for every read or write, making a positive buffer selection for every operation.

Since TPM uses the default I/O buffer for all file opens and closes, except for program chaining (see Function 33, above), you should set a new I/O buffer address if your program will need to access the information in it after it has been written out.

GET DISK INFORMATION

Function number: 27
I/O mode: byte
CP/M-TPM mode: Both

Returns a set of addresses which point to the information TPM maintains concerning disk drives. The program may then access whatever information is desired. Needless to say, these tables should not be altered in any way.

The addresses returned are as follows:

IX: Generic Information for the selected disk drive. The following table lists the definition of each parameter as well as its size, hex offset from the beginning of the block, and label for the parameters that have equivalents under CP/M. This information is equivalent to the CP/M Disk Parameter Block (DPB).

**** TABLE 10 - 4 ****

OFFSET	SIZE	LABEL	DEFINITION
00	WORD	SPT	Sectors Per Track
02	BYTE	BSH	Block Shift Factor
03	BYTE	BLM	Block Mask
04	BYTE	EXM	Extent Mask
05	WORD	DSM	Number of Blocks - 1
07	WORD	DRM	Number of Directory Entries - 1
09	WORD	ALV	Directory Allocation Mask
0B	WORD	CKS	Directory Check Size
0D	WORD	OFF	Directory Track
0F	WORD	*	Total Track in this Unit
11	WORD	*	Total Bytes for Allocation Map
13	BYTE	*	KBytes per Block
14	BYTE	*	Error Retry Count
15	BYTE	*	Sector Count in Track 00
16	BYTE	*	S-Flag (If bit 0 = 1, a Skew Table is present)
17	WORD	*	Address of Physical Device Handler
19	BYTE	*	X-Flags
1A	WORD	*	Physical Media Sector Size
1C	BYTE	*	Blocking/Deblocking Mask
1D	n BYTES	*	Skew Table Size (used to map Logical to Physical Sector Numbers. n is equal to the number of sectors per track (SPT above).

* denotes parameter unique to TPM-II

****

IY: Current Information for the selected disk drive. The information is equivalent to the CP/M Disk Parameter Header (DPH).

**** TABLE 10 - 5 ****

OFFSET	SIZE	LABEL	DEFINITION
00	WORD	*	Current Track Number
02	WORD	*	Current Sector Number
04	BYTE	*	Online Flag. (If bit 0 = 1, disk is mounted in drive and ready to use.)
05	BYTE	*	PIOS System Flag Byte
06	BYTE	*	Current User Group Number
07	32 BYTES	*	Directory Check Vector for CP/M 2.2
27	WORD	*	Translation Table Address
29	WORD	*	CP/M Scratch Pad Area # 1
2B	WORD	*	CP/M Scratch Pad Area # 2
2D	WORD	*	CP/M Scratch Pad Area # 3
2F	WORD	DIRBUF	Directory Buffer Address
31	WORD	PDB	Address of DPB (IX register contents)
33	WORD	CSV	Address of Check Vector
35	WORD	*	Address of Allocation Map
37	Variable	ALV	Allocation Map Vector. (One bit per available block on disk. Bit 0 of 1st byte represents first block, Bit 1 the second block, and so on. If the bit is 1, block is in use. A 0 indicates block is available. The word in bytes 11 & 12 of DPB contains total number of block on disk.)

* denotes parameter unique to TPM-II

****

HL, BC: Allocation vector for the selected disk drive. The allocation table is actually contained within the current information table, as shown below, so it may accessed via the IY register as well.

DIRECT DISK I/O

Function number: 35
I/O mode: byte
CP/M-TPM mode: TPM

Performs disk I/O at any track and sector, and It effectively bypasses the standard TPM disk file interface. Be careful: This function can wreak havoc on the content of a disk.

The Function Codes given below define the operation to be performed; the Parameters defined in the table specify register usage for the codes.

**** TABLE 10 - 6 ****

FUNCTION CODE	OPERATION	PARAMETERS
00	Home Head	None
01	Select Drive	Drive number in L. A=00, B=01, etc.
02	Set Track #	Track number in HL. (Note: track #s start with 0)
03	Set Sector #	Sector number in L. (Note: sector #s start with 1)
04	Set I/O address	New address is in HL.
05	Read Disk Sector	Retry count is in L.
06	Write Disk Sector	Retry count is in L.

****

To perform direct disk I/O with Function 35, set the address where the I/O will take place, select the desired drive, track, and sector, and initiate the read or write. If required, use Function 27 (see above) to determine the characteristics of the particular drive in use.

If the read or write fails on the initial try, TPM will retry the operation the number of times you specify in the L register, and will return an error code only when the retry is exhausted. Error codes are returned in the A register; zero in A therefore means that the operation was successful.

Table 10 - 7 lists disk I/O error codes for the QX-10 implementation of TPM-II. Any error with bit 7 set is considered fatal, and operator intervention is required to clear it.

**** TABLE 10 - 7 ****

ERROR CODE	DEFINITION
00H	Blank Unformatted Disk
04H	Lost Data (speed wrong)
08H	CRC Error (data) encountered
10H	Requested record header not found
18H	CRC Error (header) encountered
C0H	Disk Physically write-protected
FFH	Drive empty

****

PARSE DISK FILE NAME

Function number: 36
I/O mode: byte
CP/M-TPM mode: TPM

The TPM internal Function that parses disk file names for programs (normally executed at address 5CH) is also available for use in application programs, as follows:

Function 36 attempts to parse a file name from the string pointed to by HL. If a valid file name is found it is placed in the 12-byte area pointed to by the DE register. The result is in the format of the first 12 bytes of a file control block.

Function 36 terminates the file name string at the first character that is not valid in a TPM file name. ([]= :;<>,). The BC register returns the address of the byte that follows the last byte of the valid file name.

When the file name string in HL is ambiguous (contains * or ?), Function 36 expands the asterisk (*) to enough question marks (three to eight) to produce the proper FCB format.

When only a disk drive ID is found (that is, when the string terminator is :) Function 36 leaves the file name and file type blank.

The A register returns a byte that codes the result of the Function, as shown below. The codes are indicated by the corresponding bit being set to 1.

**** TABLE 10 - 8 ****

BIT	DEFINITION
0	Error, No File Name Found
1	Only Disk Drive ID Found
2	No File Type Supplied
6	File Name is *.* (all files)
7	File Name Ambiguous

****

WRITE PROTECT DISK

Function number: 28
I/O mode: byte
CP/M-TPM mode: CP/M

Sets the write-protect bit on the default drive. The write protection remains in effect until the next warm boot. Once this bit is set, any attempt to write to this drive will generate a write-protect message: "*The Diskette is Write Protected*," which the operator can override by typing RETURN. Entering a CTRL-C will abort the operation.

Calling Function 28 a second time resets the write-protect bit to remove the protection.

GET READ-ONLY VECTOR

Function number: 29
I/O mode: byte
CP/M-TPM mode: CP/M

Identifies drives for which the write protect bit has been set. This allows an application program to discover if a disk is Write Protected. A drive is considered Write Protected if either the notch is covered (hardware) or if the Write Protect bit has been set via function 28 (software). The Least Significant Bit (LSB) represents Drive A: -- the Most Significant Bit (MSB) drive P:.

SET/RESET FILE ATTRIBUTES

Function number: 30
I/O mode: byte
CP/M-TPM mode: Both

Searches the directory of the selected drive for all matches to the file name stored at the FCB address. When a match is found, bytes 01 through 0BH (Filename and Filetype) are replaced by those in the FCB, which can contain a pattern of most significant bits that is different from the disk file. The disk file directory is thereby updated to a new attribute status.

GET DISK PARAMETER BLOCK (DPB) ADDRESS

Function number: 31
I/O mode: byte
CP/M-TPM mode: CP/M

Returns the address of the active DPB in HL. For an explanation of DPB refer to Function 27. GET/SET CURRENT USER CODE

Function number: 32
I/O mode: byte
CP/M-TPM mode: CP/M

Used to interrogate the current User Code by entering FFH in the E register, or change it by entering the new User Code in E. (See Chapter 3.) If you interrogate the User Code, the result is returned in A. There are 256 possible numerical User Codes, from 0 through 255. Note, however, that you can't set the User Code to 255 with this function since you would need to place 0FFH in register E, thus instructing TPM-II to get the current User Code. User Code 255 is still a valid code, and may be set with the ICP User command.

OTHER TPM-II SYSTEM CALLS

In addition to the basic and disk I/O calls we have covered, TPM-II supports several other calls, which, among others, provide access to the QX-10's real-time clock and provide a mechanism for chaining programs via system calls.

CHAIN TO ANOTHER PROGRAM

Function number: 33
I/O mode: byte
CP/M-TPM mode: TPM

Loads a second program and conditionally executes it. The DE register must point to an FCB that gives the disk file name of the program. After the new program is loaded, control is either returned to the caller (B=0), or the called program is executed at the <load address>. If TPM encounters an error, it always returns control to the caller, regardless of the value passed in B.

Note that the <load address> + <file length> must always be below the LIOS (the value at address 6, D800), or you will cause a MEMORY OVFL error.

If the load fails because the called program you try to read is too large, TPM terminates both programs. A 01H is returned in the A register if you try to read a random access file that has "holes" in it. (Refer to section on random access files.)

GET SYSTEM DATE

Function number: 29
I/O mode: block
CP/M-TPM mode: TPM

Returns the current date, in ASCII, to the 8-byte area pointed to by the DE register, in the format MM/DD/YY (standard calendar usage). You can set the date in either of two ways:

-By means of the SET-TIME utility (see Chapter 3).

-By means of TPM Function 32 (see below).

GET SYSTEM TIME

Function number: 30
I/O mode: block
CP/M-TPM mode: TPM

Returns the current time, in ASCII, to the 8-byte area pointed to by the DE register, in the format HH:MM:SS ("military" time). You can set the time in either of two ways:

-By means of the SET-TIME utility (see Chapter 3).

-By means of TPM Function 32 (see below).

GET SYSTEM TIME (IN SECONDS)

Function number: 37
I/O mode: byte
CP/M-TPM mode: TPM

The current time, expressed in seconds is returned to the HL register (least significant digits) and the BC register (most significant digits). When the BC and HL registers are concatenated, they create a 4-byte hexadecimal number that expresses the total number of seconds elapsed since midnight. The maximum value is 86,400 seconds (24:00:00 hours).

SET SYSTEM TIME AND DATE

Function number: 32
I/O mode: block
CP/M-TPM mode: TPM

Under normal circumstances, the operator will set the date and time via the SET-TIME utility (see Chapter 3). However, if an application requires that the date and time be set or reset under program control, Function 32 permits you to do so.

Each call to this routine changes either the date or the time, depending on whether the first hexadecimal byte of the input area is a zero (date) or one (time). In either case, the remaining three bytes give the three fields of the date or time, in binary coded decimal (BCD). For example, to set the date to 10/7/79, the four input bytes would be: 0, 10, 7, 79. To change the time to 23:09:31 these bytes would be 1, 23, 09, 31.

GET TPM SERIAL NUMBER

Function number: 6
I/O mode: block
CP/M-TPM mode: TPM

Returns the address of the 6-byte area containing the TPM serial number, which is stored as an ASCII value. The address is returned in the BC register pair. Your program should never write into the area containing the serial number, for obvious reasons. Note: this is the single exception to TPM-II's compatibility with CP/M for function calls 0 through 27. In the CP/M mode, function call 6 is Direct Console I/O.

GET SYSTEM IDENTIFICATION

Function number: 12
I/O mode: block
CP/M-TPM mode: Both

Returns the address of the area containing the current TPM version number, which is stored as an ASCII value, delimited by a null (00). The address is returned in the BC register pair. For obvious reasons, your program should never write into the area containing the version number.

In keeping with CP/M convention, Function 12 returns zero in the HL pair if the system is in the TPM mode, and 0022H if it's in the CP/M mode.

You can examine the DE register pair from your application program to find out if the system is operating in TPM or CP/M. If DE returns 0054H (ASCII T), the system is operating in TPM, regardless of the current mode indicated by HL.

You can shuttle from TPM to CP/M between extended calls by passing a unique value in the DE pair before calling Function 12. This approach allows your program to take full advantage of TPM's extended capability as well as CP/M 2.2 features. To set the mode to TPM, pass the value 0AAAAH to the DE pair; to set it to CP/M, pass 0CCCCH to DE. (Any other value will leave the current mode unchanged.)

RESET SYSTEM

Function number: 0
I/O mode: non-data
CP/M-TPM mode: Both

Identical to a JMP 0. It terminates program execution and re-initializes the system.

VIDEO BIT DRIVER SUPPORT

Function number: 39
I/O mode: block
CP/M-TPM mode: TPM

Provides access to the QX-10's Video Bit Driver Routines. Due to its complexity, this function is described in Appendix C.

MULTIBANK AND INTERRUPT SUPPORT

Function number: 40
I/O mode: block
CP/M-TPM mode: TPM

Provides the linkage to such Multi-bank Functions as Spooler, Clock Display, and Serial port buffered reads. A complete description of these functions is available in the SYSINIT document.

PROGRAMMING NOTES

When TPM begins executing a user program, the stack pointer (the SP register) points to the stack used privately by TPM. If your program will use the stack, move it to some other location in the TPA. This will keep your program from smashing into TPM. Here's the fast way:

```
LXI  SP,<top of stack address>
```

(Remember, the stack grows towards smaller memory addresses.) If you use Linker to create your program, the code it generates initializes the stack to the highest address in user memory with:

```
LSPD 6
```

NOTE: Your programs should reserve a few extra words of stack space to allow for compatibility with possible future versions of TPM.

The usual way to terminate a program is to jump to address zero, which causes TPM to reinitialize itself. A slightly faster way to terminate the program is to simply return (RET) to TPM, since TPM actually calls 100H to initiate the program. However, the program must restore the stack pointer to its original location. For example:

```
LXI  H,0
DAD  SP
LXI  SP,<user stack address>
PUSH H
```

This moves the TPM stack pointer into the HL register, and saves it on the user stack. You can then let the user program run normally, using its own stack. At the end of the program:

```
POP  H
SPHL
RET
```

This restores the TPM stack pointer from the user stack, and returns to TPM. The program must not have overwritten any portion of the ICP, or this method will not work. If the ICP is not intact, a normal warm boot must be executed.

NOTE: This program termination method is very powerful -- but dangerous. Don't use it in programs that require the operator to change disks while the program is running; you could do severe damage to the disk when the operator executes the next program.

END CHAPTER 10