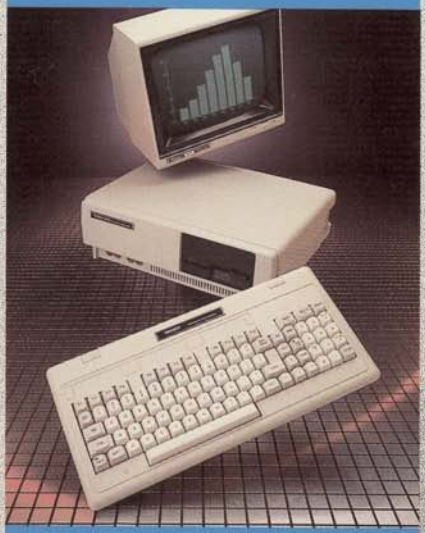
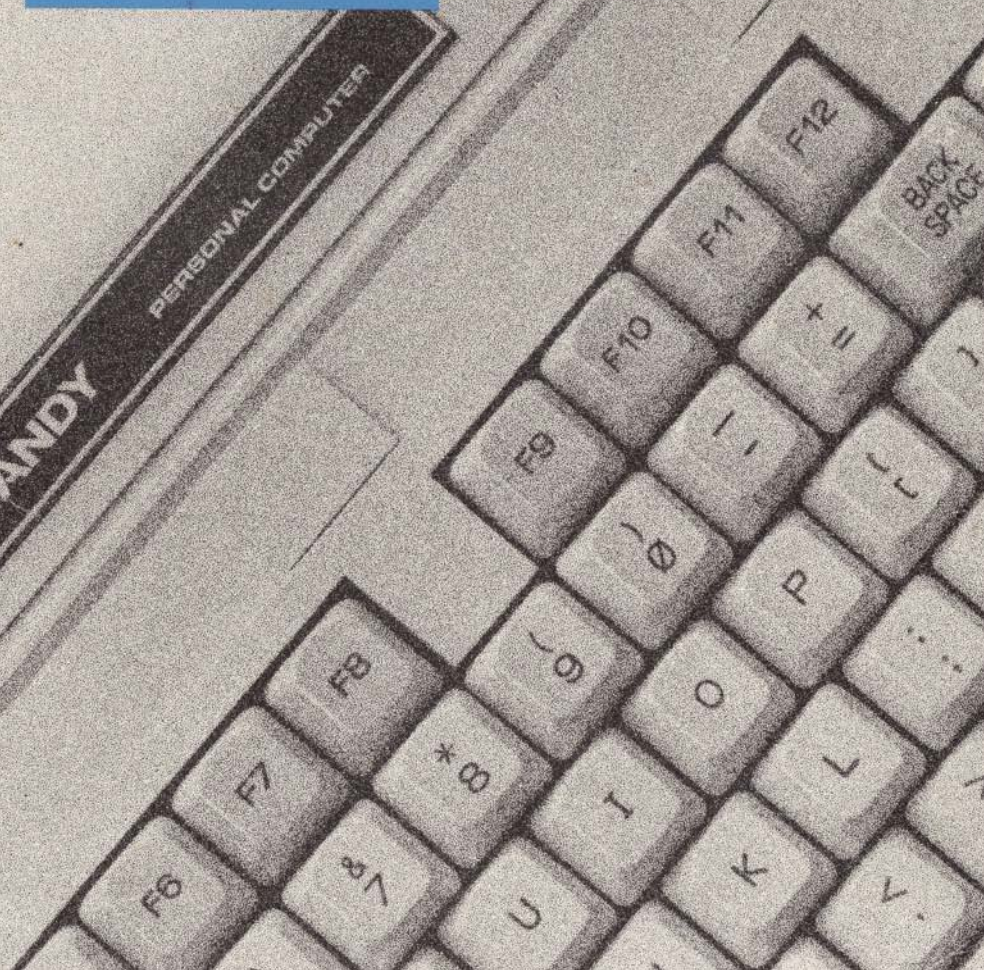
CAT. NO. 25-1503

TANDY® 1000

PROGRAMMER'S REFERENCE





TERMS AND CONDITIONS OF SALE AND LICENSE OF TANDY COMPUTER EQUIPMENT AND SOFTWARE PURCHASED FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND RADIO SHACK FRANCHISEES OR DEALERS AT THEIR AUTHORIZED LOCATIONS

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this computer hardware purchased (the "Equipment"), and any copies of software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. **This warranty is only applicable to purchases of Tandy Equipment by the original customer from Radio Shack company-owned computer centers, retail stores, and Radio Shack franchisees and dealers at their authorized locations**. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or a participating Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, a participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. **EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE." IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE." NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.**

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the TANDY Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby Radio Shack sells or conveys such Equipment to a third party for lease to CUSTOMER.

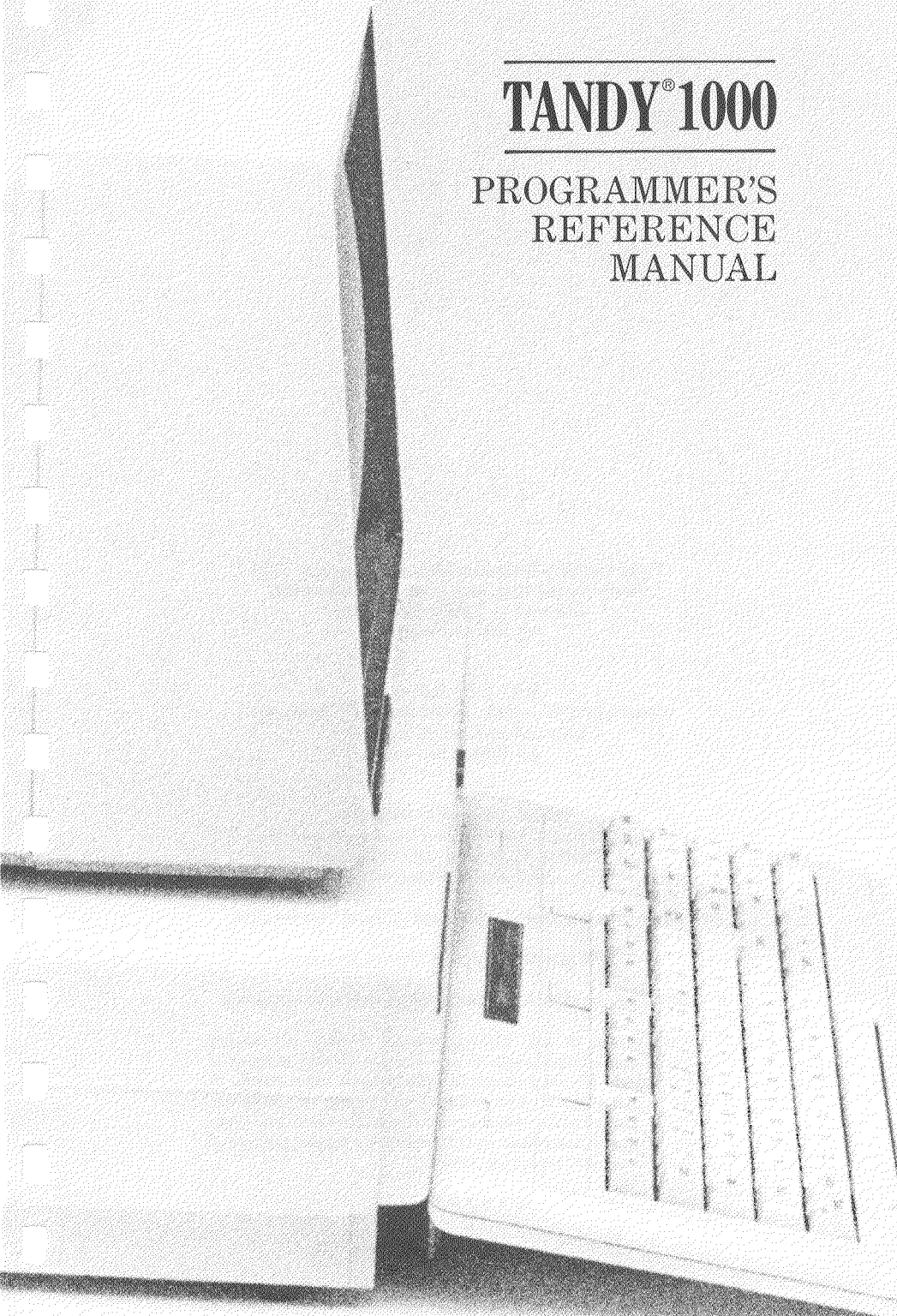
B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and or licensor of the Software and any manufacturer of the Equipment sold by Radio Shack.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

TANDY® 1000

PROGRAMMER'S REFERENCE MANUAL



10 9 8 7 6 5 4 3 2 1

Contents

Chapter 1. System Calls 9
Calling and Returning 9
Console, Printer, and Disk Input/Output Calls 10
Hierarchical Directories 10
System Call Descriptions 12
Interrupts 12
Function Calls 25
Categories of Calls 25
Error Codes 25
File Handles 26
ASCIIZ Strings 26
Calling MS-DOS Functions 27
CP/M® — Compatible Calling Sequence 27
Treatment of Registers 28
MS-DOS Function Calls in Numeric Order 28
MS-DOS Function Calls in Alphabetic Order 30
Function Call Reference 32
Macro Definitions for MS-DOS System Call Examples 137
Interrupts 137
Functions 137
General 142
Extended Example of MS-DOS System Calls 143

Chapter 2. MS-DOS Control Blocks and Work Areas 151
File Control Block (FCB) 151
Extended File Control Block 153
Program Segment 153
Program Segment Prefix 157

Chapter 3. MS-DOS Initialization and Command Processor 161

Chapter 4. MS-DOS Disk Allocation 165
MS-DOS Disk Directory 166
File Allocation Table (FAT) 168
How to Use the File Allocation Table 170

Chapter 5. Device Drivers 173
Types of Devices 173
Device Headers 174
Pointer to Next Device Header Field 174
Attribute Field 175
Strategy and Interrupt Routines 176
Name Field 176
Creating a Device Driver 176
Installation of Device Drivers 177
Request Header 177
Unit Code Field 177
Command Code Field 178
MEDIA CHECK and BUILD BPB 178
Status Field 179
Function Call Parameters 181
MEDIA CHECK 182
BUILD BPB (BIOS Parameter Block) 183
Media Descriptor Byte 184

Chapter 6. BIOS Services 191
Device I/O Services 191
Introduction 191
Keyboard 193
Video Display 195
Serial Communications 204
Line Printer 207
System Clock 208
Disk I/O Support for the Floppy Only 210
Equipment 213
Memory Size 213

Appendix A Extended Screen and Keyboard Control .. 217
Cursor Control 218
Erasing 220
Modes of Operation 221
Keyboard Key Reassignment 223

Appendix B Keyboard ASCII and Scan Codes 225

Appendix C MS-DOS Memory Map 229
ROM BIOS Data Area 230
Keyboard Illustration 235

System Calls

Chapter 1

SYSTEM CALLS

MS-DOS interrupt and function calls provide a convenient way to perform certain primitive functions and make it easier for you to write machine-independent programs. This chapter describes how and when these routines can be called and the process performed by each.

Calling and Returning

You invoke the system calls from Macro Assembler by moving required data into registers and issuing an interrupt. As some of the calls destroy registers, you may have to save the register contents before using a system call.

The system calls are also invoked from any high-level language that has modules capable of linking with assembly language modules.

Control is returned to MS-DOS in 4 ways:

- Issue Function Call 4CH:

```
MOV  AH,4CH
INT  21H
```

  This is the preferred method.

- Issue Interrupt 20H:

```
INT  20H
```

  This method simulates system call 00H.

- Jump to location 0 (the beginning of the Program Segment Prefix):

```
JMP  0
```

  As location 0 of the Program Segment Prefix contains an INT 20H instruction, this technique is 1 step removed from technique 2.

- Issue Function Call 00H:

```
MOV  AH,00H
INT  21H
```

  This transfers control to location 0 in the Program Segment Prefix.

Console, Printer, and Disk Input/Output Calls

The console and printer system calls let you interface with the console device and the printer without using machine-specific codes. You will still be able to take advantage of specific capabilities (such as clear the screen or position the cursor on your display or double-strike and underline on your printer). You do this by using constants for the required codes and reassembling with the correct constant values for the attributes.

Many system calls that perform disk input and output require the placing of values into or the reading of values from 2 system control blocks: the File Control Block (FCB) and the directory entry. For a description of the FCB, see the section "File Control Block" in Chapter 2. For details on the directory entry, see "Disk Directory" in Chapter 4.

Hierarchical Directories

MS-DOS supports hierarchical (tree-structured) directories, similar to those found in the XENIX operating system. (For information on tree-structured directories, refer to the MS-DOS User's Guide.)

The following system calls are compatible with the XENIX system:

Function 39H	create a subdirectory
Function 3AH	remove a directory entry
Function 3BH	change the current directory
Function 3CH	create a file
Function 3DH	open a file
Function 3FH	read from a file or device
Function 40H	write to a file or device
Function 41H	delete a directory entry
Function 42H	move a file pointer
Function 43H	change attributes
Function 44H	I/O control for devices
Function 45H	duplicate a file handle
Function 46H	force a duplicate of a file handle
Function 4BH	load and execute a program
Function 4CH	terminate a process
Function 4DH	retrieve the return code of a child process

There is no restriction in MS-DOS on the depth of a tree (the length of the longest path from root to leaf) except in the number of allocation units available. The root directory will have a fixed number of entries. For non-root directories, the number of files per directory is limited only by the number of allocation units available.

Subdirectories of the root have a special attribute set indicating that they are directories. The subdirectories themselves are files, linked through the File Allocation Table (FAT). Their contents are identical in character to the contents of the root directory.

Attributes apply to the tree-structured directories in the following manner:

Attribute	**Meaning/Function for files**	**Meaning/Function for directories**
volume id	Present at the root. Only 1 file may have this set.	Meaningless.
directory	Meaningless.	Indicates that the directory entry is a directory. Cannot be changed with Function Call 43H.
read only	Old create, new create, new open (for write or read/write) will fail.	Meaningless.
archive	Set when file is written. Set/reset via Function 43H.	Meaningless.
hidden/ system	Prevents file from system being found in search first/ search next. Old open will fail.	Prevents directory entry from being found. Function 3BH will still work.

System Call Descriptions

Many system calls require that parameters be loaded into 1 or more registers before the call is issued; this information is given under Entry Conditions in the individual system call descriptions. Most calls return information in the registers, as given under Exit Conditions and Error Returns.

For some of the system calls, a macro is defined and used in the example for that call. All macro definitions are listed at the end of the chapter, together with an extended example that illustrates the system calls.

The examples are not intended to represent good programming practice. Error checking and user friendliness have been sacrificed to conserve space. Many of the examples are not usable as stand-alone programs but merely show you how to get started with this command. You may, however, find the macros a convenient way to include system calls in your assembly language programs.

Interrupts

MS-DOS reserves Interrupts 20H through 3FH for its own use. Memory locations 80H to FCH are reserved for the table of interrupt routine addresses (vectors).

To set an interrupt vector, use Function Call 25H. To retrieve the contents of a vector, use Function Call 35H. Do not write or read vectors directly to or from the vector table.

List of MS-DOS Interrupts

Interrupt		Description
Hex	Dec	
20H	32	Program terminate
21H	33	Function request
22H	34	Terminate address
23H	35	CONTROL-C exit address
24H	36	Fatal error abort address
25H	37	Absolute disk read
26H	38	Absolute disk write
27H	39	Terminate but stay resident

Program Terminate

Interrupt 20H

Causes the current process to terminate and returns control to its parent process. All open file handles are closed and the disk buffer is written to disk. All files that have changed in length should be closed before issuing this interrupt. (See Function Calls 10H and 3EH for descriptions of the Close File function calls.)

The following exit addresses are restored from the Program Segment Prefix:

Exit Address	Offset
Program Terminate	0AH
CONTROL-C	0EH
Critical Error	12H

Note: Interrupt 20H is provided for compatibility with earlier versions of MS-DOS. New programs should use Function Call 4CH, Terminate a Process.

Entry Conditions:

CS = *segment address of Program Segment Prefix*

Macro Definition:

```
terminate macro
      int 20H
      endm
```

Example:

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
      INT 20H
;There is no return from this interrupt
```

Function Request

Interrupt 21H

See "Function Calls" later in this chapter for a description of the MS-DOS system functions.

Entry Conditions:

AH = *function call number*
Other registers as specified in individual function.

Exit Conditions:

As specified in individual function.

Example:

To call the Get Time function:

```
mov             ah,2CH          ;Get Time is Function 2CH
int             21H             ;THIS INTERRUPT
```

Terminate Address

Interrupt 22H

When a program terminates, control transfers to the address at offset 0AH of the Program Segment Prefix. This address is copied into the Program Segment Prefix from the Interrupt 22H vector when the segment is created. You can set this address using Function Call 25H.

CONTROL-C Exit Address

Interrupt 23H

If the user types CONTROL-C during keyboard input or display output, control transfers to the Interrupt 23H vector in the interrupt table. This address is copied into the Program Segment Prefix from the Interrupt 23H vector when the segment is created. You can set this address using Function Call 25H.

If the CONTROL-C routine saves all registers, it can end with an IRET instruction (return from interrupt) to continue program execution. When the interrupt occurs, all registers are set to the value they had when the original call to MS-DOS was made. There are no restrictions on what a CONTROL-C handler can do (including MS-DOS function calls) as long as the registers are unchanged if IRET is used.

If Function 09H or 0AH (Display String or Buffered Keyboard Input) is interrupted by CONTROL-C, the 3 byte sequence 03H-0DH-0AH (ETX-CR-LF) is sent to the display and the function resumes at the beginning of the next line.

If the program creates a new segment and loads a second program that changes the CONTROL-C address, termination of the second program restores the CONTROL-C address to the value it had before execution of the second program.

Fatal Error Abort Address

Interrupt 24H

If a fatal disk error occurs during execution of one of the disk I/O function calls, control transfers to the Interrupt 24H vector in the vector table. This address is copied into the Program Segment Prefix from the Interrupt 24H vector when the segment is created. You can set this address using Function Call 25H.

BP:SI contains the address of a Device Header Control Block from which additional information can be retrieved.

Interrupt 24H is not issued if the failure occurs during execution of Interrupt 25H (Absolute Disk Read) or Interrupt 26H (Absolute Disk Write). These errors are usually handled by the MS-DOS error routine in COMMAND.COM that retries the disk operation and then gives the user the choice of aborting, retrying the operation, or ignoring the error.

Entry Conditions:

BP:SI = *Device Header Control Block address*

Error Returns:

When an error handling program gains control from Interrupt 24H, the AX and DI registers contains codes that describe the error. If Bit 7 of AH is 1, either the error is a bad image of the File Allocation Table or an error occurred on a character device. The device header passed in BP:SI can be examined to determine which case exists. If the attribute byte high-order bit indicates a block device, then the error was a bad FAT. Otherwise, the error is on a character device.

The following are error codes for Interrupt 24H:

Error Code	Description
0	Attempt to write on write-protected disk
1	Unknown unit
2	Drive not ready
3	Unknown command
4	Data Error
5	Bad request structure length
6	Seek error
7	Unknown media type
8	Sector not found
9	Printer out of paper
A	Write fault
B	Read fault
C	General failure

The user stack will be in effect and, from top to bottom, will contain the following:

IP CS FLAGS	MS-DOS registers from issuing INT 24H
AX BX CX DX SI DI BP DS ES	User registers at time of original INT 21H request
IP CS FLAGS	From the original INT 21H from the user to MS-DOS

If an IRET is executed, the registers are set to cause MS-DOS to respond according to the contents of AL as follows:

AL = 0	ignore the error
= 1	retry the operation
= 2	terminate the program via 23H

Notes:

- This exit is taken only for disk errors occurring during an Interrupt 21H. It is not used for errors during Interrupts 25H or 26H.
- This routine is entered in a disabled state.
- The SS, SP, DS, ES, BX, CX, and DX registers must be reserved.
- This interrupt handler should not use MS-DOS function calls. If necessary, it can use calls 01H through 0CH. Use of any other call will destroy the MS-DOS stack and will leave MS-DOS in an unpredictable state.
- The interrupt handler must not change the contents of the device header.
- If the interrupt handler has its own error management routine, rather than returning to MS-DOS, it should restore the application program's registers from the stack, remove all but the last 3 words on the stack, and then issue an IRET. This will cause a return to the program immediately after the INT 21H that experienced the error. Note that if this is done, MS-DOS will be in an unstable state until a function call higher than 0CH is issued.

Absolute Disk Read

Interrupt 25H

Transfers control to MS-DOS. The number of sectors specified in CX is read from the disk to the Disk Transfer Address.

This call destroys all registers except the segment registers. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still on the stack upon return. This is necessary because data is passed back in the current flags. Be sure to pop the stack upon return to prevent uncontrolled growth.

Entry Conditions:

AL = *drive number (0 = A, 1 = B, etc.)*
DS:BX = *Disk Transfer Address*
CX = *number of sectors to read*
DX = *beginning relative sector*

Exit Conditions:

Carry set:
    AL = *error code*
Carry not set:
    Operation was successful.

Error Returns:

Error codes are the same as for Interrupt 24H.

Macro Definition:

```
abs_disk_read macro disk,buffer,num_sectors,first_sector

      mov       al,disk
      mov       bx,offset buffer
      mov       cx,num_sectors
      mov       dx,first_sector
      int       25H
      popf
      endm
```

Example:

The following program copies the contents of a single sided disk in Drive A to the disk in Drive B. It uses a buffer of 32K bytes.

```
prompt          db "Source in A, target in B",13,10
                db "Any key to start. $"
start           dw 0
buffer          db 64 dup (512 dup (?)) ;64 sectors
                .
                .
int_25H:        display prompt              ;see Function 09H
read_kbd                                    ;see Function 08H
                mov cx,5                    ;copy 5 groups of
                                            ;64 sectors
copy:           push cx                     ;save the loop counter
                abs_disk_read 0,buffer,64,start   ;THIS INTERRUPT
                abs_disk_write 1,buffer,64,start  ;see INT 26H
                add start, 64               ;do the next 64 sectors
                pop cx                      ;restore the loop counter
                loop copy
```

Absolute Disk Write

Interrupt 26H

Transfers control to the MS-DOS BIOS. The number of sectors specified in CX is written from the Disk Transfer Address to the disk.

This call destroys all registers except the segment registers. Be sure to save any registers your program uses before issuing the interrupt.

The system pushes the flags at the time of the call; they are still on the stack upon return. This is necessary because data is passed back in the current flags. Be sure to pop the stack upon return to prevent uncontrolled growth.

Entry Conditions:

AL = *drive number (0 = A, 1 = B, etc.)*
DS:BX = *Disk Transfer Address*
CX = *number of sectors to write*
DX = *beginning relative sector*

Exit Conditions:

Carry set:
    AL = *error code*
Carry not set:
    Operation was successful.

Error Returns:

Error codes are the same as for Interrupt 24H.

Macro Definition:

```
abs_disk_write macro disk,buffer,num_sectors,first_sector
mov   al,disk
mov   bx,offset buffer
mov   cx,num_sectors
mov   dx,first_sector
int   26H
popf
endm
```

Example:

The following program copies the contents of a single sided disk in Drive A to the disk in Drive B, verifying each write. It uses a buffer of 32K bytes.

```
off       equ       0
on        equ       1
          .
          .
prompt    db        "Source in A, target in B",13,10
          db        "Any key to start. $"
start     dw        0
buffer    db        64 dup (512 dup(?));64 sectors
          .
          .
int_26H:  display prompt          ;see Function 09H
          read_kbd                ;see Function 08H
          verify on               ;see Function 2EH
          mov   cx,5              ;copy 5 groups of 64
                                  ;sectors
copy:     push   cx               ;save the loop counter
          abs_disk_read0,buffer,64,start     ;see INT 25H
          abs_disk_write 1, buffer,64,start  ;THIS INTERRUPT
          add start,64            ;do the next 64 sectors
          pop cx                  ;restore the loop counter
          loop copy
          verify off              ;see Function 2EH
```

Terminate But Stay Resident

Interrupt 27H

Used to make a piece of code remain resident in the system after its termination. This call is typically used in **.COM** files to allow some device-specific interrupt handler to remain resident to process asynchronous interrupts.

When Interrupt 27H is executed, the program terminates but is treated as an extension of MS-DOS. It remains resident and is not overlaid by other programs when it terminates.

This interrupt provides compatibility with earlier versions of MS-DOS. New programs should use the function 31H, Keep Process.

Entry Conditions:

CS:DX = *first byte following last byte of code in the program*

Macro Definition:

```
stay_resident macro last_instruc
mov   dx,offset last_instruc
inc   dx
int   27H
endm
```

Example:

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
mov   DX,LastAddress
int   27H
;There is no return from this interrupt
```

Function Calls

Categories of Calls

The MS-DOS function calls are divided into 2 groups: old and new. The old calls, Functions 00H through 2EH, are included in this version of MS-DOS to provide compatibility with earlier versions. The new calls, Functions 2FH through 57H, should be used in new programs instead of the old calls wherever possible. Programs that use the new calls cannot be run on earlier versions of MS-DOS.

The function calls can be divided into the following categories:

00H-12H	Old character device I/O
13H-24H	Old file management
25H-26H	Old non-device functions
27H-29H	Old file management
2AH-2EH	Old non-device functions
2FH-38H	New function group
39H-3BH	Directory group
3CH-46H	New file management group
47H	Directory group
48H-4BH	New memory management group
4CH-4FH	New function group
54H-57H	New function group

Error Codes

Many of the function calls in the new group (2FH-57H) return with the carry flag reset if the operation was successful. If the carry flag is set, then an error occurred and register AX contains the binary error return code. These codes are as follows:

Code	Error
1	Invalid function number
2	File not found
3	Path not found
4	Too many open files (no handles left)
5	Access denied
6	Invalid handle
7	Memory control blocks destroyed
8	Insufficient memory
9	Invalid memory block address
10	Invalid environment
11	Invalid format
12	Invalid access code
13	Invalid data
15	Invalid drive was specified
16	Attempted to remove the current directory
17	Not same device
18	No more files

File Handles

Some of the new calls use a "file handle" to identify a file or device. A handle is a 16-bit binary value that is returned in register AX when you create or open a file or device using the new calls. This handle should be used in subsequent references to the file.

ASCIIZ Strings

Some calls require an ASCIIZ string in one of the registers as an entry condition. An ASCIIZ string is an ASCII string followed by a byte of binary zeroes. The string consists of an optional drive specifier followed by a directory path and (in some cases) a filename. The following string, if followed by a byte of zeroes, is an example:

```
B:\LEVEL1\LEVEL2\FILE
```

Calling MS-DOS Functions

Most of the MS-DOS function calls require input to be passed to them in registers. After setting the proper register values, the function may be invoked in one of the following ways:

1. Place the function number in AH and execute a long call to offset 50H in your Program Segment Prefix. Note that programs using this method will not operate correctly on earlier versions of MS-DOS.

2. Place the function number in AH and issue Interrupt 21H. All of the examples in this chapter use this method.

3. An additional method exists for programs that were written with different calling conventions. This method should be avoided for all new programs. The function number is placed in the CL register and other registers are set according to the function specification. Then an intrasegment call is made to location 5 in the current code segment. That location contains a long call to the MS-DOS function dispatcher. Register AX is always destroyed if this method is used; otherwise, it is the same as normal function calls. Note that this method is valid only for Function Requests 00H through 024H.

CP/M™ -Compatible Calling Sequence

A different sequence can be used for programs that must conform to CP/M calling conventions:

1. Move any required data into the appropriate registers (just as in the standard sequence).

2. Move the function number into the CL register.

3. Execute an intrasegment call to location 5 in the current code segment.

This method can be used only with the functions 00H through 24H that do not pass a parameter in AL. Register AX is always destroyed when a function is called in this manner.

Treatment Of Registers

When MS-DOS takes control, it switches to an internal stack. All registers are saved except AX and those registers used to return information. The calling program's stack must be large enough to accommodate the interrupt system. It should be at least 80H bytes, in addition to the program's needs.

MS-DOS Function Calls in Numeric Order

Function Number	Function Name
00H	Terminate Program
01H	Read Keyboard and Echo
02H	Display Character
03H	Auxiliary Input
04H	Auxiliary Output
05H	Print Character
06H	Direct Console I/O
07H	Direct Console Input
08H	Read Keyboard
09H	Display String
0AH	Buffered Keyboard Input
0BH	Check Keyboard Status
0CH	Flush Buffer, Read Keyboard
0DH	Reset Disk
0EH	Select Disk
0FH	Open File
10H	Close File
11H	Search for First Entry
12H	Search for Next Entry
13H	Delete File
14H	Sequential Read
15H	Sequential Write
16H	Create File
17H	Rename File
19H	Current Disk
1AH	Set Disk Transfer Address
21H	Random Read
22H	Random Write
23H	File Size
24H	Set Relative Record

25H	Set Interrupt Vector
27H	Random Block Read
28H	Random Block Write
29H	Parse File Name
2AH	Get Date
2BH	Set Date
2CH	Get Time
2DH	Set Time
2EH	Set/Reset Verify Flag
2FH	Get Disk Transfer Address
30H	Get Version Number
31H	Keep Process
33H	CONTROL-C Check
35H	Get Interrupt Vector
36H	Get Disk Free Space
38H	Return Country-Dependent Information
39H	Create Subdirectory
3AH	Remove a Directory Entry
3BH	Change the Current Directory
3CH	Create a File
3DH	Open a File
3EH	Close a File Handle
3FH	Read From a File or Device
40H	Write to a File or Device
41H	Delete a Directory Entry
42H	Move a File Pointer
43H	Change Attributes
44H	I/O Control for Devices
45H	Duplicate a File Handle
46H	Force a Duplicate of a File Handle
47H	Return Text of Current Directory
48H	Allocate Memory
49H	Free Allocated Memory
4AH	Modify Allocated Memory Blocks
4BH	Load and Execute a Program
4CH	Terminate a Process
4DH	Retrieve the Return Code of a Child
4EH	Find Matching File
4FH	Find Next Matching File
54H	Return Current Setting of Verify
56H	Move a Directory Entry
57H	Get or Set a File's Date and Time

MS-DOS Function Calls in Alphabetic Order

Function Name	Number
Allocate Memory	48H
Auxiliary Input	03H
Auxiliary Output	04H
Buffered Keyboard Input	0AH
Change Attributes	43H
Change the Current Directory	3BH
Check Keyboard Status	0BH
Close a File Handle	3EH
Close File	10H
CONTROL-C Check	33H
Create a File	3CH
Create File	16H
Create Subdirectory	39H
Current Disk	19H
Delete a Directory Entry	41H
Delete File	13H
Direct Console Input	07H
Direct Console I/O	06H
Display Character	02H
Display String	09H
Duplicate a File Handle	45H
File Size	23H
Find Matching File	4EH
Flush Buffer, Read Keyboard	0CH
Force a Duplicate of a File Handle	46H
Free Allocated Memory	49H
Get Date	2AH
Get Disk Free Space	36H
Get Disk Transfer Address	2FH
Get Version Number	30H
Get Interrupt Vector	35H
Get Time	2CH
Get or Set a File's Date or Time	57H
I/O Control for Devices	44H
Keep Process	31H
Load and Execute a Program	4BH
Modify Allocated Memory Blocks	4AH
Move a Directory Entry	56H
Move a File Pointer	42H

Open a File	3DH
Open File	0FH
Parse File Name	29H
Print Character	05H
Random Block Read	27H
Random Block Write	28H
Random Read	21H
Random Write	22H
Read From a File or Device	3FH
Read Keyboard	08H
Read Keyboard and Echo	01H
Remove a Directory Entry	3AH
Rename File	17H
Reset Disk	0DH
Retrieve the Return Code of a Child	4DH
Return Current Setting of Verify	54H
Return Country-Dependent Information	38H
Return Text of Current Directory	47H
Search for First Entry	11H
Search for Next Entry	12H
Select Disk	0EH
Sequential Read	14H
Sequential Write	15H
Set Date	2BH
Set Disk Transfer Address	1AH
Set Relative Record	24H
Set Time	2DH
Set Interrupt Vector	25H
Set/Reset Verify Flag	2EH
Find Next Matching File	4FH
Terminate a Process	4CH
Terminate Program	00H
Write to a File or Device	40H

Abort

Terminate Program — Function Call 00H

Terminates a program. This function is called by Interrupt 20H, and performs the same processing.

The following exit addresses are restored from the specified offsets in the Program Segment Prefix:

Program terminate	0AH
CONTROL-C	0EH
Critical error	12H

All file buffers are written to disk. Be sure to close all files that have been changed in length before calling this function. If a changed file is not closed, its length will not be recorded correctly in the disk directory. See Function Call 10H for a description of the Close File call.

Entry Conditions:

AH = 00H
CS = *segment address of the Program Segment Prefix*

Macro Definition:

```
terminate_program       macro
                        xor         ah,ah
                        int         21H
                        endm
```

Example:

```
;CS must be equal to PSP values given at program start
;(ES and DS values)
     mov   ah,0
     int   21H
;There are no returns from this interrupt
```

StdConInput

Keyboard Input — Function Call 01H

Waits for a character to be typed at the keyboard, then echoes the character to the display and returns it in AL. If the character is CONTROL-C, Interrupt 23H is executed.

Entry Conditions:

AH = 01H

Exit Conditions:

AL = *character typed*

Macro Definition:

```
read_kbd_and_echo       macro
                        mov ah, 01H
                        int 21H
                        endm
```

Example:

The following program displays and prints characters as they are typed. If ENTER is pressed, the program sends a Line-Feed/Carriage-Return to both the display and the printer.

```
func_01H:   read_kbd_and_echo
            print_char      al          ;THIS FUNCTION
            cmp             al,0DH      ;see Function 05H
            jne             func_01H    ;is it a CR?
            print_char      10          ;no, print it
            display_char    10          ;see Function 02H
            jmp             func_01H    ;see Function 02H
                                        ;get another character
```

StdConOutput

Display Character — Function Call 02H

Displays a character on the video screen. If a CONTROL-C is typed, Interrupt 23H is executed.

Entry Conditions:

AH = 02H
DL = *character to display*

Macro Definition:

```
display_chr     macro       character
                mov         dl,character
                mov         ah,02H
                int         21H
                endm
```

Example:

The following program converts lowercase characters to uppercase before displaying them.

```
func_02H:       read_kbd                        ;see Function 08H
                cmp         al,"a"
                jl          uppercase           ;don't convert
                cmp         al,"z"
                jg          uppercase           ;don't convert
                sub         al,20H              ;convert to ASCII code
                                                ;for uppercase
uppercase:      display_char al                 ;THIS FUNCTION
                jmp         func_02H            ;get another character
```

AuxInput

Auxiliary Input — Function Call 03H

Waits for a character from the auxiliary input device, and then returns the character in AL. No status or error code is returned.

If CONTROL-C is typed at console input, Interrupt 23H is executed.

Entry Conditions:

AH = 03H

Exit Conditions:

AL = *character returned*

Macro Definition:

```
aux_input macro
          mov ah,03H
          int 21H
          endm
```

Example:

The following program prints characters as they are received from the auxiliary device. It stops printing when an end of file character (ASCII 26, or CONTROL-Z) is received.

```
func_03H:       aux_input          ;THIS FUNCTION
                cmp al,1AH         ;end of file?
                je  continue       ;yes, all done
                print_char  al     ;see Function 05H
                jmp func_03H       ;get another character
continue        ret
```

AuxOutput

Auxiliary Output — Function Call 04H

Outputs a character to the auxiliary device. No status or error code is returned.

If CONTROL-C is typed at console input, Interrupt 23H is executed.

Entry Conditions:

AH = 04H
DL = *character to output*

Macro Definition:

```
aux_output      macro       character
                mov         dl,character
                mov         ah,04H
                int         21H
                endm
```

Example:

The following program gets a maximum of 80 bytes from the keyboard, sending each to the auxiliary device. It stops when a null string (CR only) is typed.

```
string          db      81 dup(?)               ;see Function 0AH
                .
                .
func_04H:       get_string 80,string            ;see Function 0AH
                cmp string[1],0                 ;null string?
                je  continue                    ;yes, all done
                mov cx, word ptr string[1]      ;get string length
                mov bx,0                        ;set index to 0
sent_it:        aux_output string[bx+2]         ;THIS FUNCTION
                inc  bx                         ;bump index
                loop send_it                    ;send another character
                jmp func_04H                    ;get another string
continue:       .
                .
```

PrinterOutput

Print Character — Function Call 05H

Outputs a character to the printer. If CONTROL-C is typed at console input, Interrupt 23H is executed.

Entry Conditions:

AH = 05H
DL = *character for printer*

Macro Definition:

```
print_char      macro           character
                mov             dl, character
                mov             ah,05H
                int             21H
                endm
```

Example:

The following program prints a walking test pattern on the printer. It stops if CONTROL-C is pressed.

```
line_num        db              0
                .
                .
func_05H:       mov             cx,60           ;print 60 lines
start_line:     mov             bl,33           ;first printable ASCII
                                                ;character(!)
                add             bl,line_num     ;to offset 1 character
                push            cx              ;save number-of-lines counter
                mov             cx,80           ;loop counter for line
print_it:       print_char bl                   ;THIS FUNCTION
                inc             bl              ;move to next ASCII character
                cmp             bl,126          ;last printable ASCII
                                                ;character (3/4)
                jl              no_reset        ;not there yet
                mov             bl,33           ;start over with (!)
no_reset:       loop            print_it        ;print another character
                                print_char 13   ;carriage return
                                print_char 10   ;line feed
                inc             line_num        ;to offset 1st char. of line
                pop             cx              ;restore #-of-lines counter
                loop            start_line      ;print another line
```

Conio

Direct Console I/O — Function Call 06H

Returns a keyboard input character if one is ready, or outputs a character to the video display. No check for CONTROL-C is made on the character.

Entry Conditions:

AH = 06H
DL = FFH: return *character typed at the keyboard*; if available
DL ≠ FFH: display character in DL

Exit Conditions:

If DL = FFH on entry:
Zero flag set and AL = 00H if no key was pressed
Zero flag not set and AL = *keyboard input character*, if available

If DL ≠ FFH on entry:
Character in DL is on screen

Macro Definition:

```
dir_console_io          macro       switch
                        mov         dl,switch
                        mov         ah,06H
                        int         21H
                        endm
```

Example:

The following program sets the system clock to 0 and continuously displays the time. When any character is typed, the display stops changing; when any character is typed again, the clock is reset to 0 and the display starts again.

```
time            db "00:00:00.00",13,    ;see Function 09H
                10,"$"                  ;for explanation of $
ten             db 10
                .
                .
func_06H:       set_time 0,0,0,0        ;see Function 2DH
read_clock:     get_time                ;see Function 2CH
                convert  ch,ten,time    ;see end of chapter
                convert  cl,ten,time[3] ;see end of chapter
                convert  dh,ten,time[6] ;see end of chapter
                display  time           ;see Function 09H
                dir_console_io 0FFH     ;THIS FUNCTION
                jne      stop           ;yes, stop timer
                jmp      read_clock     ;no, keep timer
                                        ;running
stop:           read_kbd                ;see Function 08H
                jmp      func_06H       ;start over
```

ConInput

Direct Console Input — Function Call 07H

Waits for a character to be typed at the keyboard, and then returns the character. This call does not echo the character or check for CONTROL-C. (For a keyboard input function that echoes or checks for CONTROL-C, see Function Call 01H or 08H.)

Entry Conditions:

AH = 07H

Exit Conditions:

AL = *character from keyboard*

Macro Definition:

```
Dir_console_input       macro
                        mov         ah,07H
                        int         21H
                        endm
```

Example:

The following program prompts for a password (8 characters maximum) and places the characters into a string without echoing them.

```
password        db 8 dup(?)
prompt          db "Password: $"        ;see Function 09H for an
                                        ;explanation of $
                .
                .
func_07H:       display prompt          ;see Function 09H
                mov cx, 8               ;maximum length of
                                        ;password
                xor bx,bx               ;so BL can be used as
                                        ;index
get_pass:       dir_console_input       ;THIS FUNCTION
                cmp al,0DH              ;was it a CR?
                je  continue            ;yes, all done
                mov password[bx],al     ;no, put character in
                                        ;string
                inc bx                  ;bump index
                loop get_pass           ;get another character
continue:       .                       ;BX has length of
                .                       ;password + 1
```

ConInputNoEcho

Read Keyboard — Function Call 08H

Waits for a character to be typed at the keyboard, and then returns it in AL. If CONTROL-C is pressed, Interrupt 23H is executed. This call does not echo the character. (For a keyboard input function that echoes the character and checks for CONTROL-C, see Function Call 01H.)

Entry Conditions:

AH = 08H

Exit Conditions:

AL = *character from keyboard*

Macro Definition:

```
read_kbd        macro
                mov         ah,08H
                int         21H
                endm
```

Example:

The following program prompts for a password (8 characters maximum) and places the characters into a string without echoing them.

```
password        db 8 dup(?)
prompt          db "Password: $"          ;see Function 09H
                                          ;for explanation of $
                .
                .
func_08H:       display prompt            ;see Function 09H
                mov cx,8                  ;maximum length of
                                          ;password
                xor  bx,bx                ;BL can be an index
get_pass:       read_kbd                  ;THIS FUNCTION
                cmp   al,0DH              ;was it a CR?
                je   continue             ;yes, all done
                mov   password[bx],al     ;no, put char. in string
                inc   bx                  ;bump index
                loop   get_pass           ;get another character
continue:       .                         ;BX has length of
                .                         ;password + 1
```

ConStringOutput

Display String — Function Call 09H

Displays a string of characters. Each character is checked for CONTROL-C. If a CONTROL-C is detected, an interrupt 23H is executed.

Entry Conditions:

AH = 09H
DS:DX = *pointer to a string to be displayed terminated by a $ (24H)*

Macro Definition:

```
display     macro       string
            mov         dx,offset string
            mov         ah,09H
            int         21H
            endm
```

Example:

The following program displays the hexadecimal code of the key that is typed.

```
table       db          "0123456789ABCDEF"
sixteen     db          16
result      db          " - 00H"            ;see text for
crlf        db          13,10, "$"          ;explanation of $
            .
            .
func_09H:read_kbd_and_echo                  ;see Function 01H
            convert al,sixteen,result[3]    ;see end of chapter
            display result                  ;THIS FUNCTION
            jmp         func_09H            ;do it again
```

ConStringInput

Buffered Keyboard Input Function Call 0AH

Waits for characters to be typed, reads characters from the keyboard, and places them in an input buffer until [ENTER] is pressed. Characters are placed in the buffer beginning at the third byte. If the buffer fills to 1 less than the maximum specified, then additional keyboard input is ignored and ASCII 7 (BEL) is sent to the display until [ENTER] is pressed.

The string can be edited as it is being entered. If CONTROL-C is typed, Interrupt 23H is executed.

The input buffer pointed to by DS:DX must be in this form:

byte 1 — Maximum number of characters in buffer, including the carriage return (1-255; you set this value).

byte 2 — Actual number of characters typed, not including the carriage return (the function sets this value).

bytes 3-n — Buffer; must be at least as long as the number in byte 1.

Entry Conditions:

AH = 0AH
DS:DX = *pointer to an input buffer (see above)*

Exit Conditions:

DS:[DS + 1] = *number of characters received, excluding the carriage return*

Macro Definition:

```
get_string      macro       limit,string
                mov         dx,offset string
                mov         string,limit
                mov         ah,0AH
                int         21H
                endm
```

Example:

The following program gets a 16-byte (maximum) string from the keyboard and fills a 24 line by 80 character screen with it.

```
buffer              label       byte
max_length          db          ?                   ;maximum length
chars_entered       db          ?                   ;number of chars
string              db          17 dup (?)          ;16 chars + CR
strings_per_line    dw          0                   ;how many strings
                                                    ;fit on line
crlf                db          13,10,"$"
                    .
                    .
func_0AH:           get_string16,buffer             ;THIS FUNCTION
                    xor         bx,bx               ;so byte can be
                                                    ;used as index
                    mov         bl,chars_entered    ;get string length
                    mov         buffer[bx+2],"$"    ;see Function 09H
                    mov         al,50H              ;columns per line
                    cbw
                    div         chars_entered       ;times string fits
                                                    ;on line
                    xor         ah,ah               ;clear remainder
                    mov         strings_per_line,ax ;save col. counter
                    mov         cx,24               ;row counter
display_screen:     push        cx                  ;save it
                    mov         cx,strings_per_line ;get col. counter
display_line:       display string                  ;see Function 09H
                    loop        display_line
                    display     crlf                ;see Function 09H
                    pop         cx                  ;get line counter
                    loop        display_screen      ;display 1 more line
```

ConInputStatus

Check Keyboard Status Function Call 0BH

Checks to see if a character is available in the type-ahead buffer. If CONTROL-C is in the buffer, Interrupt 23H is executed.

Entry Conditions:

AH = 0BH

Exit Conditions:

If AL = FFH, there are characters in the type-ahead buffer.

If AL = 00H, there are no characters in the type-ahead buffer.

Macro Definition:

```
check_kbd_status        macro
                        mov         ah,0BH
                        int         21H
                        endm
```

Example:

The following program continuously displays the time until any key is pressed.

```
time            db   "00:00:00.00", 13,10,"$"
ten             db   10
                .
                .
func_0BH:       get_time                ;see Function 2CH
                convert ch,ten, time    ;see end of chapter
                convert cl,ten, time[3] ;see end of chapter
                convert dh,ten, time[6] ;see end of chapter
                convert dl,ten, time[9] ;see end of chapter
                display time            ;see Function 09H
                check_kbd_status        ;THIS FUNCTION
                cmp     al,0FFH         ;has a key been typed?
                je      all_done        ;yes, go home
                jmp     func_0BH        ;no, keep displaying
all_done:       ret                     ;time
```

ConInputFlush

Flush Buffer, Read Keyboard — Function Call 0CH

Empties the keyboard type-ahead buffer. Further processing depends on the value in AL when the function is called:

01H, 06H, 07H, 08H, or 0AH — The corresponding input system call is executed.

Any other value — No further processing is done.

Entry Conditions:

AH = 0CH
AL = *function code*
01H, 06H, 07H, 08H, or 0AH = call corresponding function
Any other value = perform no further processing

Exit Conditions:

If AL = 00H, type-ahead buffer was flushed; no other processing was performed.

Macro Definition:

```
flush_and_read_ kbd          macro       switch
                             mov         al,switch
                             mov         ah,0CH
                             int         21H
                             endm
```

Example:

The following program both displays and prints characters as they are typed. If ENTER is pressed, the program sends a Carriage-Return/Line-Feed to both the display and the printer. (The example assumes that a CONTROL-C processing routine has been set up before the loop is entered.)

```
func_0CH:   flush_and_read_kbd 1          ;THIS FUNCTION
            print_char     al             ;see Function 05H
            cmp            al,0DH         ;is it a CR?
            jne            func_ 0CH      ;no, print it
            print_ char    10             ;see Function 05H
            display_ char  10             ;see Function 02H
            jmp            func_ 0CH      ;get another character
```

ResetDisk

Reset Disk — Function Call 0DH

Ensures that the internal buffer cache matches the disks in the drives. This call flushes all file buffers. All buffers that have been modified are written to disk and all buffers in the internal cache are marked as free. Directory entries are not updated; you must close files that have changed in order to update their directory entries (see Function Call 10H, Close File).

This function need not be called before a disk change if all files that were written to have been closed. It is generally used to force a known state of the system; CONTROL-C interrupt handlers should call this function.

Entry Conditions:

AH - 0DH

Macro Definition:

```
reset_disk      macro       disk
                mov         ah,0DH
                int         21H
                endm
```

Example:

```
mov             ah,0DH
int             21H
;There are no errors returned by this call.
```

SelectDisk

Select Disk — Function Call 0EH

Selects the specified drive as the default drive.

Entry Conditions:

AH = 0EH
DL = *new default drive number (0 = A, 1 = B, etc.)*

Exit Conditions:

AL = *number of logical drives*

Macro Definition:

```
select_disk     macro       disk
                mov         dl,disk[-64]
                mov         ah,0EH
                int         21H
                endm
```

Example:

The following program selects the drive not currently selected in a 2-drive system.

```
func_0EH:       current_disk                ;see Function 19H
                cmp     al,00H              ;drive A selected?
                je      select_ b           ;yes, select B
                select_disk "A"             ;THIS FUNCTION
                jmp     continue
select_ b:      select_disk "B"             ;THIS FUNCTION
continue:
```

OpenFile

Open File — Function Call 0FH

Opens a File Control Block (FCB) for the named file, if the file is found in the disk directory. The FCB is filled in as follows:

- If the drive code in the file specification is 0 (default drive), it is changed to the number of the actual disk used (1 = A, 2 = B, etc.). This lets you change the default drive without interfering with subsequent operations on this file.
- The current block field (offset 0CH) is set to zero.
- The record size (offset 0EH) is set to the system default of 128.
- The file size (offset 10H), date of last write (offset 14H), and time of last write (offset 16H) are set from the directory entry.

  Before performing a sequential disk operation on the file, you must set the current record field (offset 20H). Before performing a random disk operation on the file, you must set the relative record field (offset 21H). If the default record size (128 bytes) is not correct, set it to the correct length.

Entry Conditions:

AH = 0FH
DS:DX = *pointer to an unopened FCB for the file*

Exit Conditions:

If AL = 00H, the directory entry was found.
If AL = FFH, the directory entry was not found.

Macro Definition:

```
open    macro   fcb
        mov     dx,offset fcb
        mov     ah,0FH
        int     21H
        endm
```

Example:

The following program prints the file names TEXTFILE.ASC that is on the disk in Drive B. If a partial record is in the buffer at end of file, the routine that prints the partial record prints characters until it encounters an end of file mark (ASCII 26, or CONTROL-Z).

```
fcb             db        2, "TEXTFILEASC"
                db        25 dup (?)
buffer          db        128 dup (?)
                .
                .
func_ 0FH:      set_dta   buffer          ;see Function 1AH
                open      fcb             ;THIS FUNCTION
read_ line:     read_seq  fcb             ;see Function 14H
                cmp       al,02H          ;end of file?
                je        all_done        ;yes, go home
                cmp       al,00H          ;more to come?
                jg        check_more      ;no, check for partial
                                          ;Record
                mov       cx,128          ;yes, print the buffer
                xor       si,si           ;set index to 0
print_it:       print_char buffer[si]     ;see Function 05H
                inc       si              ;bump index
                loop      print_it        ;print next character
                jmp       read_line       ;read another record
check_ more:    cmp       al,03H          ;part. record to print?
                jne       all_ done       ;no
                mov       cx,128          ;yes, print it
                xor       si,si           ;set index to 0
find_eof:       cmp       buffer[si],26   ;end of file mark?
all_ done:      close     fcb             ;see Function 10H
```

Close File

Close File — Function Call 10H

Closes an open file and updates the directory information on that file. This function must be called after a file is changed to update the directory entry.

If a directory entry for the file is found, the location of the file is compared with the corresponding entries in the File Control Block (FCB). The directory is updated, if necessary, to match the FCB.

Entry Conditions:

AH = 10H
DS:DX = *pointer to the open FCB of the file to close*

Exit Conditions:

If AL = 00H, the directory entry was found.
If AL = FFH, no directory entry was found.

Macro Definition:

```
close   macro   fcb
        mov  dx,offset fcb
        mov  ah,10H
        int  21H
        endm
```

Example:

The following program checks the first byte of the file named MOD1.BAS in Drive B to see if it is FFH, and prints a message if it is.

```
message     db          "Not saved in ASCII format",13,10,"$"
fcb         db          2,"MOD1    BAS"
            db          25 dup (?)
buffer      db          128 dup (?)
            .
            .
func_10H:   set_dta     buffer          ;see Function 1AH
            open        fcb             ;see Function 0FH
            read_ seq   fcb             ;see Function 14H
            cmp         buffer,0FFH     ;is first byte FFH?
            jne         all_ done       ;no
            display     message         ;see Function 09H
all_ done:  close       fcb             ;THIS FUNCTION
```

DirSearchFirst

Search for First Entry — Function Call 11H

Searches the disk directory for the first name that matches the filename in the FCB. The name can have the ? wildcard character to match any character. To search for hidden or system files, DS:DX must point to the first byte of the extended FCB prefix.

If a directory entry for the filename in the FCB is found, an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.

If an extended FCB is pointed to by DS:DX, the following search pattern is used:

- If the attribute byte (offset FCB-1) is zero, only normal file entries are found. Entries for the volume label, subdirectories, hidden files, and system files will not be returned.
- If the attribute field is set for hidden or system files, or directory entries, it is considered an inclusive search. All normal file entries plus all entries matching the specified attributes are returned. To look at all directory entries except the volume label, the attribute byte can be set to hidden + system + directory (all 3 bits set).
- If the attribute field is set for the volume label, it is considered an exclusive search, and only the volume label entry is returned.

Entry Conditions:

AH = 11H
DS:DX = *pointer to the unopened FCB of the file for which to search*

Exit Conditions:

If AL = 00H, a directory entry was found.
If AL = FFH, no directory entry was found.

Macro Definition:

```
search_first        macro       fcb
                    mov         dx,offset fcb
                    mov         ah,11H
                    int         21H
                    endm
```

Example:

The following program verifies the existence of a file named REPORT.ASM on the disk in Drive B.

```
yes             db      "FILE EXISTS.$"
no              db      "FILE DOES NOT EXIST.$"
fcb             db      2,"REPORT ASM"
                db      25 dup (?)
buffer          db      128 dup (?)
crlf            db      13,10,"$"
                .
                .
func_11H:       set_dta buffer                  ;see Function 1AH
                search_first fcb                ;THIS FUNCTION
                cmp     al,0FFH                 ;directory entry found?
                je      not_there               ;no
                display yes                     ;see Function 09H
                jmp     continue
not_there:      display no                      ;see Function 09H
continue:       display crlf                    ;see Function 09H
```

SearchNext

Search for Next Entry — Function Call 12H

Used after Function Call 11H (Search for First Entry) to find additional directory entries that match a filename that contains wildcard characters. The ? wildcard character in the filename matches any character. This call searches the disk directory for the next matching name. To search for hidden or system files, DS:DX must point to the first byte of the extended FCB prefix.

If a directory entry for the filename in the FCB is found, an unopened FCB of the same type (normal or extended) is created at the Disk Transfer Address.

Entry Conditions:

AH = 12H
DS:DX = *pointer to the unopened FCB of the file for which to search*

Exit Conditions:

If AL = 00H, a directory entry was found.
If AL = FFH, no directory entry was found.

Macro Definition:

```
search_next     macro     fcb
                mov       dx,offset fcb
                mov       ah,12H
                int       21H
                endm
```

Example:

The following program displays the number of files on the disk in Drive B.

```
message         db              "No files",10,13,"$"
files           db              0
ten             db              10
ten             db              10
fcb             db              2,"???????????"
                db              25 dup (?)
buffer          db              128 dup (?)
                .
                .
func_12H:       set_dta buffer                  ;see Function 1AH
                search_first fcb                ;see Function 11H
                cmp             al,0FFH         ;directory entry found?
                je              all_done        ;no, no files on disk
                inc             files           ;yes, increment file
                                                ;counter
search_dir:     search_next  fcb                 ;THIS FUNCTION
                cmp             al,0FFH         ;directory entry found?
                je              done            ;no
                inc             files           ;yes, increment file
                                                ;counter
                jmp             search_dir      ;check again
done:           convert files,ten,message       ;see end of chapter
all_done:       display         message         ;see Function 09H
```

DeleteFile

Delete File — Function Call 13H

Deletes all directory entries that match the filename given in the specified unopened FCB. The filename can contain the ? wildcard character to match any character.

Entry Conditions:

AH = 13H
DS:DX = *pointer to an unopened FCB*

Exit Conditions:

If Al = 00H, a directory entry was found.
If AL = FFH, no directory entry was found.

Macro Definition:

```
delete      macro     fcb
            mov       dx,offset fcb
            mov       ah,13H
            int       21H
            endm
```

Example:

The following program deletes each file on the disk in Drive B that was last written before December 31, 1982.

```
year            dw              1982
month           db              12
day             db              31
files           db              0
ten             db              10
message         db              "NO FILES DELETED.",13,10,"$"
                                                ;see Function 09H for
                                                ;explanation of $
fcb             db              2,"???????????"
                db              25 dup (?)
buffer          db              128 dup (?)
                .
                .
func_13H:       set_dta  buffer                 ;see Function 1AH
                search_first  fcb               ;see Function 11H
                cmp             al,FFH          ;directory entry found?
                je              all_done        ;no, no files on disk
compare:        convert_date buffer             ;see end of chapter
                cmp             cx,year         ;next several lines
                jg              next            ;check date in directory
                cmp             dl,month        ;entry against date
                jg              next            ;above & check next file
                cmp             dh,day          ;if date in directory
                jge             next            ;entry isn't earlier.
                delete          buffer          ;THIS FUNCTION
                inc             files           ;bump deleted-files
                                                ;counter
next:           search_next fcb                 ;see Function 12H
                cmp             al,00H          ;directory entry found?
                je              compare         ;yes, check date
                cmp             files,0         ;any files deleted?
                je              all_done        ;no, display NO FILES
                                                ;message.
                convert files,ten,message       ;see end of chapter
all_done:       display         message         ;see Function 09H
```

SeqRead

Sequential Read — Function Call 14H

Reads a record sequentially. The record pointed to by the current block (offset 0CH) and the current record (offset 20H) fields of the FCB is loaded at the Disk Transfer Address. The current block and current record fields are then incremented.

The record size is set to the value at offset 0EH in the FCB.

Entry Conditions:

AH = 14H
DS:DX = *pointer to the opened FCB of the file to read*

Exit Conditions:

If AL = 00H, the read was completed successfully.
If AL = 01H, end of file was encountered; there was no data in the record.
If AL = 02H, there was not enough room at the Disk Transfer Address to read 1 record; the read was canceled.
If AL = 03H, end of file was encountered; a partial record was read and padded to the record length with zeroes.

Macro Definition:

```
read_seq        macro       fcb
                mov         dx,offset fcb
                mov         ah,14H
                int         21H
                endm
```

Example:

The following program displays the file named TEXTFILE.ASC that is on the disk in Drive B; its function is similar to the MS-DOS TYPE command. If a partial record is in the buffer at end of file, the routine that displays the partial record displays characters until it encounters an end of file mark (ASCII 26, or CONTROL-Z).

```
fcb             db          2,"TEXTFILEASC"
                db          25 dup (?)
buffer          db          128 dup (?),"$"
                .
                .
func_14H:       set_dta buffer                  ;see Function 1AH
                open        fcb                 ;see Function 0FH
read_line:      read_seq    fcb                 ;THIS FUNCTION
                cmp         al,02H              ;end of file?
                je          all_done            ;yes
                cmp         al,02H              ;end of file with
                                                ;partial record?
                jg          check_more          ;yes
                display     buffer              ;see Function 09H
                jmp         read_line           ;get another record
check_more:     cmp         al,03H              ;partial record in buffer?
                jne         all_done            ;no, go home
                xor         si,si               ;set index to 0
find_eof:       cmp         buffer[si]26        ;is character EOF?
                je          all_done            ;yes, no more to display
                display_char buffer [si]        ;see Function 02H
                inc         si                  ;bump index to next
                                                ;character
                jmp         find_eof            ;check next character
all_done:       close       fcb                 ;see Function 10H
```

SeqWrite

Sequential Write — Function Call 15H

Writes a record sequentially. The record pointed to by the current block (offset 0CH) and the current record (offset 20H) fields of the FCB is written from the Disk Transfer Address. The current block and current record fields are then incremented.

The record size is set to the value at offset 0EH in the FCB. If the record size is less than a sector, the data at the Disk Transfer Address is written to a buffer. The buffer is written to disk when it contains a full sector of data, when the file is closed, or when Function Call 0DH (Reset Disk) is issued.

Entry Conditions:

AH = 15H
DS:DX = *pointer to the opened FCB of the file to write*

Exit Conditions:

If AL = 00H, the write was completed successfully.
If AL = 01H, the disk was full; the write was canceled.
If AL = 02H, there was not enough room in the disk transfer segment to write 1 record; the write was canceled.

Macro Definition:

```
write_seq       macro       fcb
                mov         dx,offset fcb
                mov         ah,15H
                int         21H
                endm
```

Example:

The following program creates a file named DIR.TMP on the disk in Drive B that contains the disk number (0 = A, 1 = B, etc.) and filename from each directory entry on the disk.

```
record_size     equ             14              ;offset of Record Size
                                                ;field in FCB
                .
                .
fcb1            db              2,"DIR    TMP"
                db              25 dup (?)
fcb2            db              2,"???????????"
                db              25 dup (?)
buffer          db              128 dup (?)
                .
                .
func_15H:       set_dta         buffer          ;see Function 1AH
                search_first    fcb2            ;see Function 11H
                cmp             al,0FFH         ;directory entry found?
                je              all_done        ;no, no files on disk
                create          fcb1            ;see Function 16H
                mov             fcb1[record_size],12
                                                ;set record size to 12
write_it:       write_seq       fcb1            ;THIS FUNCTION
                search_next     fcb2            ;see Function 12H
                cmp             al,0FFH         ;directory entry found?
                je              all_done        ;no, go home
                jmp             write_it        ;yes, write the record
all_done:       close           fcb1            ;see Function 10H
```

Create

Create File — Function Call 16H

Searches the directory for an empty entry or an existing entry for the filename in the specified FCB.

If an empty directory entry is found, it is initialized to a zero-length file and the Open File function call (0FH) is called. You can create a hidden file by using an extended FCB with the attribute byte (offset FCB-1) set to 2.

If an entry is found for the specified filename, all data in the file is released, making a zero-length file, and the Open File function call (0FH) is issued for the filename. If you try to create a file that already exists, the existing file is erased and a new, empty file is created.

Entry Conditions:

AH = 16H
DS:DX = *pointer to an unopened FCB for the file*

Exit Conditions:

If AL = 00H, an empty directory entry was found.
If AL = FFH, no empty directory entry was available.

Macro Definition:

```
create      macro     fcb
            mov       dx,offset fcb
            mov       ah,16H
            int       21H
            endm
```

Example:

The following program creates a file named DIR.TMP on the disk in Drive B that contains the disk number (0 = A, 1 = B, etc.) and filename from each directory entry on the disk.

```
record_size     equ             14              ;offset of Record Size
                                                ;field of FCB
                .
                .
fcb1            db              2,"DIR  TMP"
                db              25 dup(?)
fcb2            db              2,"???????????"
                db              25 dup (?)
buffer          db              128 dup(?)
                .
                .
func_16H:       set_dta         buffer          ;seeFunction 1AH
                search_first    fcb2            ;see Function 11H
                cmp             al,0FFH         ;directory entry found?
                je              all_done        ;no, no files on disk
                create          fcb1            ;THIS FUNCTION
                mov             fcb1[record_size],12
                                                ;set record size to 12
write_it:       write_seq       fcb1            ;see Function 15H
                search_next     fcb2            ;see Function 12H
                cmp             al,0FFH         ;directory entry found?
                je              all_done        ;no, go home
                jmp             write_it        ;yes, write the record
all_done:       close           fcb1            ;see Function 10H
```

Rename

Rename File — Function Call 17H

Changes the name of a file. The current drive code and filename occupy the usual position in the file's FCB, and are followed by a second filename at offset 11H. (The 2 filenames cannot be the same name.) The disk directory is searched for an entry that matches the first filename, which can contain the ? wildcard character.

If a matching directory entry is found, the filename in the directory entry is changed to match the second filename in the modified FCB. If the ? wildcard character is used in the second filename, the corresponding characters in the filename of the directory entry are not changed.

Entry Conditions:

AH = 17H
DS:DX = *pointer to the FCB containing the current and new filenames*

Exit Conditions:

If AL = 00H, a directory entry was found.
If AL = FFH, no directory entry was found or no match exists.

Macro Definition:

```
rename      macro     fcb,newname
            mov       dx,offset fcb
            mov       ah,17H
            int       21H
            endm
```

Example:

The following program prompts for the name of a file and a new name, then renames the file.

```
fcb             db              37 dup (?)
prompt1         db              "Filename: $"
prompt2         db              "New name: $"
reply           db              17 dup(?)
crlf            db              13,10,:"$"
                .
                .
func_17H:       display prompt1                 ;see Function 09H
                get_string  15,reply            ;see Function 0AH
                display         crlf            ;see Function 09H
                parse           reply[2],fcb    ;see Function 29H
                display         prompt2         ;see Function 09H
                get_string 15,reply             ;see Function 0AH
                display         crlf            ;see Function 09H
                parse           reply[2],fcb[16] ;see Function 29H
                rename          fcb             ;THIS FUNCTION
```

Curdsk

Current Disk — Function Call 19H

Returns the code of the currently selected drive.

Entry Conditions:

AH = 19H

Exit Conditions:

AL = *currently selected drive (0 = A, 1 = B, etc.)*

Macro Definition:

```
current_disk        macro
                    mov             ah,19H
                    int             21H
                    endm
```

Example:

The following program displays the currently selected (default) drive in a 2-drive system.

```
message         db              "Current disk is $" ;see Function 09H
                                                    ;for explanation of $
crlf            db              13,10,"$"
                .
                .
func_19H:       display         message             ;see Function 09H
                current_disk                        ;THIS FUNCTION
                cmp             al,00H              ;is it disk A?
                jne             disk_b              ;no, it's disk B
                display_char "A"                    ;see Function 02H
                jmp             all_done
disk_b:         display_char "B"                    ;see Function 02H
all_done:       display         crlf                ;see Function 09H
```

Set Disk Transfer Address Function Call 1AH

Sets the Disk Transfer Address to the specified address. Disk transfers cannot wrap around from the end of the segment to the beginning, nor can they overflow into the next segment.

If you do not set the Disk Transfer Address, it defaults to offset 80H in the Program Segment Prefix.

Entry Conditions:

AH = 1AH
DS:DX = *address to set as Disk Transfer Address*

Macro Definition:

```
set_dta         macro       buffer
                mov         dx,offset buffer
                mov         ah,1AH
                int         21H
                endm
```

Example:

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), and then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in Drive B. The file contains 26 records; each record is 28 bytes long.

```
record_size                 equ  14          ;offset of Record Size
                                             ;field of FCB
relative_record             equ  33          ;offset of Relative Record
                                             ;field of FCB
                .
                .
fcb             db          2,"ALPHABETDAT"
                db          25 dup (?)
buffer          db          34 dup(?),"$"
prompt          db          "Enter letter: $"
crlf            db          13,10,"$"
                .
                .
func_1AH:       set_dta     buffer           ;THIS FUNCTION
                open        fcb              ;see Function 0FH
                mov         fcb[record_size],28 ;set record size
get_char:       display     prompt           ;see Function 09H
                read_kbd_and_echo            ;see Function 01H
                cmp         al,0DH           ;just a CR?
                je          all_done         ;yes, go home
                sub         al,41H           ;convert ASCII
                                             ;code to record #
                mov         fcb[relative_record],al ;set relative record
                display     crlf             ;see Function 09H
                read_ran    fcb              ;see Function 21H
                display     buffer           ;see Function 09H
                display     crlf             ;see Function 09H
                jmp         get_char         ;get another character
all_done:       close       fcb              ;see Function 10H
```

RandomRead

Random Read — Function Call 21H

Performs a random read of a record. The current block (offset 0CH) and current record (offset 20H) fields in the FCB are set to agree with the relative record field (offset 21H). The record addressed by these fields is then loaded at the Disk Transfer Address.

Entry Conditions:

AH = 21H
DS:DX = *pointer to the opened FCB of the file to read*

Exit Conditions:

If AL = 00H, the read was completed successfully.
If AL = 01H, end of file was encountered; no data is in the record.
If AL = 02H, there was not enough room at the Disk Transfer Address to read 1 record; the read was canceled.
If AL = 03H, end of file was encountered; a partial record was read and padded to the record length with zeroes.

Macro Definition:

```
read_ran    macro    fcb
            mov      dx,offset fcb
            mov      ah,21H
            int      21H
            endm
```

Example:

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), and then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in Drive B. The file contains 26 records; each record is 28 bytes long.

```
record_size                 equ  14          ;offset of Record Size
                                             ;field of FCB
relative_record             equ  33          ;offset of Relative Record
                                             ;field of FCB
                .
                .
fcb             db          2,"ALPHABETDAT"
                db          25 dup(?)
buffer          db          34 dup(?),"$"
prompt          db          "Enter Letter: $"
crlf            db          13,10,"$"
                .
                .
func_21H:       set_dta     buffer           ;see Function 1AH
                open        fcb              ;see Function 0FH
                mov         fcb[record_size],28 ;set record size
get_char:       display     prompt           ;see Function 09H
                read_kbd_and_echo            ;see Function 01H
                cmp         al,0DH           ;just a CR?
                je          all_done         ;yes, go home
                sub         al,41H           ;convert ASCII code
                                             ;to record #
                mov         fcb[relative_record],al ;set relative
                                             ;record
                display     crlf             ;see Function 09H
                read_ran    fcb              ;THIS FUNCTION
                display     buffer           ;see Function 09H
                display     crlf             ;see Function 09H
                jmp         get_char         ;get another char
all_done:       close       fcb              ;see Function 10H
```

RandomWrite

Random Write — Function Call 22H

Performs a random write of a record. The current block (offset 0CH) and current record (offset 20H) fields in the FCB are set to agree with the relative record field (offset 21H). The record addressed by these fields is then written from the Disk Transfer Address. If the record size is smaller than a sector (512 bytes), the records are buffered until a full sector is ready to write.

Entry Conditions:

AH = 22H
DS:DX = *pointer to the opened FCB of the file to write*

Exit Conditions:

If AL = 00H, the write was completed successfully.
If AL = 01H, the disk is full.
If AL = 02H, there was not enough room at the Disk Transfer Address to write 1 record; the write was canceled.

Macro Definition:

```
write_ran       macro       fcb
                mov         dx,offset fcb
                mov         ah,22H
                int         21H
                endm
```

Example:

The following program prompts for a letter, converts the letter to its alphabetic sequence (A = 1, B = 2, etc.), and then reads and displays the corresponding record from a file named ALPHABET.DAT on the disk in Drive B. After displaying the record, it prompts the user to enter a changed record. If the user types a new record, it is written to the file; if the user presses [ENTER], the record is not replaced. The file contains 26 records; each record is 28 bytes long.

```
record_size                 equ  14          ;offset of Record Size
                                             ;field of FCB
relative_record             equ  33          ;offset of Relative Record
                                             ;field of FCB
                .
                .
fcb             db          2,"ALPHABETDAT"
                db          25 dup (?)
buffer          db          26 dup(?),13,10,"$"
prompt1         db          "Enter letter: $"
prompt2         db          "New record (RETURN for no change): $"
crlf            db          13,10,"$"
reply           db          28 dup (32)
blanks          db          26 dup (32)
                .
                .
func_22H:       set_dta     buffer           ;see Function 1AH
                open        fcb              ;see Function 0FH
                mov         fcb[record_size],32 ;set record size
get_char:       display     prompt1          ;see Function 09H
                read_kbd_and_echo            ;see Function 01H
                cmp         al,0DH           ;just a CR?
                je          all_done         ;yes, go home
                sub         al,41H           ;convert ASCII
                                             ;code to record #
                mov         fcb[relative_record],al
                                             ;set relative record
                display     crlf             ;see Function 09H
                read_ran    fcb              ;THIS FUNCTION
                display     buffer           ;see Function 09H
                display     crlf             ;see Function 09H
                display     prompt2          ;see Function 09H
                get_string  27,reply         ;see Function 0AH
                display     crlf             ;see Function 09H
                cmp         reply[1],0       ;was anything typed
                                             ;besides CR?
                je          get_char         ;no
                                             ;get another char.
                xor         bx,bx            ;to load a byte
                mov         bl,reply[1]      ;use reply length as
                                             ;counter
                move_string blanks,buffer,26 ;see chapter end
                move_string reply[2],buffer,bx ;see chapter end
                write_ran   fcb              ;see Function 21H
                jmp         get_char         ;get another character
all_done:       close       fcb              ;see Function 10H
```

FileSize

File Size — Function Call 23H

Searches the disk directory for the first matching entry for a specified FCB. If a matching directory entry is found, the relative record field (offset 21H) is set to the number of records in the file, calculated from the total file size in the directory entry (offset 1CH) and the record size field (offset 0EH) of the FCB.

If the value of the record size field of the FCB does not match the actual number of characters in a record, this function does not return the correct file size. If the default record size (128) is not correct, you must set the record size field to the correct value before using this function.

Entry Conditions:

AH = 23H
DS:DX = *pointer to the file's unopened FCB*

Exit Conditions:

If AL = 00H, a directory entry was found.
If AL = FFH, no directory entry was found.

Macro Definition:

```
file_size       macro       fcb
                mov         dx,offset fcb
                mov         ah,23H
                int         21H
                endm
```

Example:

The following program prompts for the name of a file, opens the file to fill in the Record Size field of the FCB, issues a File Size function call, and displays the file size and number of records in hexadecimal format.

```
fcb             db              37 dup (?)
prompt          db              "File name: $"
msg1            db              "Record length:    ",13,10,"$"
msg2            db              "Records:    ",13,10,"$"
crlf            db              13,10,"$"
reply           db              17 dup(?)
sixteen         db              16
                .
                .
func_23H:       display prompt                      ;see Function 09H
                get_string  17,reply                ;see Function 0AH
                cmp             reply[1],0          ;just a CR?
                jne             get_length          ;no,keep going
                jmp             all_done            ;yes, go home
get_length:     display         crlf                ;see Function 09H
                parse           reply[2],fcb        ;see Function 29H
                open            fcb                 ;see Function 0FH
                file_size       fcb                 ;THIS FUNCTION
                mov             si,33               ;offset to Relative
                                                    ;Record field
                mov             di,9                ;reply in msg_2
convert_it:     cmp             fcb[si],0           ;digit to convert?
                je              show_it             ;no, prepare message
                convert         fcb[si],sixteen,msg_2[di]
                inc             si                  ;bump n-o-r index
                inc             di                  ;bump message index
                jmp             convert_it          ;check for a digit
show_it:        convert         fcb[14],sixteen,msg_1[15]
                display         msg_1               ;see Function 09H
                display         msg_2               ;see Function 09H
                jmp             func_23H            ;get a filename
all_done:       close           fcb                 ;see Function 10H
```

SetRelRec

Set Relative Record — Function Call 24H

Sets the relative record field (offset 21H) in a specified FCB to the same file address that is indicated by the current block (offset 0CH) and current record (offset 20H) fields.

Entry Conditions:

AH = 24H
DS:DX = *pointer to an opened FCB*

Macro Definition:

```
set_relative_record     macro       fcb
                        mov         dx,offset fcb
                        mov         ah,24H
                        int         21H
                        endm
```

Example:

The following program copies a file using the Random Block Read and Random Block Write function calls. It speeds the copy by setting the record length equal to the file size and the record count to 1, and using a buffer of 32K bytes. It positions the file pointer by setting the current record field (offset 20H) to 1 and using the Set Relative Record function call to make the relative record field (offset 21H) point to the same record as the combination of the current block (offset 0CH) and current record (offset 20H) fields.

```
current_record  equ             32              ;offset of Current Record
                                                ;field of FCB
file_size       equ             16              ;offset of File Size
                                                ;field of FCB
                .
                .
fcb             db              37 dup (?)
filename        db              17 dup(?)
prompt1         db              "File to copy: $"   ;see Function 09H for
prompt2         db              "Name of copy: $"   ;explanation of $
crlf            db              13,10,"$"
file_length     dw              ?
buffer          db              32767 dup(?)
                .
                .
func_24H:       set_dta         buffer          ;see Function 1AH
                display         prompt1         ;see Function 09H
                get_string      15,filename     ;see Function 0AH
                display         crlf            ;see Function 09H
                parse           filename[2],fcb ;see Function 29H
                open            fcb             ;see Function 0FH
                mov             fcb[current_record],0 ;set Current Record
                                ;field
                set_relative_record  fcb        ;THIS FUNCTION
                mov             ax,word ptr fcb[file_size] ;get file size
                mov             file_length,ax  ;save it for
                ran_blook_read  fcb,1,ax        ;ran_block_write
                                                ;see Function 27H
                display         prompt2         ;see Function 09H
                get_string      15,filename     ;see Function 0AH
                display         crlf            ;see Function 09H
                parse           filename[2],fcb ;see Function 29H
                create          fcb             ;see Function 16H
                mov             fcb[current_record],0 ;set Current Record
                                                ;field
                set_relative_record  fcb        ;THIS FUNCTION
                mov             ax,file_length  ;get original file
                                                ;length
                ran_block_write  fcb,1,ax       ;see Function 28H
                close           fcb             ;see Function 10H
```

Setvector

Set Interrupt Vector — Function Call 25H

Sets a particular interrupt vector. The operating system can then manage the interrupts on a per-process basis. This call sets the address in the vector table for the specified interrupt to the address of the interrupt handling routine in AL.

Note that programs should never set interrupt vectors by writing them directly in the low memory vector table.

Entry Conditions:

AH = 25H
AL = *number of the interrupt to set*
DS:DX = *address of the interrupt handling routine*

Macro Definition:

```
set_vector      macro       interrupt,seg_addr,off_addr
                push        ds
                mov         ax,seg_addr
                mov         ds,ax
                mov         dx,off_addr
                mov         al,interrupt
                mov         ah,25H
                int         21H
                pop         ds
                endm
```

Example:

```
lds             dx,intvector
mov             ah,25H
mov             al,intnumber
int             21H
;There are no errors returned
```

RBRead

Random Block Read — Function Call 27H

Reads the specified number of records (calculated from the record size field at offset 0EH of the FCB), starting at the record specified by the relative record field (offset 21H). The records are placed at the Disk Transfer Address. The current block (offset 0CH), current record (offset 20H), and relative record (offset 21H) fields are set to address the next record.

If the number of records to read is specified as zero, the call returns without reading any records (no operation).

Entry Conditions:

AH = 27H
CX = *number of records to read*
DS:DX = *pointer to the opened FCB of the file to read*

Exit Conditions:

CX = actual number of records read
If AL = 00H, all records were read successfully.
If AL = 01H, end of file was encountered before all records were read; the last record is complete.
If AL = 02H, wrap-around above address FFFFH in the disk transfer segment would occur if all records were read; therefore, only as many records were read as was possible without wrap-around.
If AL = 03H, end of file was encountered before all records were read; the last record is partial.

Macro Definition:

```
ran_block_read      macro       fcb,count,rec_size
                    mov         dx,offset fcb
                    mov         cx,count
                    mov         word ptr fcb[14],rec_size
                    mov         ah,27H
                    int         21H
                    endm
```

Example:

The following program copies a file using the Random Block Read function call. It speeds the copy by specifying a record count of 1 and a record length equal to the file size, and using a buffer of 32K bytes; the file is read as a single record. (Compare this example with the sample program for Function 28H, that specifies a record length of 1 and a record count equal to the file size.)

```
current_record          equ   32          ;offset of Current Record
                                          ;field
file_size               equ   16          ;offset of File Size field
                .
                .
fcb             db      37 dup (?)
filename        db      17 dup(?)
prompt1         db      "File to copy: $"  ;see Function 09H
                                           ;for explanation
prompt2         db      "Name of copy: $"  ;of $
crlf            db      13,10,"$"
file_length     dw      ?
buffer          db      32767 dup(?)
                .
                .
func_27H:       set_dta     buffer          ;see Function 1AH
                display     prompt1         ;see Function 09H
                get_string  15,filename     ;see Function 0AH
                display     crlf            ;see Function 09H
                parse       filename[2],fcb ;see Function 29H
                open        fcb             ;see Function 0FH
                mov         fcb[current_record],0  ;set Current
                                           ;Record field
                set_relative_record  fcb    ;see Function 24H
                mov         ax, word ptr    ;fcb[file_size]
                                           ;get file size
                mov         file_length,ax  ;save it for
                ran_block_read fcb,1,ax     ;ran_block_write
                                           ;THIS FUNCTION
                display     prompt2         ;see Function 09H
                get_string  15,filename     ;see Function 0AH
                display     crlf            ;see Function 09H
                parse       filename[2],fcb ;see Function 29H
                create      fcb             ;see Function 16H
                mov         fcb[current_record],0
                                           ;set Current Record
                                           ;field
                set_relative_record  fcb    ;see Function 24H
                mov         ax,file_length  ;get original file
                                           ;size
                ran_block_write  fcb,1,ax   ;see Function 28H
                close       fcb             ;see Function 10H
```

RBWrite

Random Block Write — Function Call 28H

Writes the specified number of records (calculated from the record size field at offset 0EH of the FCB) from the Disk Transfer Address. The records are written to the file starting at the record specified in the relative record field (offset 21H). The current block (offset 0CH), current record (offset 20H), and relative record (offset 21H) are then set to address the next record.

If the number of records is specified as zero, no records are written, but the file size field of the directory entry (offset 1CH) is set to the number of records specified by the relative record field of the FCB (offset 21H). Allocation units are allocated or released, as required.

Entry Conditions:

AH = 28H
DS: DX = *pointer to the opened FCB of the file to write*
CX = *number of records to write (non zero)*
or
CX = 0 (sets the file size field; see above)

Exit Conditions:

CX = *actual number of records written*
If AL = 00H, all records were written successfully.
If AL = 01H, no records were written because there is insufficient space on the disk.

Macro Definition:

```
ran_block_write     macro     fcb,count,rec_size
                    mov       dx,offset fcb
                    mov       cx,count
                    mov       word ptr fcb[14],rec_size
                    mov       ah,28H
                    int       21H
                    endm
```

Example:

The following program copies a file using the Random Block Read and Random Block Write function calls. It speeds the copy by specifying a record count equal to the file size and a record length of 1. With a buffer of 32K bytes the file is copied quickly, requiring 1 disk access each to read and write. (Compare this example with the sample program for Function 27H, that specifies a record count of 1 and a record length equal to file size.)

```
current_record              equ    32          ;offset - Current Rec field
file_size                   equ    16          ;offset - File Size field
                .
                .
fcb             db          37 dup (?)
filename        db          17 dup(?)
prompt1         db          "File to copy: $"  ;see Function 09H for
prompt2         db          "Name of copy: $"  ;explanation of $
crlf            db          13,10,"$"
num_recs        dw          ?
buffer          db          32767 dup(?)
                .
                .
func_28H:       set_dta     buffer             ;see Function 1AH
                display     prompt1            ;see Function 09H
                get_string  15,filename        ;see Function 0AH
                display     crlf               ;see Function 09H
                parse       filename[2],fcb    ;see Function 29H
                open        fcb                ;see Function 0FH
                mov         fcb[current_record],0
                                               ;set Current Record
                                               ;field
                set_relative_record  fcb       ;see Function 24H
                mov         ax, word ptr fcb [file_size]
                                               ;get file size
                mov         num_recs,ax        ;save it for
                                               ;ran_block_write
                ran_block_read  fcb,num_recs,1 ;THIS FUNCTION
                display     prompt2            ;see Function 09H
                get_string  15,filename        ;see Function 0AH
                display     crlf               ;see Function 09H
                parse       filename[2],fcb    ;see Function 29H
                create      fcb                ;see Function 16H
                mov         fcb[current_       ;set Current
                            record],0          ;Record field
                set_relative_record  fcb       ;see Function 24H
                mov         ax, file_length    ;get size of
                                               ;original
                ran_block_write  fcb,num_recs,1 ;see Function 28H
                close       fcb                ;see Function 10H
```

Fname

Parse Filename — Function Call 29H

Parses a string for a filename of the form *d:filename.ext*. If one is found, a corresponding unopened FCB is created at a specified location.

Bits 0-3 of AL control the parsing and processing (bits 4-7 are ignored):

Bit	Value	Meaning
0	0	All parsing stops if a file separator is encountered.
	1	Leading separators are ignored.
1	0	The drive number in the FCB is set to 0 (default drive) if the string does not contain a drive number.
	1	The drive number in the FCB is not changed if the string does not contain a drive number.
2	0	The filename in the FCB is set to 8 blanks if the string does not contain a filename.
	1	The filename in the FCB is not changed if the string does not contain a filename.
3	0	The extension in the FCB is set to 3 blanks if the string does not contain an extension.
	1	The extension in the FCB is not changed if the string does not contain an extension.

If the filename or extension includes an asterisk (*), all remaining characters in the name or extension are set to question mark (?).

The filename separators are:

: . ; , = + [ ] / < > | space tab

Filename terminators include all the filename separators plus all control characters. A filename cannot contain a filename terminator; if one is encountered, parsing stops.

Entry Conditions:

AH = 29H
DS:SI = *pointer to string to parse*
ES:DI = *pointer to a portion of memory to fill in with an unopened FCB*
AL = *controls parsing (see above)*

Exit Conditions:

If AL = 00H, then no wildcard characters appeared in the filename or extension.
If AL = 01H, then wildcard characters appeared in the filename or extension.
DS:SI = *pointer to the first byte after the string that was parsed*
ES:DI = *unopened FCB*

Macro Definition:

```
parse       macro     string,fcb
            mov       si,offset string
            mov       di,offset fcb
            push      es
            push      ds
            pop       es
            mov       al,0FH ;bits 0, 1, 2, 3 on
            mov       ah,29H
            int       21H
            pop       es
            endm
```

Example:

The following program verifies the existence of the file named in reply to the prompt.

```
fcb             db              37 dup (?)
prompt          db              "Filename: $"
reply           db              17 dup(?)
yes             db              "FILE EXISTS",13,10,"$"
no              db              "FILE DOES NOT EXIST",13,10,"$"
                .
                .
func_29H:       display         prompt              ;see Function 09H
                get_string      15,reply            ;see Function 0AH
                parse           reply[2],fcb        ;THIS FUNCTION
                search_first    fcb                 ;see Function 11H
                cmp             al,0FFH             ;dir. entry found?
                je              not_there           ;no
                display         yes                 ;see Function 09H
                jmp             continue
not_there:      display         no
continue:       .
                .
```

GetDate

Get Date — Function Call 2AH

Returns the current date set in the operating system. The date is returned as binary numbers.

Entry Conditions:

AH = 2AH

Exit Conditions:

CX = *year (1980-2099)*
DH = *month (1 = January, 2 = February, etc.)*
DL = *day of the month (1-31)*
AL = *day of the week (0 = Sunday, 1 = Monday, etc.)*

Macro Definition:

```
get_date        macro
                mov         ah,2AH
                int         21H
                endm
```

Example:

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date.

```
month       db          31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func_2AH:   get_date                    ;see above
            inc         dl              ;increment day
            xor         bx,bx           ;so BL can be used as
                                        ;index
            mov         bl,dh           ;move month to index
                                        ;register
            dec         bx              ;month table starts with 0
            cmp         dl,month[bx]    ;past end of month?
            jle         month_ok        ;no, set the new date
            mov         dl,1            ;yes, set day to 1
            inc         dh              ;and increment month
            cmp         dh,12           ;past end of year?
            jle         month_ok        ;no, set the new date
            mov         dh,1            ;yes, set the month to 1
            inc         cx              ;increment year
month_ok:   set_date    cx,dh,dl        ;see Function 2BH
```

SetDate

Set Date — Function Call 2BH

Sets the date to a valid date in binary given in CX and DX.

Entry Conditions:

AH = 2BH
CX = *year (1980-2099)*
DH = *month (1 = January, 2 = February, etc.)*
DL = *day of the month (1-31)*

Exit Conditions:

If AL = 00H, the date was valid.
If AL = FFH, the date was not valid and the function was canceled.

Macro Definition:

```
set_date    macro     year,month,day
            mov       cx,year
            mov       dh,month
            mov       dl,day
            mov       ah,2BH
            int       21H
            endm
```

Example:

The following program gets the date, increments the day, increments the month or year, if necessary, and sets the new date.

```
month       db          31,28,31,30,31,30,31,31,30,31,30,31
            .
            .
func_2BH:   get_date                    ;see Function 2AH
            inc         dl              ;increment day
            xor         bx,bx           ;so BL can be used as
                                        ;index
            mov         bl,dh           ;move month to index
                                        ;register
            dec         bx              ;month table starts with 0
            cmp         dl,month[bx]    ;past end of month?
            jle         month_ok        ;no, set the new date
            mov         dl,1            ;yes, set day to 1
            inc         dh              ;and increment month
            cmp         dh,12           ;past end of year?
            jle         month_ok        ;no, set the new date
            mov         dh,1            ;yes, set the month
                                        ;to 1
            inc         cx              ;increment year
month_ok:   set_date    cx,dh,dl        ;THIS FUNCTION
```

GetTime

Get Time — Function Call 2CH

Returns the current time set in the operating system as binary numbers.

Entry Conditions:

AH = 2CH

Exit Conditions:

CH = *hour (0-23)*
CL = *minutes (0-59)*
DH = *seconds (0-59)*
DL = *hundredths of a second (0-99)*

Macro Definition:

```
get_time        macro
                mov         ah,2CH
                int         21H
                endm
```

Example:

The following program continuously displays the time until any key is pressed.

```
time            db          "00:00:00.00",13,10,"$"
ten             db          10
                .
                .
func_2CH:       get_time                        ;THIS FUNCTION
                convert ch,ten,time             ;see end of chapter
                convert cl,ten,time[3]          ;see end of chapter
                convert dh,ten,time[6]          ;see end of chapter
                convert dl,ten,time[9]          ;see end of chapter
                display time                    ;see Function 09H
                check_kbd_status                ;see Function 0BH
                cmp         al,0FFH             ;has a key been pressed?
                je          all_done            ;yes, terminate
                jmp         func_2CH            ;no, display time
all_done:       ret
```

SetTime

Set Time — Function Call 2DH

Sets the time to a valid time in binary given in CX and DX.

Entry Conditions:
AH = 2DH
CH = *hour (0-23)*
CL = *minutes (0-59)*
DH = *seconds (0-59)*
DL = *hundredths of a second (0-99)*

Exit Conditions:

If AL = 00H, the time specified on entry is valid.
If AL = FFH, the time was not valid; the function was canceled.

Macro Definition:

```
set_time    macro    hour,minutes,seconds,hundredths
            mov      ch,hour
            mov      cl,minutes
            mov      dh,seconds
            mov      dl,hundredths
            mov      ah,2DH
            int      21H
            endm
```

Example:

The following program sets the system clock to 0 and continuously displays the time. When a character is typed, the display freezes; when another character is typed, the clock is reset to 0 and the display starts again.

```
time            db              "00:00:00.00", 13,10,"$"
ten             db              10
                .
                .
func_2DH:       set_time 0,0,0,0                ;THIS FUNCTION
read_clock:     get_time                        ;see Function 2CH
                convert ch,ten,time             ;see end of chapter
                convert cl,ten,time[3]          ;see end of chapter
                convert dh,ten,time[6]          ;see end of chapter
                convert dl,ten,time[9]          ;see end of chapter
                display time                    ;see Function 09H
                dir_console_io 0FFH             ;see Function 06H
                cmp             al,00H          ;was a char. typed?
                jne             stop            ;yes, stop the timer
                jmp             read_clock      ;no keep timer on
stop:           read_kbd                        ;see Function 08H
                jmp             func_2DH        ;keep displaying time
```

SetVerify

Set/Reset Verify Flag — Function Call 2EH

Specifies whether each disk write is to be verified or not. MS-DOS checks this flag each time it writes to a disk. The verify flag is normally off.

Entry Conditions:

AH = 2EH
AL = *verify Flag*
00H = *do not verify*
01H = *verify*

Macro Definition:

```
verify      macro     switch
            mov       al,switch
            mov       ah,2EH
            int       21H
            endm
```

Example:

The following program copies the contents of a single sided disk in Drive A to the disk in Drive B, verifying each write. It uses a buffer of 32K bytes.

```
on              equ             1
off             equ             0
                .
                .
prompt          db              "Source in A,target in B",13,10
                db              "Any key to start. $"
start           dw              0
buffer          db              64 dup (512 dup(?))  ;64 sectors
                .
                .
func_2DH:       display prompt                       ;see Function 09H
                read_kbd                             ;see Function 08H
                verify on                            ;THIS FUNCTION
                mov cx,5                             ;copy 64 sectors
                                                     ;5 times
copy:           push            cx                   ;save counter
                abs_disk_read   0,buffer,64,start    ;see Interrupt 25H
                abs_disk_write  1,buffer,64,start    ;see Interrupt 26H
                add             start,64             ;do next 64 sectors
                pop             cx                   ;restore counter
                loop            copy                 ;do it again
                verify          off                  ;THIS FUNCTION
                disk_read       0,buffer,64,start    ;see Interrupt 25H
                abs_disk_write                       1,buffer,64,start
                                                     ;see Interrupt 26H
                add             start,64             ;do next 64 sectors
                pop             cx                   ;restore counter
                loop            copy                 ;do it again
                verify          off
```

GetDTA

Get Disk Transfer Address Function Call 2FH

Returns the Disk Transfer Address.

Entry Conditions:

AH = 2FH

Exit Conditions:

ES:BX = *pointer to current Disk Transfer Address*

Error Returns: None.

Example:

```
Get DTA         equ         2FH
                mov         ah,GetDTA
                int         21H
```

GetVersion

Get Version Number — Section Call 30H

Returns the MS-DOS version number. AL:AH contains the 2 part version designation on return. For example, for MS-DOS 2.0, AL would contain 2 and AH would contain 0.

Entry Conditions:

AH = 30H

Exit Conditions:

AL = *major version number*
AH = *minor version number*
BH = *OEM (original equipment manufacturer) number*
BL:CX = *24-bit user number.*

Error Returns: None.

Example:

```
GetVersion      equ         30H
                mov         ah,GetVersion
                int         21H
```

KeepProcess

Keep Process — Function Call 31H

Terminates the current process and attempts to set the initial allocation block to the specified size in paragraphs. No other allocation blocks belonging to that process are freed. The exit code passed in AX is retrievable by the parent via function call 4DH.

This method is preferred over Interrupt 27H and has the advantage of allowing more than 64K to be kept.

Entry Conditions:

AH = 31H
AL = *exit code*
DX = *memory size in paragraphs*

Error Returns: None.

Example:

```
KeepProcess     equ     31H
                mov     al,exitcode
                mov     dx,parasize
                mov     ah,KeepProcess
                int     21H
```

SetCtrlCTrapping

CONTROL-C Check — Function Call 33H

MS-DOS only checks for a CONTROL-C on the controlling device when performing function call operations 01H-0CH to that device. Function Call 33H lets you expand this checking to include any system call. For example, with the CONTROL-C trapping off, all disk I/O proceeds without interruption. With CONTROL-C trapping on, the CONTROL-C interrupt is given at the system call that initiates the disk operation.

Note that programs using Function Calls 06H or 07H to read CONTROL-Cs as data must ensure that the CONTROL-C check is off.

Entry Conditions:

AH = 33H
AL = *function*
00H = Return current state
01H = Set state
DL = *switch (if setting state)*
00H = Off
01H = On

Exit Conditions:

DL = *current state*
00H = Off
01H = On

Error Return:

AL = FFH
The function passed in AL was not in the range 00H-01H.

Example:

```
SetCtrlCTrapping        equ         33H
                        mov         dl,val
                        mov         al,func
                        mov         al,SetCtrlCTrapping
```

GetVector

Get Interrupt Vector — Function Call 35H

Returns the interrupt vector associated with a specified interrupt. Note that programs should never get an interrupt vector by reading the low memory vector table directly.

Entry Conditions:

AH = 35H
AL = *interrupt number*

Exit Conditions:

ES:BX = *pointer to interrupt routine*

Error Returns: None.

Example:

```
GetVector       equ     35H
                mov     al,interrupt
                mov     ah,GetVector
                int     21H
```

GetFreeSpace

Get Disk Free Space — Function Call 36H

Returns the amount of free space on the disk along with additional information about the disk.

Entry Conditions:

AH = 36H
DL = *drive (0 = default, 1 = A, etc.)*

Exit Conditions:

BX = *number of free allocation units on drive*
DX = *total number of allocation units on drive*
CX = *bytes per sector*
AX = *sectors per allocation unit or*
AX = FFFFH (if drive number is invalid)

Error Returns:

AX = FFFFH
The drive number given in DL was invalid.

Example:

```
GetFreespace    equ     36H
                mov     dl,drive
                mov     ah,GetFreespace
                int     21H
```

International

Return Country-Dependent Information — Function Call 38H

Returns information pertinent to international applications in a buffer pointed to by DS:DX. The information is specific to the country indicated in AL. The value passed in AL is either 0 (for current country) or a country code. Country codes are typically the international telephone prefix code for the country.

If DX = −1, this call sets the current country to the country code in AL. If the country code is not found, the current country is not changed.

This function is fully supported only in MS-DOS versions 2.01 and higher. It exists in MS-DOS 2.0, but is not fully implemented.

This function returns the following information in the block of memory pointed to by DS:DX:

WORD Date/time format
5-BYTE ASCIIZ string Currency symbol
2-BYTE ASCIIZ string Thousands separator
2-BYTE ASCIIZ string Decimal separator
2-BYTE ASCIIZ string Date separator
2-BYTE ASCIIZ string Time separator
1-BYTE Bit field
1-BYTE Currency places
1-BYTE Time format
DWORD Case Mapping call
2-BYTE ASCIIZ string Data list separator

The format for most entries is ASCIIZ (a NUL-terminated ASCII string), but a fixed size is allocated for each field for easy indexing into the table.

The date/time format has the following values:

```
0 - USA standard       h:m:s m/d/y
1 - Europe standard    h:m:s d/m/y
2 - Japan standard     y/m/d h:m:s
```

The bit field contains 8 bit values. Any bit not currently defined can be assumed to have a random value.

Bit 0 = 0 If currency symbol precedes the currency amount.
= 1 If currency symbol comes after the currency amount.
Bit 1 = 0 If the currency symbol immediately precedes the currency amount.
= 1 If there is a space between the currency symbol and the amount.

The currency places field indicates the number of places which appear after the decimal point on currency amounts.

The time format has the following values:

0 = 12-hour time
1 = 24-hour time

The Case Mapping call is a FAR procedure which performs country-specific lower- to uppercase mapping on character values 80H-FFH. It is called with the character to be mapped in AL. It returns the correct uppercase code for that character, if any, in AL. AL and the FLAGS are the only registers altered. It is allowable to pass codes below 80H to this routine but nothing is done to characters in this range. In the case where there is no mapping, AL is not altered.

Entry Conditions:

AH = 38H
DS:DX = *pointer to 32-byte memory area*
AL = *country code* (In MS-DOS 2.0, this must be 0.)

Exit Conditions:

Carry set:
AX = *error code*

Carry not set:
DS:DX = *country data*

Error returns:

AX = 2
File not found. The country passed in AL was not found (no table exists for the specified country).

Example:

```
lds         dx, blk
mov         ah, 38H
mov         al, country_code
int         21H
   ;AX = country code of country returned
```

MkDir

Create Subdirectory — Function Call 39H

Creates a new directory entry at the end of a specified pathname.

Entry Conditions:

AH = 39H
DS:DX = *pointer to ASCIIZ pathname*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 3
    Path not found. The path specified was invalid or not found.
AX = 5
    Access denied. The directory could not be created (no room in the parent directory), the directory/file already existed, or a device name was specified.

Example:

```
MkDir       equ         39H
            lds         dx,pathname
            mov         ah,MkDir
            int         21H
```

RmDir

Remove a Directory Entry — Function Call 3AH

Removes a specified directory from its parent directory.

Entry Conditions:

AH = 3AH
DS:DX = *pointer to ASCIIZ pathname*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 3
    Path not found. The path specified was invalid or not found.
AX = 5
    Access denied. The path specified was not empty, was not a directory, was the root directory, or contained invalid information.
AX = 16
    Current directory. The path specified was the current directory on a drive.

Example:

```
RmDir       equ     3AH
            lds     dx,pathname
            mov     ah,RmDir
            int     21H
```

ChDir

Change the Current Directory — Function Call 3BH

Sets the current directory to the directory specified. If any member of the specified pathname does not exist, then the current directory is unchanged.

Entry Conditions:

AH = 3BH
DS:DX = *pointer to ASCIIZ pathname*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

A = 3
    Path not found. The path specified either indicated a file or the path was invalid.

Example:

```
ChDir       equ     3BH
            lds     dx,pathname
            mov     ah,ChDir
            int     21H
```

Create

Create a File — Function Call 3CH

Creates a new file or truncates an old file to zero length in preparation for writing. If the file did not exist, then the file is created in the appropriate directory and the file is given the attribute(s) found in CX. (See the section "Disk Directory" in Chapter 4 for a discussion of file attributes.) The file handle returned has been opened for read/write access.

Entry Conditions:

AH = 3CH
DS:DX = *pointer to ASCIIZ pathname*
CX = *file attribute(s)*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    AX = *file handle number*

Error Returns:

AX = 5
    Access denied: (1) the attributes specified in CX included one that could not be created (directory, volume ID) (2) file already existed with a more inclusive set of attributes, or (3) a directory existed with the same name.
AX = 3
    Path not found. The path specified was invalid.
AX = 4
    Too many open files. The file was created with the specified attributes but, either there were no free handles available for the process, or the internal system tables were full.

Example:

```
Creat           equ         3CH
                lds         dx,pathname
                mov         cx,attribute
                mov         ah,Creat
                int         21H
```

Open

Open a File — Function Call 3DH

Opens a file. The following values are allowed for the access code:

Ø — The file is opened for reading.
1 — The file is opened for writing.
2 — The file is opened for both reading and writing.

The read/write pointer is set at the first byte of the file and the record size of the file is 1 byte. The returned file handle must be used for subsequent I/O to the file.

Entry Conditions:

AH = 3DH
DS:DX = *pointer to ASCIIZ pathname for file to open*
AL = *access code (Ø = read, 1 = write, 2 = read and write)*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    AX = *file handle number*

Error Returns:

AX = 12
    Invalid access. The access specified in AL was not in the range Ø-2.
AX = 2
    File not found. The path specified was invalid or not found.
AX = 5
    Access denied. The user attempted to open a directory or volume ID, or open a read-only file for writing.
AX = 4
    Too many open files. There were no free handles available in the current process or the internal system tables were full.

Example:

```
Open            equ         3DH
                lds         dx,pathname
                mov         al,access
                mov         ah,Open
                int         21H
```

Close

Close a File Handle — Function Call 3EH

Closes the file associated with a specified file handle. Internal buffers are flushed.

Entry Conditions:

AH = 3EH
BX = *file handle for file to close*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Return:

AX = 6
    Invalid handle, The handle passed in BX was not currently open.

Example:

```
Close       equ     3EH
            mov     bx,handle
            mov     ah,Close
            int     21H
```

Read

Read from a File or Device — Function Call 3FH

Transfers a specified number of bytes from a file into a buffer location. It is not guaranteed that all bytes will be read. At most 1 line of text will be read from the keyboard. If the returned value for number of bytes read is zero, then the program tried to read from the end of file.

All I/O is done using normalized pointers; no segment wrap-around will occur. MS-DOS takes the pointer you specify in DS:DX and modifies it so that DX is 000FH or smaller.

Entry Conditions:

AH = 3FH
DS:DX = *pointer to buffer*
CX = *number of bytes to read*
BX = *file handle for the file to read*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    AX = *number of bytes read*

Error Returns:

AX = 6
    Invalid handle. The handle passed in BX is not currently open.
AX = 5
    Access denied. The handle passed in BX was opened in a mode that did not allow reading.

Example:

```
Read        equ         3FH
            lds         dx,buffer
            mov         cx,count
            mov         bx,handle
            mov         ah,Read
            int         21H
```

Write

Write to a File or Device — Function Call 40H

Transfers a specified number of bytes from a buffer into a file. If the number of bytes written is not the same as the number requested, then an error has occurred.

If the number of bytes to be written is zero, the file size is set to the current position. Allocation units are allocated or released as required.

All I/O is done using normalized pointers; no segment wrap-around will occur. MS-DOS takes the pointer you specify in DS:DX and modifies it so that DX is 000FH or smaller.

Entry Conditions:

AH = 40H
DS:DX = *pointer to buffer*
CX = *number of bytes to write*
BX = *file handle for file to write*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    AX = *number of bytes written*

Error Returns:

AX = 6
    Invalid handle. The handle passed in BX is not currently open.
AX = 5
    Access denied. The handle passed in BX was opened in a mode that did not allow writing.

Example:

```
Write       equ         40H
            lds         dx,buffer
            mov         cx,count
            mov         bx,handle
            mov         ah,Write
            int         21H
```

Unlink

Delete a Directry Entry — Function Call 41H

Removes a directory entry associated with a specified filename.

Entry Conditions:

AH = 41H
DS:DX = *pointer to pathname*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 2
    File not found. The path was invalid or not found.
AX = 5
    Access denied. The path was a directory or was read only.

Example:

```
Unlink      equ         41H
            lds         dx,pathname
            mov         ah,Unlink
            int         21H
```

LSeek

Move a File Pointer — Function Call 42H

Moves the read/write pointer a specified number of bytes according to the following methods:

- 0 - The pointer is moved to the specified offset from the beginning of the file.
- 1 - The pointer is moved to the current location plus the offset.
- 2 - The pointer is moved to the end of file plus the offset.

Entry Conditions:

AH = 42H
CX:DX = *distance to move the pointer, offset in bytes (CX contains the most significant part)*
AL = *method of moving (0, 1, or 2; see above)*
BX = *file handle*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    DX:AX = *new file pointer position*

Error Returns:

AX = 6
    Invalid handle. The handle passed in BX is not currently open.
AX = 1
    Invalid function. The function passed in AL was not in the range 0-2.

Example:

```
LSeek       equ         42H
            mov         dx,offsetlow
            mov         cx,offsethigh
            mov         al,method
            mov         bx,handle
            mov         ah,LSeek
            int         21H
```

ChMod

Change Attributes — Function Call 43H

Gets the attributes of a file, or sets the attributes of a file to those specified. (See the section "Disk Directory" in Chapter 4 for a description of file attributes.)

Entry Conditions:

AH = 43H
DS:DX = *pointer to ASCIIZ pathname of file*
AL = *function number*
    01H = set file's attributes to those in CX
    00H = return file's attributes in CX
CX = *attribute(s) to be set*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    CX = *current attribute(s) (if AL = 00H on entry)*

Error Returns:

AX = 3
    Path not found. The path specified was invalid.
AX = 5
    Access denied. The attributes specified in CX included one that could not be changed (directory, volume ID).
AX = 1
    Invalid function. The function passed in AL was not in the range 0-1.

Example:

```
ChMod       equ         43H
            lds         dx,pathname
            mov         dx,attribute
            mov         al,function
            mov         ah,43H
            int         21H
```

Ioctl

I/O Control for Devices — Function Call 44H

Performs the following functions: (1) gets or sets device information associated with an open handle, (2) sends a control string to a device handle or device, or (3) receives a control string from a device handle or device.

The following values are allowed for the function code passed in AL:

- 0 - Get device information (returned in DX).
- 1 - Set device information (as determined by DX).
- 2 - Read the number of bytes indicated in CX from the device control channel into buffer pointed to by DS:DX. (BX = file handle.)
- 3 - Write the number of bytes indicated in CX to the device control channel from the buffer pointed to by DS:DX. (BX = file handle.)
- 4 - Same as 2, but use the drive number in BL.
- 5 - Same as 3, but use the drive number in BL.
- 6 - Get input status.
- 7 - Get output status.

You can use this system call to get information about device channels. In addition, you can make calls on regular files using function values 0, 6, and 7. Function values other than these return an "Invalid function" error.

Calls AL = 0 and AL = 1:

The bits of DX are defined as follows. Note that the upper byte must be zero on a set call (A = 1).

15	14	13 12 11 10 9 8	7	6	5	4	3	2	1	0
Res	CTRL	Reserved D	ISFEV	EOW	RAE	SPCCL	ISNLK	ISCUL	ISCOT	ISIN

ISDEV = 1 if this channel is a device.
= 0 if this channel is a disk file (bits 8-15 = 0 in this case)

If ISDEV = 1:
EOF = 0 if end of file on input.
RAW = 1 if this device is in Raw mode (literal mode with no interpretation given to characters).
= 0 if this device is cooked (normal mode where the device interprets the characters).
SPECL = 1 if this device is special.
ISCLK = 1 if this device is the clock device.
ISNUL = 1 if this device is the null device.
ISCOT = 1 if this device is the console output.
ISCIN = 1 if this device is the console input.
CTRL = 0 if this device cannot process control strings via calls AL = 2 and AL = 3.
CTRL = 1 if this device can process control strings via calls AL = 2 and AL = 3. Note that the CTRL bit cannot be set by the Ioctl system call.

If ISDEV = 0:
EOF = 0 if channel has been written.
Bits 0-5 are the block device number for the channel (0 = A, 1 = B, etc.).

Bits 4, 8-13, and 15 are reserved and should not be altered.

Calls AL = 2, AL = 3, AL = 4, and AL = 5:

These 4 calls allow arbitrary control strings to be sent or received from a device. The call syntax is the same as for the read and write calls, except for calls AL = 4 and AL = 5 that pass a drive number in BL instead of a handle in BX.

An "Invalid function" error is returned if the CTRL bit is zero. An "Access denied" error is returned by calls AL = 4 and AL = 5 if the drive number is invalid.

Calls AL = 6 and AL = 7:

These calls allow you to check if a file handle is ready for input or output. Checking the status of handles open to a device is the intended use of these calls. Checking the status of a handle open to a disk file is also allowed, and is defined as follows:

Input: Always ready (AL = FFH) until end of file is reached, then always not ready (AL = 00H) unless the current position is changed via the Move a File Pointer function call (42H).

Output: Always ready (even if disk is full).

Note: The status is defined at the time the system is called. In future versions of MS-DOS, by the time control is returned to the user from the system, the status returned may not correctly reflect the true current state of the device or file.

Entry Conditions:

AH = 44H
BX = *handle*
BL = *drive (0=default, 1=A, etc.) (for calls AL=4, 5)*
DS:DX = *pointer to data or buffer*
CX = *number of bytes to read or write*
AL = *function code (0-7; see below)*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    For calls AL = 2, 3, 4, 5:
        AX = *number of bytes transferred*
    For calls AL = 6, 7:
        AX = 00H (not ready)
            or
        AX = FFH (ready)

Error Returns:

AX = 6
Invalid handle. The handle passed in BX is not currently open.

AX = 1
Invalid Function. The function passed in AL was not in the range Ø-7.

AX = 13
Invalid data.

AX = 5
Access denied (for calls AL = 4, 5, 6, 7).

Example:

```
ioctl           equ         44H
                mov         bx, handle
            (or mov         bl, drive       for calls AL = 4,5)
                mov         dx, data
            (or lds         dx, buffer      and
                mov         dx, count       for calls AL = 2,
                                            3, 4, 5)
                mov         al, function
                mov         ah, ioctl
                int         21H
```

Dup

Duplicate a File Handle — Function Call 45H

Takes an already opened file handle and returns a new handle that refers to the same file at the same position.

Entry Conditions:

AH = 45H
BX = *file handle to duplicate*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    AX = *new file handle*

Error Returns:

AX = 6
    Invalid Handle. The handle passed in BX is not currently open.
AX = 4
    Too many open files. There were no free handles available in the current process or the internal system tables were full.

Example:

```
Dup         equ         45H
            mov         bx, handle
            mov         ah, Dup
            int         21H
```

Dup2

Force a Duplicate of a File Handle — Function Call 46H

Takes a file handle that is already open and returns a new handle that refers to the same file at the same position. If there already is a file open on the handle specified in CX, it is closed first.

Entry Conditions:

AH = 46H
BX = *existing file handle*
CX = *new file handle*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 6
    Invalid handle. The handle passed in BX is not currently open.
AX = 4
    Too many open files. There were no free handles available in the current process or the internal system tables were full.

Example:

```
Dup2        equ         46H
            mov         bx, handle
            mov         cx, newhandle
            mov         ah, Dup2
            int         21H
```

CurrentDir

Return Text of Current Directory — Function Call 47H

Returns the current directory for a particular drive. The directory is root-relative and does not contain the drive specifier or leading path separator.

Entry Conditions:

AH = 47H
DS:SI = *pointer to 64-byte memory area to receive directory*
DL = *drive (0 = default, 1 = A, 2 = B, etc.)*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    DS:DI = *pointer to 64-byte area containing directory*

Error Returns:

AX = 15
    Invalid drive. The drive specified in DL was invalid.

Example:

```
CurrentDir      equ     47H
                lds     si, area
                mov     dl, drive
                mov     ah, CurrentDir
                int     21H
```

Alloc

Allocate Memory — Function Call 48H

Returns a pointer to a free block of memory that has the requested size in paragraphs.

Entry Conditions:

AH = 48H
BX = *size of memory to be allocated, in paragraphs*

Exit Conditions:

Carry set:
    BX = *maximum size that could be allocated, in paragraphs (if the requested size was not available)*
Carry not set:
    AX:0 = *pointer to the allocated memory*

Error Returns:

AX = 8
    Not enough memory. The largest available free block is smaller than that requested or there is no free block.
AX = 7
    Arena trashed. The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.

Example:

```
Alloc           equ         48H
                mov         bx, size
                mov         ah, Alloc
                int         21H
```

Dealloc

Free Allocated Memory — Function Call 49H

Returns to the system pool a piece of memory that was allocated by the Allocate Memory function call (48H).

Entry Conditions:

AH = 49H
ES = *segment address of memory area to be freed*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 9
    Invalid block. The block passed in ES is not one allocated via the Allocate Memory function call (48H).

AX = 7
    Arena trashed. The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.

Example:

```
Dealloc         equ         49H
                mov         es, block
                mov         ah, Dealloc
                int         21H
```

SetBlock

Modify Allocated Memory Blocks — Function Call 4AH

Attempts to "grow" or "shrink" an allocated block of memory.

Entry Conditions:

AH = 4AH
ES = *segment address of memory area*
BX = *requested memory area size, in paragraphs*

Exit Conditions:

Carry set:
BX = *maximum size possible, in paragraphs (if the requested size was not available on a grow request)*
Carry not set:
No error.

Error Returns:

AX = 9
Invalid block. The block specified in ES is not one allocated via this call.
AX = 7
Arena trashed. The internal consistency of the memory arena has been destroyed. This is due to a user program changing memory that does not belong to it.
AX = 8
Not enough memory. There was not enough free memory after the specified block to satisfy the grow request.

Example:

```
SetBlock        equ         4AH
                mov         es, block
                mov         bx, newsize
                mov         ah, SetBlock
                int         21H
```

Exec

Load and Execute a Program Function Call 4BH

Lets a program load another program into memory and optionally begin execution of the second program.

A function code is passed in AL:

- Ø - Load and execute the program. A Program Segment Prefix is established for the program and the terminate and CONTROL-C addresses are set to the instruction after the Exec function call.
- 3 - Load the program, do not create the program segment prefix, and do not begin execution. This is useful in loading program overlays.

For each value of AL, the parameter block pointed to by ES:BX has the following format:

AL = Ø Load and execute program

WORD segment address of environment string
DWORD pointer to command line to be placed at PSP + 80H
DWORD pointer to default FCB to be passed at PSP + 5CH
DWORD pointer to default FCB to be passed at PSP + 6CH

AL = 3 Load overlay

WORD segment address where file will be loaded
WORD relocation factor to be applied to the image

For function AL = Ø, there must be enough free memory for MS-DOS to load the program. For function AL = 3, it is assumed that the program doing the loading will load the overlay into its own memory space, so no free memory is needed.

Note that all open files of a process are duplicated in the child process after an Exec call. This is extremely powerful; the parent process has control over the meanings of stdin, stdout, stderr, stdaux, and stdprn. The parent could, for example, write a series of records to a file, open the file as standard input, open a listing file as standard output, and then Exec a sort program that takes its input from stdin and writes to stdout.

Also inherited (or passed from the parent) is an "environment." This is a block of text strings (less than 32K bytes total) that convey various configuration parameters. The format of the environment is as follows:

(paragraph boundary)

BYTE ASCIIZ string 1
BYTE ASCIIZ string 2
...
BYTE ASCIIZ string n
BYTE of zero

Typically the environment strings have the form:

parameter = value

For example, COMMAND.COM might pass its execution search path as:

PATH = A: BIN;B: BASIC LIB

A zero value of the environment address causes the child process to inherit a copy of the parent's environment unchanged.

Entry Conditions:

AH = 4BH
DS:DX = *pointer to ASCIIZ pathname*
ES:BX = *pointer to parameter block*
AL = *function code*
00H = load and execute program
03H = load program

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 1
    Invalid function. The function passed in AL was not 0, 1, or 3.
AX = 10
    Bad environment. The environment was larger than 32K bytes.
AX = 11
    Bad format. The file pointed to by DS:DX was an EXE format file and contained information that was internally inconsistent.
AX = 8
    Not enough memory. There was not enough memory for the process to be created.
AX = 2
    File not found. The path specified was invalid or not found.

Example:

```
Exec        qu          4BH
            lds         dx, pathname
            les         bx, block
            mov         al, function
            mov         ah, Exec
            int         21H
```

Exit

Terminate a Process — Function Call 4CH

Terminates the current process and transfers control to the invoking process. In addition, a return code may be sent. All files open at the time are closed.

This method is preferred over all others (Interrupt 20H, JMP 0).

Entry Conditions:

AH = 4CH
AL = *return code*

Error Returns: None.

Example:

```
Exec        equ         4CH
            mov         al, code
            mov         ah, Exit
            int         21H
```

Wait

Retrieve the Return Code of a Child — Function Call 4DH

Returns the Exit code specified by a child process. It returns this Exit code only once. The low byte of this code is sent by the Keep Process function call (31H). The high byte indicates the circumstances that caused the child to terminate, and is one of the following:

0 - Terminate/abort
1 - CONTROL-C
2 - Hard error
3 - Terminate and stay resident

Entry Conditions:

AH = 4DH

Exit Conditions:

AH = *exit code*

Error Returns: None.

Example:

```
Wait        equ         4DH
            mov         ah, Wait
            int         21H
```

FindFirst

Find Matching File — Function Call 4EH

Takes a pathname, with wildcard characters in the filename portion and a set of attributes, and attempts to find a file that matches the pathname and has a subset of the required attributes. If one is found, a data block at the current Disk Transfer Address is written. The block contains information in the following form:

21 bytes	-	Reserved for MS-DOS use on subsequent FindNext function calls (4FH)
1 byte	-	Attribute found
2 bytes	-	Time of file
2 bytes	-	Date of file
2 bytes	-	Least significant word of file size
2 bytes	-	Most significant word of file size
13 bytes	-	Packed filename and extension

To obtain the subsequent matches of the pathname, see the description of the FindNext function call (4FH).

Entry Conditions:

AH = 4EH
DS:DX = *pointer to ASCIIZ pathname*
CS = *search attributes*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 2
    File not found. The path specified in DS:DX was an invalid path.
AX = 18
    No more files. There were no files matching this specification.

Example:

```
FindFirst       equ     4EH
                lds     dx, pathname
                mov     cx, attribute
                mov     ah, FindFirst
                int     21H
```

FindNext

Find Next Matching File — Function Call 4FH

Finds the next matching entry in a directory. The current Disk Transfer Address must contain the block returned by the Find-First function call (4EH).

Entry Conditions:

AH = 4FH

Exit Conditions:

Carry set:
AX = *error code*
Carry not set:
No error.

Error Returns:

AX = 18
No more files. There are no more files matching this pattern.

Example:

```
FindNext        equ     4FH
                ;DTA contains block returned by
                ;FindFirst
                mov     ah, FindNext
                int     21H
```

GetVerifyFlag

Return Current Setting of Verify — Function Call 54H

Returns the current setting of the verify flag.

Entry Conditions:

AH = 54H

Exit Conditions:

AL = *current verify flag value*
00H = off
01H = on

Error Returns: None.

Example:

```
GetVerifyFlag       equ         54H
                    mov         ah, GetVerifyFlag
                    int         21H
```

Rename

Move a Directory Entry — Function Call 56H

Renames a file and/or moves it to another directory. This is done by giving the file a new filename, path, or both. The drive for both paths must be the same.

Entry Conditions:

AH = 56H
DS:DX = *pointer to ASCIIZ pathname of existing file*
ES:DI = *pointer to new ASCIIZ pathname*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    No error.

Error Returns:

AX = 2
    File not found. The filename specified by DS:DX was not found.
AX = 17
    Not same device. The source and destination are on different drives.
AX = 5
    Access denied: (1) the path specified in DS:DX was a directory, (2) the file specified by ES:DI already exists or,(3) the destination directory entry could not be created.

Example:

```
Rename      equ     56H
            lds     dx, source
            les     di, destination
            mov     ah, Rename
            int     21H
```

FileTimes

Get or Set a File's Date and Time — Function Call 57H

Returns or sets the date and time of last write for a file handle. The date and time are not recorded until the file is closed.

Entry Conditions:

AH = 57H
AL = *function code*
    00H = get date and time
    01H = set date and time
BX = *file handle*
CX = *time to be set (if AL = 01H)*
DX = *date to be set (if AL = 01H)*

Exit Conditions:

Carry set:
    AX = *error code*
Carry not set:
    If AL = 00 H on entry:
        CX = *time of last write*
        DX = *date of last write*

Error Returns:

AX = 1
    Invalid function. The function passed in AL was not 0 or 1.
AX = 6
    Invalid handle. The handle passed in BX was not currently open.

Example:

```
FileTimes       equ     57H
                mov     al, function
                mov     bx, handle if al = 1 then the next 2 are
                        ;mandatory
                mov     cx, time
                mov     dx, date
                mov     ah, FileTimes
                int     21H
```

Macro Definitions for MS-DOS System Call

Examples

Interrupts

```
terminate macro                                    ;PROGRAM_TERMINATE
      int         20H                              ;interrupt 20H
      endm                                         ;ABS_DISK_READ

abs_disk_read macro disk,buffer,num_sectors,first_sector
      mov         al,diskmov                       bx,offset buffer
      mov         cx,num_sectors
      mov         dx,first_sector
      int         25H                              ;interrupt 25H
      popf
      endm
                                                   ;ABS_DISK_WRITE
abs_disk_write macro disk,buffer,num_sectors,first_sector
      mov         al,disk
      mov         cx,num_sectors
      mov         dx,first_sector
      int         26H                              ;interrupt 26GH
      popf
      endm

stay_resident macro last_instruc                   ;STAY_RESIDENT
      mov         dx,offset last_instruc
      inc         dx
      int         27H                              ;interrupt 27H
      endm
```

Functions

```
terminate_program macro                            ;TERMINATE PROGRAM
      xor         ah,ah                            ;function 00H
      int         21H
      endm

read_kbd_and_echo  macro                           ;READ_KBD_AND_ECHO
      mov         ah,1                             ;function 01H
      int         21H
      endm

display_char macro character                       ;DISPLAY_CHAR
      mov         dl,character
      mov         ah,2                             ;function 02H
      int         21H
      endm
```

```
aux_input macro                          ;AUX_INPUT
      mov       ah,3                     ;function 03H
      int       21H
      endm

aux_output macro  character              ;AUX_OUTPUT
      mov       dl,character
      mov       ah,4                     ;function 04H
      int       21H
      endm

print_char macro  character              ;PRINT_CHAR
      mov       dl,character
      mov       ah,5                     ;function 05H
      int       21H
      endm

dir_console_io macro switch              ;DIR_CONSOLE_IO
      mov       dl,switch
      mov       ah,6                     ;function 06H
      int       21H
      endm

dir_console_input macro                  ;DIR_CONSOLE_INPUT
      mov       ah,7                     ;function 07H
      int       21H
      endm

read_kbd macro                           ;READ_KBD
      mov       ah,8                     ;function 08H
      int       21H
      endm

display macro string                     ;DISPLAY
      mov       dx,offset string
      mov       ah,9                     ;function 09H
      int       21H
      endm

get_string macro limit,string            ;GET_STRING
      mov       dx,offset string
      mov       string, limit
      mov       ah,0AH                   ;function 0AH
      int       21H
      endm

check_kbd_status macro                   ;CHECK_KBD_STATUS
      mov       ah,0BH                   ;function 0BH
      int       21H
      endm

flush_and_read_kbd macro switch          ;FLUSH_AND_READ_KBD
      mov       al,switch
      mov       ah,0CH                   ;function 0CH
      int       21H
      endm
```

```
reset_disk macro disk                    ;RESET DISK
      mov       ah,0DH                   ;function 0DH
      int       21H
      endm

select_disk macro disk                   ;SELECT_DISK
      mov       dl,disk[-65]
      mov       ah,0EH                   ;function 0EH
      int       21H
      endm

open macro fcb                           ;OPEN
      mov       dx,offset fcb
      mov       ah,0FH                   ;function 0FH
      int       21H
      endm

close macro fcb                          ;CLOSE
      mov       dx,offset fcb
      mov       ah,10H                   ;function 10H
      int       21H
      endm

search_first macro fcb                   ;SEARCH_FIRST
      mov       dx,offset fcb
      mov       ah,11H                   ;Function 11H
      int       21H
      endm

search_next macro fcb                    ;SEARCH_NEXT
      mov       dx,offset fcb
      mov       ah,12H                   ;function 12H
      int       21H
      endm

delete macro fcb                         ;DELETE
      mov       dx,offset fcb
      mov       ah,13H                   ;function 13H
      int       21H
      endm

read_seq macro fcb                       ;READ_SEQ
      mov       dx,offset fcb
      mov       ah,14H                   ;function 14H
      int       21H
      endm

write_seq macro fcb                      ;WRITE_SEQ
      mov       dx,offset fcb
      mov       ah,15H                   ;function 15H
      int       21H
      endm
```

```
create macro fcb                                ;CREATE
        mov         dx,offset fcb
        mov         ah,16H                      ;function 16H
        int         21H
        endm

rename macro fcb,newname                        ;RENAME
        mov         dx,offset fcb
        mov         ah,17H                      ;function 17H
        int         21H
        endm

current_disk macro                              ;CURRENT_DISK
        mov         ah,19H                      ;function 19H
        int         21H
        endm

set_dta macro buffer                            ;SET_DTA
        mov         dx,offset buffer
        mov         ah,1AH                      ;function 1AH
        int         21H
        endm

read_ran macro fcb                              ;READ_RAN
        mov         dx,offset fcb
        mov         ah,21H                      ;function 21H
        int         21H
        endm

write_ran macro fcb                             ;WRITE_RAN
        mov         dx,offset fcb
        mov         ah,22H                      ;function 22H
        int         21H
        endm

file_size macro fcb                             ;FILE_SIZE
        mov         dx,offset fcb
        mov         ah,23H                      ;function 23H
        int         21H
        endm

set_relative_record macro fcb                   ;SET_RELATIVE_RECORD
        mov         dx,offset fcb
        mov         ah,24H                      ;function 24H
        int         21H
        endm
```

```
set_vector macro interrupt,seg_addr,off_addr   ;SET_VECTOR
      push        ds
      mov         ax,seg_addr
      mov         ds,ax
      mov         dx,off_addr
      mov         al,interrupt
      mov         ah,25H                       ;function 25H
      int         21H
      pop         ds
      endm

ran_block_read macro fcb,count,rec_size        ;RAN_BLOCK_READ
      mov         dx,offset fcb
      mov         cx,count
      mov         word ptr fcb[0EH],rec_size
      mov         ah,27H                       ;function 27H
      int         21H
      endm

ran_block_write macro fcb,count,rec_size       ;RAN_BLOCK_WRITE
      mov         dx,offset fcb
      moc         cx,count
      mov         word ptr fcb[0EH],rec_size
      mov         ah,28H                       ;function 28H
      int         21H
      endm

parse macro string,fcb                         ;PARSE
      mov         si,offset string
      mov         di,offset fcb
      push        es
      push        ds
      pop         es
      mov         al,0FH
      mov         ah,29H                       ;function 29H
      int         21H
      pop         es
      endm

get_date macro                                 ;GET_DATE
      mov         ah,2AH                       ;function 2AH
      int         21H
      endm

set_date macro year,month,day                  ;SET_DATE
      mov         cx,year
      mov         dh,month
      mov         dl,day
      mov         ah,2BH                       ;function 2BH
      int         21H
      endm
```

```
get_time macro                                          ;GET_TIME
      mov         ah,2CH                                ;function 2CH
      int         21H
      endm
                                                        ;SET_TIME

set_time macro hour,minutes,seconds,hundredths
      mov         ch,hour
      mov         cl,minutes
      mov         dh,seconds
      mov         dl,hundredths
      mov         ah,2DH                                ;function 2DH
      int         21H
      endm

verify macro switch                                     ;VERIFY
      mov         al,switch
      mov         ah,2EH                                ;function 2EH
      int         21H
      endm
```

General

```
mov_string  macro  source,destination,num_bytes
                                                        ;MOV_STRING
                push      es
                mov       ax,ds
                mov       es,ax
                assume    es:data
                mov       si,offset source
                mov       di,offset destination
                mov       cx,num_bytes
           rep  movs      es:destination,source
                assume    es:nothing
                pop       es
                endm

convert         macro     value,base,destination  ;CONVERT
                local     table,start
                jmp       start
table           db        "0123456789ABCDEF"
start:          mov       al,value
                xor       ah,ah
                xor       bx,bx
                div       base
                mov       bl,al
                mov       al,cs:table[bx]
                mov       destination,al
                mov       bl,ah
                mov       al,cs:table[bx]
                mov       destination[1],al
                endm
```

```
_convert_to_binary  macro  string,number,value
                                                    ;CONVERT_TO_BINARY
                local      ten,start,calc,mult,no_mult
                jmp        start
ten             db         10
start:          mov        value,0
                xor        cx,cx
                mov        cl,number
                xor        si,si
calc:           xor        ax,ax
                mov        al,string[si]
                sub        al,48
                cmp        cx,2
                jl         no_mult
                push       cx
                dec        cx
mult:           mul        cs:ten
                loop       mult
                pop        cx
no_mult:        add        value,ax
                inc        si
                loop       calc
                endm

convert_date macro         dir_entry                ;CONVERT DATE
                mov        dx,word ptr dir_entry[25]
                mov        cl,5
                shr        dl,cl
                mov        dh,dir_entry[25]
                and        dh,1fh
                xor        cx,cx
                mov        cl,dir_entry[26]
                shr        cl,1
                add        cx,1980
                endm
```

Extended Example of MS-DOS System Calls

```
title DISK DUMP
zero                equ             0
disk_B              equ             1
sectors_per_read    equ             9
cr                  equ             13
blank               equ             32
period              equ             46
tilde               equ             126
     INCLUDE        B:CALLS.EQU
;
```

```
subttl DATA SEGMENT
page +
data                segment
;
input_buffer        db              9 dup (512 dup(?))
output_buffer       db              77 dup(" ")
                    db              0DH,0AH,"$"
start_prompt        db              "Start at sector: $"
sectors_prompt      db              "Number of sectors: $"
continue_prompt     db              "RETURN to continue $"
header              db              "Relative sector $"
end_string          db              0DH,0AH,0AH,07H,"ALL DONE$"
                                    ;DELETE THIS
crlf                db              0DH,0AH,"$"
table               db              "0123456789ABCDEF$"
;
ten                 db              10
sixteen             db              16
;
start_sector        dw              1
sector_num          label           byte
sector_number       dw              0
sectors_to_dump     dw              sectors_per_read
sectors_read        dw              0
;
buffer              label           byte
max_length          db              0
current_length      lb              0
digits              db              5 dup(?)
;
data                ends
;
subttl STACK SEGMENT
page +
stack               segment         stack
                    dw              100 dup(?)
stack_top           label           word
stack               ends
;
subttl MACROS
page +
;
INCLUDE B:CALLS.MAC
;BLANK LINE
blank_line          macro           number
                    local           print_it
                    push            cx
                    call            clear_line
                    mov             cx,number
print_it:           display         output_buffer
                    loop            print_it
                    pop             cx
                    endm
```

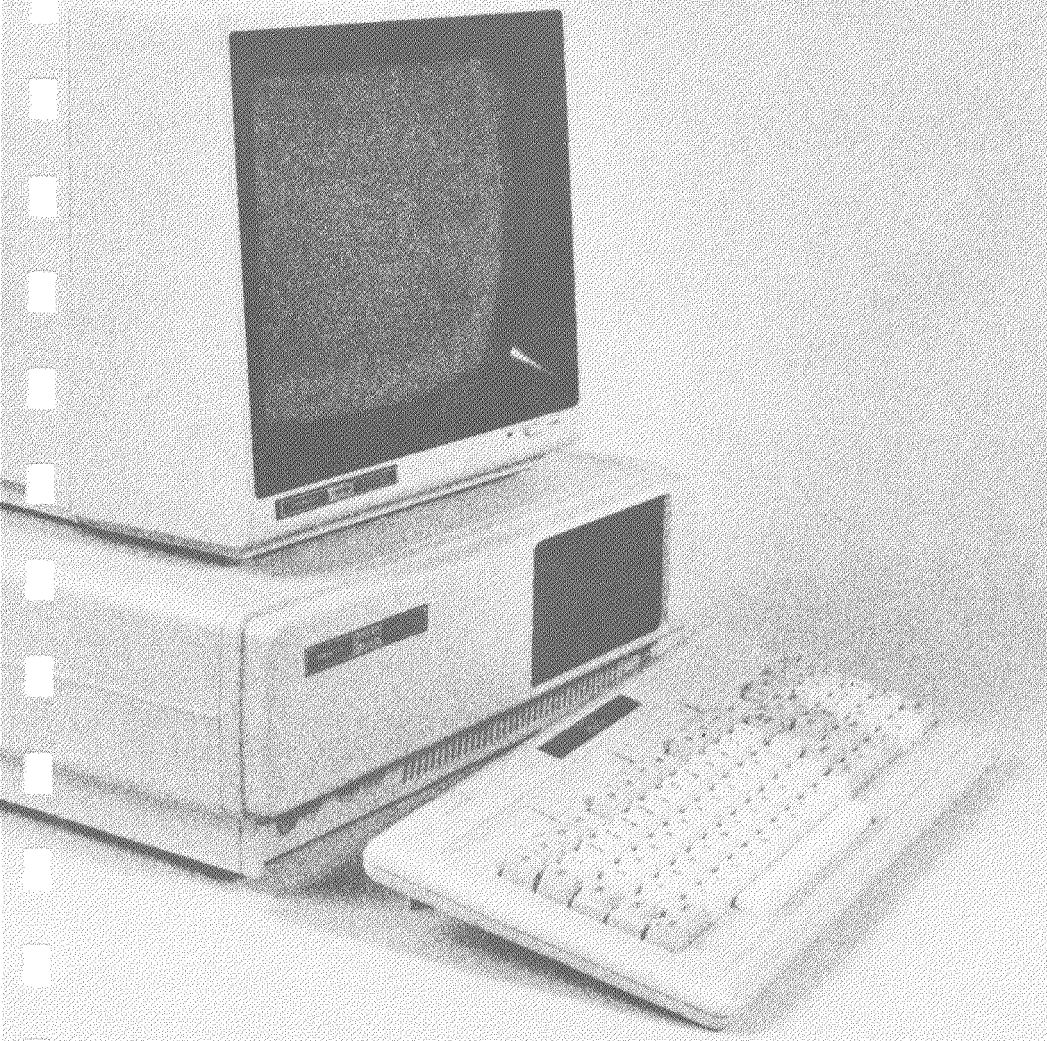
```
subttl ADDRESSABILITY
page +
code            segment
                assume          cs:code,ds:data,ss:stack
start:          mov             ax,data
                mov             ds,ax
                mov             ax,stack
                mov             ss,ax
                mov             sp,offset stack_top
;
                jmp             main_procedure
subttl PROCEDURES
page +
;
; PROCEDURES
; READ_DISK
read_disk       proc;
                cmp             sectors_to_dump,zero
                jle             done
                mov             bx,offset input_buffer
                mov             dx,start_sector
                mov             al,disk_b
                mov             cx,sectors_per_read
                cmp             cx,sectors_to_dump
                jle             get_sector
                mov             cx,sectors_to_dump
get_sector:     push            cx
                int             disk_read
                popf
                pop             cx
                sub             sectors_to_dump,cx
                add             start_sector,cx
                mov             sectors_read,cx
                xor             si,si
done:           ret
read_disk       endp
;CLEAR LINE
clear_line      proc;
                push            cx
                mov             cx,77
                xor             bx,bx
move_blank:     mov             output_buffer[bx]," "
                inc             bx
                loop            mov_blank
                pop             cx
                ret
clear_line      endp
;
;PUT_BLANK
put_blank       proc;
                mov             output buffer[di]," "
                inc             di
                ret
```

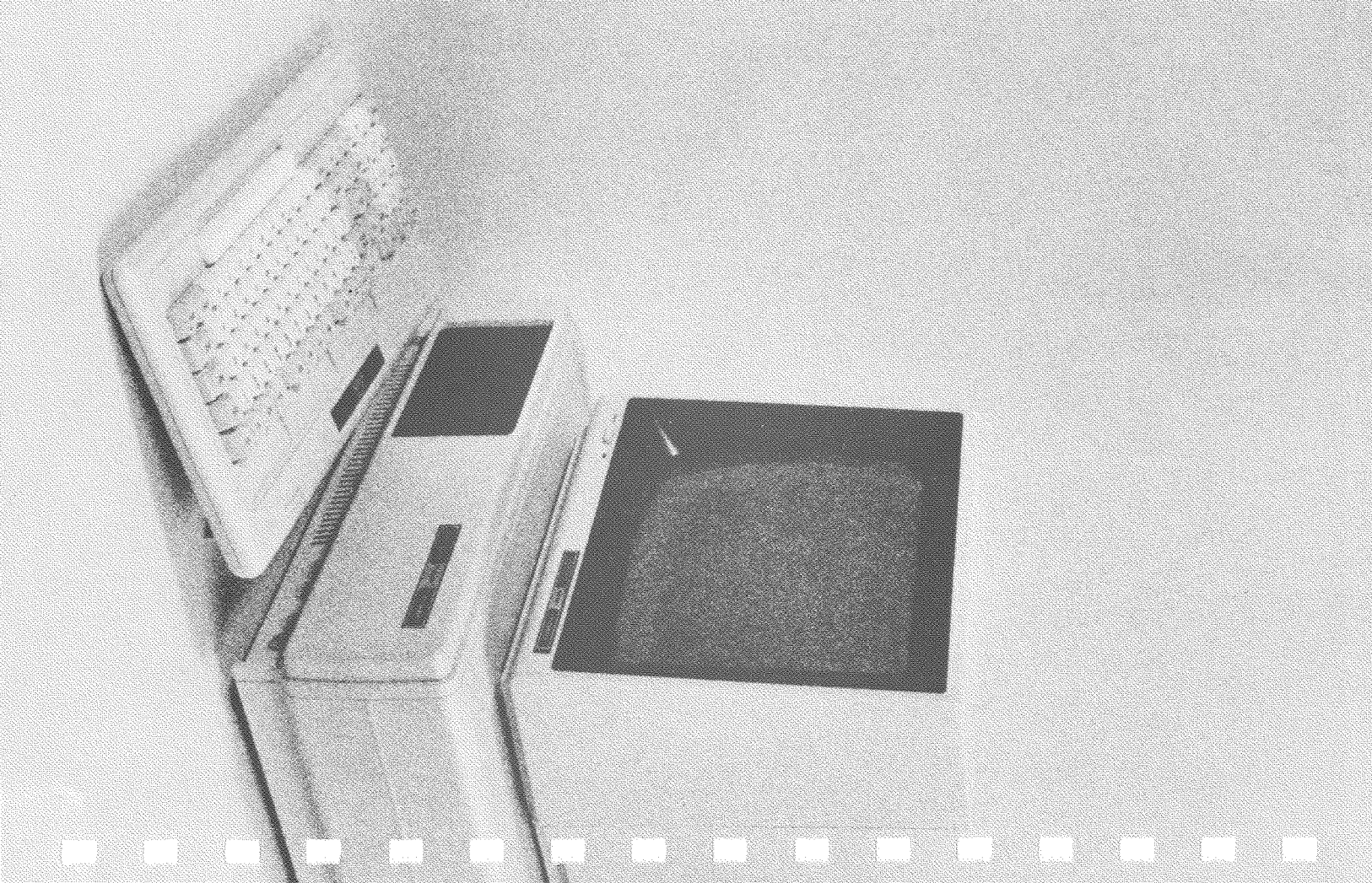
```
put_blank       endp
;
;
setup           proc;
                display         start_prompt
                get_string      4,buffer
                display         crlf
                convert_to_binary digits,
                current_length,start_sector
                mov             ax,start_sector
                mov             sector_number,ax
                display         sectors_prompt
                get_string      4,buffer
                convert_to_binary digits,
                current_length,sectors_to_dump
                ret
setup           endp
;
;CONVERT_LINE
convert_line    proc;
                push            cx
                mov             di,9
                mov             cx,16
convert_it:     convert         input_buffer[si],sixteen,
                                output_buffer[di]
                inc             si
                add             di,2
                call            put_blank
                loop            convert_it
                sub             si,16
                mov             cx,16
                add             di,4
display_ascii:  mov             output_buffer[di],period
                cmp             input_buffer[si],blank
                jl              non_printable
                cmp             input_buffer[si],tilde
                jg              non_printable
printable:      mov             dl,input_buffer[si]
                mov             output_buffer[di],dl
non_printable:  inc             si
                inc             di
                loop            display_ascii
                pop             cx
                ret
convert_line    endp
;
;DISPLAY_SCREEN
                proc;
                push            cx
                call            clear_line
;
                mov             cx,17
```

```
;I WANT length header
                    dec             cx
;minus 1 in cx

                    xor             di,di
mov_header:         mov             al,header[di]
                    mov             output_buffer[di],al
                    inc             di
                    loop            move_header ;FIX THIS!
;
                    convert         sector_num[1],sixteen,
                                    output_buffer[di]
                    add             di,2
                    convert         sector num,sixteen,
                                    output_buffer[di]
                    display         output_buffer
                    blank_line      2
                    mov             cx,16
dump_it:            call            clear_line
                    call            convert_line
                    display         output_buffer
                    loop            dump_it
                    blank_line      3
                    display         continue_prompt
                    get_char_no_echo
                    display         crlf
                    pop             cx
                    ret
display_screen      endp
;
;
; END PROCEDURES
subttl MAIN PROCEDURES
page +
main_procedure:     call            setup
check_done:         cmp             sectors_to_dump,zero
                    jng             all_done
                    call            read_disk
                    mov             cx,sectors_read
display_it:         call            display_screen
                    call            display_screen
                    inc             sector_number
                    loop            display_it
                    jmp             check_done
all_done:           display         end_string
                    get_char_no_echo
code                ends
                    end             start
```

MS™-DOS Control Blocks and Work Areas





MS-DOS CONTROL BLOCKS AND WORK AREAS

File Control Block (FCB)

The Program Segment Prefix includes room for two FCBs at offsets 5CH and 6CH. The system call descriptions refer to unopened and opened FCBs. An unopened FCB is one that contains only a drive specifier and filename, which can contain wild card characters (* and ?). An opened FCB contains all fields filled by the Open File function call (Function 0FH) or the Create File function call (16H).

The user program must set bytes 00H-0FH and 20H-24H. The operating system sets bytes 10H-1FH; they must not be altered by the user program.

The fields of the FCB are as follows:

Offset	Function
00H	Drive number. 1 means Drive A, 2 means Drive B, etc. If the FCB is to be used to create or open a file, this field can be set to 0 to specify the default drive; the Open File function call (Function 0FH) sets the field to the number of the default drive.
01H-08H	Filename. Consists of eight characters, left-justified and padded (if necessary) with blanks. If you specify a reserved device name (such as LST), do not include the optional colon.
09H-0BH	Filename extension. Consists of three characters, left-justified and padded (if necessary) with blanks. This field can be all blanks (no extension).
0CH-0DH	Current block. This is the number of the block (group of 128 records) that contains the current record. This field and the current record field (offset 20H) are used for sequential reads and writes. This field is set to 0 by the Open File function call.

0EH-0FH — Logical record size in bytes. Set to 128 by the Open File Function call. If the record size is not 128 bytes, you must set this field after opening the file.

10H-13H — File size in bytes. The first word of this field is the low-order part of the size.

14H-15H — Date of last write. The date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

Offset 15H

Y 15	Y 14	Y 13	Y 12	Y 11	Y 10	Y 9	M 8

Offset 14H

M 7	M 6	M 5	D 4	D 3	D 2	D 1	D 0

16H-17H — Time of last write. The time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

Offset 17H

H 15	H 14	H 13	H 12	H 11	M 10	M 9	M 8

Offset 16H

M 7	M 6	M 5	S 4	S 3	S 2	S 1	S 0

18H-1FH — Reserved for system use.

20H — Current record number. This is one of 128 records (0-127) in the current block. This field and the current block field (offset 0CH) are used for sequential reads and writes. It is not initialized by the Open File function call. You must set it before doing a sequential read or write to the file.

21H-24H	Relative record number. This is the number of currently selected record relative to the beginning of the file (starting with 0). This field is not initialized by the Open File function call. You must set it before doing a random read or write to the file. If the record size is less than 64 bytes, both words of this field are used. If the record size is 64 bytes or more, only the first three bytes are used. Note that if you use the FCB at offset 5CH of the Program Segment Prefix, the last bytes of the relative record field is also the first byte of the unformatted parameter area that starts at offset 80H (the default Disk Transfer Address).

Extended File Control Block

The Extended File Control Block is used to create or search for files in the disk directory that have special attributes. It adds the following 7-byte prefix to the FCB:

Byte	Function
-7	Flag byte. Contains FFH to indicate this is an extended FCB.
-6 to -2	Reserved.
-1	Attribute byte. See the section on the disk directory under "MS-DOS Disk Allocation" for the meaning of this byte.

If an extended FCB is referenced by a function call, the register containing the reference should point to the first byte of the prefix.

Program Segment

When you enter an external command or execute a program through the EXEC function call, MS-DOS determines the lowest available free memory address to use as the start of the program. This area is called the Program Segment.

At offset 0 within the Program Segment, MS-DOS builds the 256-byte Program Segment Prefix control block. At offset 200H, EXEC loads the program. An .**EXE**file with minalloc and maxalloc both set to zero is loaded as high as possible.

The program returns from EXEC by one of four methods:

- By a long jump to offset 0 in the Program Segment Prefix.
- By issuing an INT 20H with CS:0 pointing at the PSP
- By issuing an INT 21H with AH = 0 and with CS:0 pointing at the PSP, or with AH = 4CH
- By a long call to location 50H in the Program Segment Prefix with AH = 0 or 4CH

All programs must ensure the CS register contains the segment address of the Program Segment Prefix when using any of these methods except function call 4CH. For this reason, function call 4CH is the preferred method.

All four methods result in transferring control to the program that issued the EXEC. During this returning process, interrupt vectors 22H, 23H, and 24H (Terminate Address, CONTROL-C Exit Address, and Fatal Error Abort Address) are restored from the values saved in the Program Segment Prefix of the terminating program. Control is then given to the terminate address. If this is a program returning to COMMAND.COM, control transfers to its resident portion. If a batch file was in process, it is continued. Otherwise, COMMAND.COM performs a checksum on the transient portion, reloads it if necessary, issues the system prompt, and waits for you to type the next command.

When a program receives control, the following conditions are in effect:

For all programs:

- The segment address of the passed environment is contained at offset 2CH in the Program Segment Prefix.
- The environment is a series of ASCII strings (totaling less than 32K) in the form:

  NAME = parameter

Each string is terminated by a byte of zeroes, and the set of strings is terminated by another byte of zeroes. The environment built by the command processor contains at least a COMSPEC = string. (The parameters on COMSPEC define the path used by MS-DOS to locate COMMAND.COM on disk.) The last PATH and PROMPT commands issued will also be in the environment, along with any environment strings defined with the MS-DOS SET command.

The environment that is passed is a copy of the invoking process environment. If your application uses a "keep process" concept, you should be aware that the copy of the environment passed to you is static. It will not change even if subsequent SET, PATH, or PROMPT commands are issued.

- Offset 50H in the Program Segment Prefix contains code to call the MS-DOS function dispatcher. By placing the desired function call number in AH, a program can issue a far call to offset 50H to invoke an MS-DOS function, rather than issuing an Interrupt 21H. Since this is a call and not an interrupt, MS-DOS may place any code appropriate to making a system call at this position. This makes the process of calling the system portable.

- The Disk Transfer Address (DTA) is set to 80H (the default DTA in the Program Segment Prefix).

- File control blocks at 5CH and 6CH are formatted from the first two parameters typed when the command was entered. If either parameter contained a pathname, then the corresponding FCB contains only a valid drive number. The filename field will not be valid.

  An unformatted parameter area at 81H contains all the characters typed after the command (including leading and embedded delimiters). The byte at 80H is set to the number of characters. If the <, >, or parameters are typed on the command line, they (and the filenames associated with them) do not appear in this area. Redirection of standard input and output is transparent to applications.

- Offset 6 (one word) contains the number of bytes available in the segment.

Register AX indicates whether or not the drive specifiers (entered with the first two parameters) are valid, as follows:

AL = FFH if the first parameter contained an invalid drive specifier (otherwise, AL = 00H).

AH = FFH if the second parameter contained an invalid drive specifier (otherwise, AH = 00H).

Offset 2 (one word) contains the segment address of the first byte of unavailable memory. Programs must not modify addresses beyond this point unless the addresses were obtained by allocating memory via the Allocate Memory function call (48H).

For executable (.EXE) programs:

- Registers DS and ES are set to point to the Program Segment Prefix.
- Registers CS, IP, SS, and SP are set to the values passed by MS-LINK.

For executable (.COM) programs:

- All four segment registers contain the segment address of the initial allocation block that starts with the Program Segment Prefix control block.
- All of user memory is allocated to the program. If the program invokes another program through the EXEC function call (4BH), it must first free some memory through the Set Block function call (4AH) to provide space for the program being executed.
- The Instruction Pointer (IP) is set to 100H.
- The Stack Pointer register is set to the end of the program's segment. The segment size at offset 6 is reduced by 100H to allow for a stack of that size.
- A word of zeroes is placed on top of the stack. This is to allow a user program to exit to COMMAND.COM by doing a RET instruction last. This assumes the user has maintained stack and code segments for the program.

Program Segment Prefix

Following is an illustration of the Program Segment Prefix. Programs must not alter any part of the PSP below offset 5CH.

00H	INT 20 H	End of allocation block	Reserved	Long call to MS-DOS function dispatcher (5 bytes)
08H		Terminate address (IP,CS)		CONTROL-C exit address (IP)
10H	CONTROL-C Exit address (CS)	Hard error exit address (IP,CS)		
2CH	Environmental pointer Used by MS-DOS			
5CH	Formatted Parameter Area 1 formatted as standard unopened FCB			
6CH	Formatted Parameter Area 2 formatted as standard unopened FCB (overlaid if FCB at 5CH is opened)			
80H	Unformatted Parameter Area (default Disk Transfer Area)			

MS™-DOS Installation and Command Processor

Chapter 3

MS-DOS INITIALIZATION AND COMMAND PROCESSOR

When the system is reset or started with an MS-DOS disk in Drive A, the ROM (Read Only Memory) bootstrap gains control. The boot sector is read from the disk into memory and given control. The IO.SYS and MSDOS.SYS files are then read into memory, and the boot process begins.

The Command Processor

The command processor supplied with MS-DOS (file COMMAND.COM) consists of three parts:

- A resident portion resides in memory immediately following MSDOS.SYS and its data area. This portion contains routines to process Interrupts 23H (CONTROL-C Exit Address) and 24H (Fatal Error Abort Address), as well as a routine to reload the transient portion, if needed. All standard MS-DOS error handling is done within this portion of COMMAND.COM, including the display of error messages and processing the Abort, Retry, or Ignore message replies.
- An initialization portion follows the resident portion. During start-up, the initialization portion is given control. It contains the AUTOEXEC file processor setup routine. The initialization portion determines the segment address at which programs can be loaded. It is overlaid by the first program COMMAND.COM loads because it is no longer needed.
- A transient portion is loaded at the high end of memory. This part contains all of the internal command processors and the batch file processor. The transient part of the command processor produces the system prompt (such as A>), reads the command from the keyboard (or batch file), and causes the command to be executed. For external commands, it builds a command line and issues the EXEC function call (function call 4BH) to load and transfer control to the program.

MS™-DOS Disk Allocation

F1
F2
F3
F4
F5
F6
F7
F8
Esc
1 !
2 @
3 #
4 $
5 %
6 ^
7 &
8 *
TAB
Q
W
E
R
T
Y
U
I
CTRL
A
S
D
F
G
H
J
SHIFT
Z
X
C
V
B
N
M
CAPS

Chapter 4

MS-DOS DISK ALLOCATION

The MS-DOS area on a diskette is formatted as follows:

Reserved area - variable size
First copy of File Allocation Table - variable size
Second copy of File Allocation Table - variable size (optional)
Additional copies of File Allocation Table - variable size (optional)
Root directory - variable size
File data area

Space for a file in the data area is not pre-allocated. The space is allocated one cluster at a time, as needed. A cluster consists of one or more consecutive sectors. All of the clusters for a file are "chained" together in the File Allocation Table (FAT).

A second copy of the FAT is usually kept for consistency. If the disk should develop a bad sector in the first FAT, the second can be used. This prevents loss of data due to an unusable disk.

The clusters are arranged on disk to minimize head movement for multi-sided media. All of the space on a track (or cylinder) is allocated before moving on to the next track. This is done by allocating all the sectors sequentially on the lowest-numbered head, then all the sectors on the next head, and so on until all sectors on all heads of the track are used. The next sector to use will be sector 1 on head 0 of the next track.

For diskettes, the following table can be used:

Number of Sides	Sectors per Track	FAT size in Sectors	Directory Sectors Cluster	Directory Entries	Sectors per
1	8	1	4	64	1
2	8	1	7	112	2
1	9	2	4	64	1
2	9	2	7	112	2

MS-DOS Disk Directory

FORMAT builds the root directory for all disks. Its location on disk and the maximum number of entries are dependent on the media.

Since directories other than the root directory are regarded as files by MS-DOS, there is no limit to the number of entries they may contain.

All directory entries are 32 bytes in length, and are in the following format:

00H-07H Filename. 8 characters, left-aligned and padded, if necessary, with blanks. The first byte of this field indicates the file status as follows:

- 00H The directory entry has never been used. This is used to limit the length of directory searches, for performance reasons.
- E5H The directory entry has been used, but the file has been erased.
- 2EH The entry is for a directory. If the second byte is also 2EH, then the cluster field contains the cluster number of this directory's parent directory (0000H if the parent directory is the root directory). Otherwise, bytes 01H through 0AH are all spaces, and the cluster field contains the cluster number of this directory.

Any other character is the first character of a filename.

08H-0AH Filename extension.

0B File Attribute. The attribute byte is mapped as follows:

- 01H File is marked read only. An attempt to open the file for writing using the Open File function call (function call 3DH) results in an error code returned. This value can be used along with other values below. Attempts to delete the file with the Delete File (13H) or Delete a Directory Entry (41H) function call will also fail.

02H Hidden file. The file is excluded from normal directory searches.

04H System file. The file is excluded from normal directory searches.

08H The entry contains the volume label in the first 11 bytes. The entry contains no other usable information (except date and time of creation) and may exist only in the root directory.

10H The entry defines a sub-directory, and is excluded from normal directory searches.

20H Archive bit. The bit is set to 1 whenever the file has been written to and closed.

Note: The system files (IO.SYS and MSDOS.SYS) are marked as read only, hidden, and system files. Files can be marked hidden when they are created. Also, the read only, hidden, system, and archive attributes may be changed through the Change Attributes function call (43H).

0CH-15H Reserved.

16H-17H Time the file was created or last updated. The hour, minutes, and seconds are mapped into two bytes as follows:

Offset 17H

H 7	H 6	H 5	H 4	H 3	M 2	M 1	M 0

Offset 16H

M 7	M 6	M 5	S 4	S 3	S 2	S 1	S 0

where:

HHHHH is the binary number of hours (0-23)
MMMMMM is the binary number of minutes (0-59)
SSSSS is the binary number of two-second increments

18H-19H Date the file was created or last updated. The year, month, and day are mapped into two bytes as follows:

Offset 19H

Y	Y	Y	Y	Y	Y	Y	M
7	6	5	4	3	2	1	0

Offset 18H

M	M	M	D	D	D	D	D
7	6	5	4	3	2	1	0

where:

YYYYYYY is 0-119 (1980-2099)
MMMM is 1-12
DDDDD is 1-31

1AH-1BH Starting cluster; the cluster number of the first cluster in the file.

Note that the first cluster for data space on all disks is cluster 002.

The cluster number is stored with the least significant byte first.

Note: Refer to the section "How to Use the File Allocation Table" for details about converting cluster numbers to logical sector numbers.

1CH-1FH File size in bytes. The first word is the low-order part of the size.

File Allocation Table (FAT)

This section is included for system programmers who wish to write installable device drivers. It explains how MS-DOS uses the File Allocation Table to convert the cluster numbers of a file to logical sector numbers. The driver is then responsible for locating the logical sector on disk. Programs must use the MS-DOS file management function calls for accessing files. Programs that access the FAT are not guaranteed to be upwardly compatible with future releases of MS-DOS.

The File Allocation Table contains a 12-bit entry (1.5 bytes) for each cluster on the disk. The first two FAT entries map a portion of the directory; these FAT entries indicate the size and format of the disk.

The second and third bytes always contain FFH.

The third FAT entry, which starts at offset 04H, begins the mapping of the data area (cluster 002). Files in the data area are not always written sequentially on the disk. The data area is allocated one cluster at a time, skipping over clusters already allocated. The first free cluster found will be the next cluster allocated, regardless of its physical location on the disk. This permits the most efficient utilization of disk space because clusters made available by the erasing of files can be allocated for new files.

Each FAT entry contains three hexadecimal characters. Any of the following combinations is possible:

000	The cluster is unused and available.
FF7	The cluster has a bad sector in it. MS-DOS will not allocate such a cluster. CHKDSK counts the number of bad clusters for its report. These bad clusters are not part of any allocation chain.
FF8-FFF	Indicates the last cluster of a file.
XXX	Any other characters that are the cluster number of the next cluster in the file. The cluster number of the first cluster in the file is kept in the file's directory entry.

The File Allocation Table always begins on the first sector after the reserved sectors. If the FAT is larger than one sector, the sectors are contiguous. Two copies of the FAT are usually written for data integrity. The FAT is read into one of the MS-DOS buffers whenever needed (open, read, write, etc.). For performance reasons, this buffer is given a high priority to keep it in memory as long as possible.

How to Use the File Allocation Table

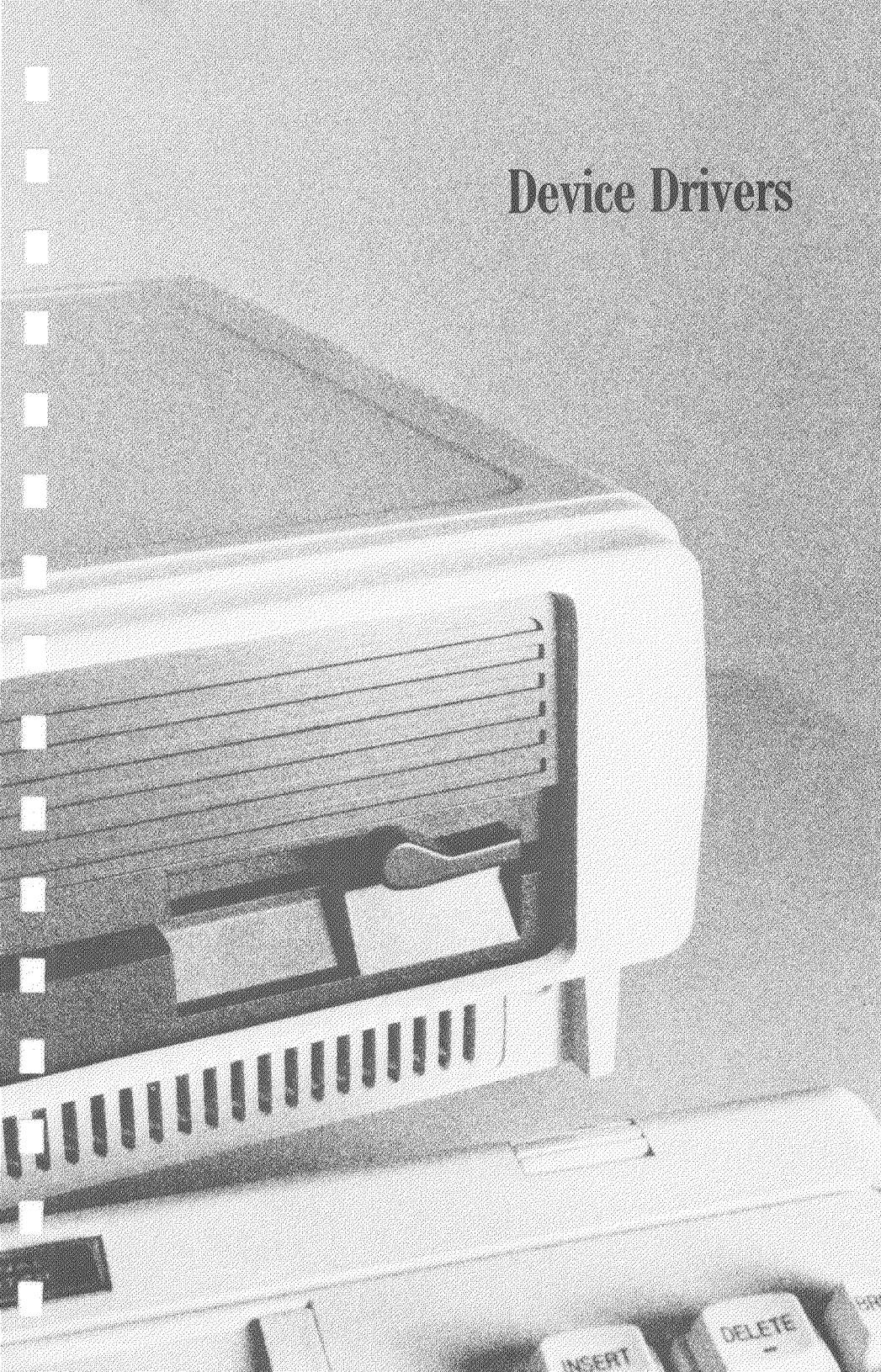
To find the starting cluster of the file, use the directory entry. Next, to locate each subsequent cluster of the file:

1. Multiply the cluster number just used by 1.5 (each FAT entry is 1.5 bytes long).
2. The whole part of the product is an offset into the FAT, pointing to the entry that maps the cluster just used. That entry contains the cluster number of the next cluster of the file.
3. Use a MOV instruction to move the word at the calculated FAT offset into a register.
4. If the last cluster used was an even number, keep the low-order 12 bits of the register by ANDing it with FFF; otherwise, keep the high-order 12 bits by shifting the register right 4 bits with a SHR instruction.
5. If the resultant 12 bits are FF8H-FFFH, the file contains no more clusters. Otherwise, the 12 bits contain the cluster number of the next cluster in the file.

To convert the cluster number to a logical sector number (relative sector, such as that used by Interrupts 25H and 26H and by DEBUG):

1. Subtract 2 from the cluster number.
2. Multiply the result by the number of sectors per cluster.
3. Add to this result the logical sector number of the beginning of the data area.

Device Drivers



DELETE
INSERT

Chapter 5

DEVICE DRIVERS

A device driver is a binary file containing code to manipulate hardware devices and provide consistent interfacing to MS-DOS. The driver has a special header that identifies the device, defines the strategy and interrupt entry points, and describes various attributes of the device.

The *.COM* file must not use 100H as the driver ORG. Because it does not use the Program Segment Prefix, the device driver file must have an origin of zero (ORG 0 or no ORG statement).

Types of Devices

There are two kinds of devices: character devices and block devices.

Character devices are designed to perform serial character I/O. They have names such as CON, AUX, and CLOCK. You open channels (handles or FCBs) for I/O operations to them.

Block devices are system *disk drives*. They perform random I/O in pieces called blocks (usually the physical sector size). These devices are not named as the character devices are, and therefore cannot be opened directly. They are identified instead by the drive letters (A, B, C, etc.).

Block devices can consist of one or more units with a single driver responsible for one or more disk drives. For example, block device driver ALPHA may be responsible for Drives A, B, C, and D (four units (0-3) are defined and are identified by four drive letters). The position of the driver in the list of all drivers determines which units correspond to which drive letters. If driver ALPHA is the first first block driver in the device list, and it defines 4 units (0-3), then they will be A, B, C, and D. If BETA is the second block driver and defines three units (0-2), then they will be E, F, and G, and so on. The theoretical limit is 63 block devices, but after 26 the drive letters are unconventional (such as], /, and ^).

Device Headers

A Device Header is required at the beginning of a device driver and looks like this:

WORD	Pointer to next Device Header (Must be set to -1)
WORD	Attributes Bit 15 = 0 if block device Bit 15 = 1 if character device If bit 15 is 1: Bit 0 = 1 if current sti device Bit 1 = 1 if current sto device Bit 2 = 1 if current NUL device Bit 3 = 1 if current CLOCK device Bit 4 = 1 if special Bit 5-12 Reserved; must be set to 0 Bit 14 is the IOCTL bit Bit 13 is the NON IBM FORMAT bit
WORD	Pointer to device strategy entry point
WORD	Pointer to device interrupt entry point
8 BYTES	Character device name field. Character devices set a device name. For block devices the first byte is the number of units.

Note that the device entry points are words. They must be offsets from the same segment number used to point to this table. For example, if XXX:YYY points to the start of this table, then XXX:strategy and XXX:interrupt are the entry points.

Pointer To Next Device Header Field

The pointer to the next Device Header field is a double word field (offset followed by segment) that is set by MS-DOS to point at the next driver in the system list at the time the device driver is loaded. It is important that this field be set to -1 prior to load (when it is on the disk as a file) unless there is more than one device driver in the file. If there is more than one driver in the file, the first word of the double word pointer should be the offset of the next driver's Device Header.

If there is more than one device driver in the .COM file, the last driver in the file must have the pointer to the next Device Header field set to -1.

Attribute Field

The attribute field tells the system whether this device is a block or character device (bit 15). Most other bits are used to give selected character devices certain special treatment. (Note that these bits mean nothing on a block device.) For example, suppose you have a new device driver that you want to be the standard input and output. Besides installing the driver, you must tell MS-DOS that you want your new driver to override the current standard input and standard output (the CON device). You do this by setting bits 0 and 1 to 1. You could also install a new CLOCK device by setting that attribute bit.

Although there is a NUL device attribute, the NUL device cannot be reassigned. This attribute exists so that MS-DOS can determine if the NUL device is being used.

The NON IBM FORMAT bit applies only to block devices and affects the operation of the BUILD BPB (Bios Parameter Block) device call. (Refer to "MEDIA CHECK and BUILD BPB" later in this chapter for further information on this call.)

The IOCTL bit has meaning on both character and block devices. This bit tells MS-DOS whether the device can handle control strings (via the IOCTL function call, Function 44H).

If a driver cannot process control strings, it should initially set the IOCTL bit to 0. This tells MS-DOS to return an error if an attempt is made (via Function 44H) to send or receive control strings to this device. A device which can process control strings should initialize the IOCTL bit to 1. For drivers of this type, MS-DOS will make calls to the IOCTL input and output device functions to send and receive IOCTL strings.

The IOCTL functions allow data to be sent and received by he device for its own use (for example, to set baud rate, stop bits, and form length), instead of passing data over the device channel as does a normal read or write. It is up to the device to interpret the passed information, but it must not be treated as a normal I/O request.

Strategy and Interrupt Routines

These two fields are the pointers to the entry points of the strategy and interrupt routines. They are word values, so they must be in the same segment as the Device Header.

Name Field

This 8-byte field contains the name of a character device or the number of units of a block device. If it is a block device, the number of units can be put in the first byte. This is optional, because MS-DOS will fill in this location with the value returned by the driver's INIT code. Refer to "Installation of Device Drivers" in this chapter for more information.

Creating a Device Driver

In order to create a device driver that MS-DOS can install, you must write a binary file with a Device Header at the beginning of the file. Note that for device drivers, the code should not be originated at 100H, but rather at 0. The link field (pointer to the next Device Header) should be -1, unless there is more than one device driver in the file. The attribute field and entry points must be set correctly.

If it is a character device, the name field should be filled in with the name of that character device. The name can be any legal 8-character filename.

MS-DOS always processes installable device drivers before handling the default devices, so to install a new CON device, simply name the device CON and set the standard input device and standard output device bits in the attribute word on a new CON device. The scan of the device list stops on the first match, so the installable device driver takes precedence.

Because MS-DOS can install the driver anywhere in memory, care must be taken in any far memory references. You should not expect that your driver will always be loaded in the same place every time.

Installation of Device Drivers

MS-DOS allows new device drivers to be installed dynamically at boot time. This is accomplished by INIT code in the BIOS, which reads and processes the CONFIG.SYS file.

MS-DOS calls a device driver by making a far call to its strategy entry point, and passes the information describing the functions of the device driver in a Request Header.

This structure allows you to program an interrupt-driven device driver. For example, you may want to perform local buffering in a printer.

MS-DOS passes a pointer to the Request Header in ES:BX. This is a fixed-length header, followed by data pertinent to the operation being performed. Note that it is the device driver's responsibility to preserve the machine state (for example, save all registers on entry and restore them on exit). There is enough room on the stack to do about 20 pushes. If more stack space is needed, the driver should set up its own stack.

Request Header

BYTE	Length of record Length in bytes of this Request Header
BYTE	Unit code The subunit the operation is for (minor device) (no meaning on character devices)
BYTE	Command code
WORD	Status
8 bytes	RESERVED

Unit Code Field

The unit code field identifies which unit in your device driver the request is for. For example, if your device driver has 3 units defined, then the possible values of the unit code field would be 0, 1, and 2.

Command Code Field

The command code field in the Request header can have the following values:

Command Code	Function
0	INIT
1	MEDIA CHECK (block only, NOP for character)
2	BUILD BPB (block only, NOP for character)
3	IOCTL input (called only if IOCTL bit is 1)
4	INPUT (read)
5	NON-DESTRUCTIVE INPUT NO WAIT (character devices only)
6	INPUT STATUS (character devices only)
7	INPUT FLUSH (character devices only)
8	OUTPUT (write)
9	OUTPUT (write) with verify
10	OUTPUT STATUS (character devices only)
11	OUTPUT FLUSH (character devices only)
12	IOCTL output (called only if IOCTL bit is 1)

MEDIA CHECK and BUILD BPB

MEDIA CHECK and BUILD BPB are used with block devices only.

MS-DOS calls MEDIA CHECK first for a drive unit. MS-DOS passes its current media descriptor byte (refer to the section "Media Descriptor Byte" later in this chapter). MEDIA CHECK returns one of the following results:

- Media Not Changed — current DBP (Disk Parameter Block) and media byte are OK.
- Media Changed — Current DPB and media are wrong. MS-DOS invalidates any buffers for this unit and calls the device driver to build the DPB with media byte and buffer.
- Not Sure — If there are dirty buffers (buffers with changed data, not yet written to disk) for this unit, MS-DOS assumes the DBP and media byte are OK (media not changed). If nothing is dirty, MS-DOS assumes the media has changed. It invalidates any buffers for the unit and calls the device driver to build the BPB with media byte and buffer.

- Error — If an error occurs, MS-DOS sets the error code accordingly.

MS-DOS will call BUILD BPB under the following conditions:

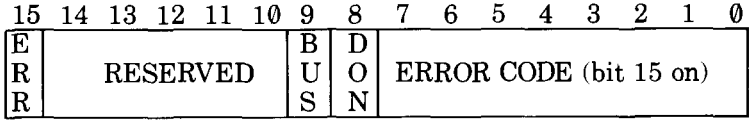
- If "Media Changed" is returned
- If "Not Sure" is returned and there are no dirty buffers

The BUILD BPB call also gets a pointer to a one-sector buffer. What this buffer contains is determined by the NON IBM FORMAT bit in the attribute field. If the bit is zero (device is IBM format-compatible), then the buffer contains the first sector of the first FAT. The FAT ID byte is the first byte of this buffer.

Note: The BPB must be the same, as far as location of the FAT is concerned, for all possible media because this first FAT sector must be read before the actual BPB is returned. If the NON IBM FORMAT bit is set, then the pointer points to one sector of scratch space (which may be used for anything).

Status Field

The following figure illustrates the status word in the Request Header:



The status word is zero on entry and is set by the driver interrupt routine on return.

Bit 8 is the done bit. When set, it means the operation is complete. The driver sets it to 1 when it exits.

Bit 15 is the error bit. If it is set, then the low 8 bits indicate the error. The error are:

00H Write Protect Violation
01H Unknown Unit
02H Drive Not Ready
03H Unknown Command
04H CRC Error
05H Bad Drive Request Structure Length
06H Seek Error

07H Unknown Media
08H Sector Not Found
09H Printer Out of Paper
0AH Write Fault
0BH Read Fault
0CH General Failure

Bit 9 is the busy bit, which is set only by status calls.

For output on character devices: If bit 9 is 1 on return, a write request (if made) would wait for completion of a current request. If it is 0, there is no current request, and a write request (if made) would start immediately.

For input on character devices with a buffer: If bit 9 is 1 on return, a read request (if made) would go to the physical device. If it is 0 on return, then there are characters in the device buffer and a read would return quickly. It also indicates that the user has typed something. MS-DOS assumes that all character devices have an input type-ahead buffer. Devices that do not have a type-ahead buffer should always return busy = 0 so that MS-DOS will not continuously wait for something to get into a buffer that does not exist.

One of the functions defined for each device is INIT. This routine is called only once, when the device is installed. The INIT routine returns a location (DS:DX), which is a pointer to the first free byte of memory after the device driver (similar to "Keep Process"). This pointer method can be used to delete initialization code after it has been used in order to save space.

Block devices are installed the same way and also return a first free byte pointer as described above. Additional information is also returned, such as the number of units.

The number of units determines logical device names. For example, if the current maximum logical device letter is F at the time of the install call and the INIT routine returns 4 as the number of units, then the units will have logical names G, H, I and J. This mapping is determined by the position of the driver in the device list and the number of units on the device (stored in the first byte of the device name field).

A pointer to a BPB (BIOS Parameter Block) pointer array is also returned. There is one table for each unit defined. These blocks will be used to build an internal DOS data structure for each of the units. The pointer passed to the DOS from the driver points to an array of*n*word pointers to BPBs, where *n* is the number of units defined. In this way, if all units are the same, all of the pointers can point to the same BPB, in order to save space.

Note that this array must be protected (below the free pointer set by the return), since an internal DOS structure will be built starting at the byte pointed to by the free pointer. The sector size defined must be less than or equal to the maximum sector size defined at default BIOS INIT time. If it is not, the install will fail.

The last thing that INIT of a block device must pass back is the media descriptor byte. This byte means nothing to MS-DOS, but is passed to devices so that they know what parameters MS-DOS is currently using for a particular drive unit.

Block devices may take several approaches; they may be dumb or smart. A dumb device defines a unit (and therefore an internal DOS structure) for each possible media drive combination. For example, unit 0 = drive 0 single side, unit 1 = drive 0 double side. For this approach, media descriptor bytes mean nothing. A smart device allows multiple media per unit. In this case, the BPB table returned at INIT must define space large enough to accommodate the largest possible media supported. Smart drivers will use the media descriptor byte to pass information about what media is currently in a unit.

Function Call Parameters

All strategy routines are called with ES:BX pointing to the Request Header. The interrupt routines get the pointers to the Request Header from the queue in which they are stored by the strategy routines. The command code in the Request Header tells the driver which function to perform.

All DWORD pointers are stored offset first, then segment.

INIT

Command code = 0

ES:BX

13-BYTE	Request Header
BYTE	Number of units
DWORD	Break address
DWORD	Pointer to BPB array (Not set by character devices)

The number of units, break address, and BPB pointer are set by the driver. On entry, the DWORD to be set to the BPB array (on block devices) points to the character following '=' on the line in CONFIG.SYS that loaded this device. This allows drivers to scan CONFIG.SYS invocation line for arguments.

If there are multiple device drivers in a single .COM file, the ending address returned by the last INIT called will be used by MS-DOS. It is recommended that all device drivers in a single .COM file return the same ending address.

MEDIA CHECK

Command Code = 1

ES:BX

13-BYTE	Request Header
BYTE	Media descriptor from DPB
BYTE	Returned

In addition to setting the status word, the driver must set the return byte to one of the following:

-1 Media has been changed
0 Don't know if media has been changed
1 Media has not been changed

If the driver can return -1 or 1 (by having a door-lock or other interlock mechanism), MS-DOS performance is enhanced because MS-DOS does not need to re-read the FAT for each directory access.

BUILD BPB (BIOS Parameter Block)

Command code = 2

ES:BX

13-BYTE Request Header

BYTE Media descriptor from DPB

DWORD transfer address
(Points to one sector worth of scratch space or first sector of FAT depending on the value of the NON IBM FORMAT bit)

DWORD pointer to BPB

If the NON IBM FORMAT bit of the device is set, then the DWORD transfer address points to a one-sector buffer, which can be used for any purpose. If the NON IBM FORMAT bit is Ø, then this buffer contains the first sector of the first FAT and the driver must not alter this buffer.

If IBM compatible format is used (NON IBM FORMAT BIT = Ø), the first sector of the first FAT must be located at the same sector on all possible media. This is because the FAT sector will be read before the media is actually determined. Use this mode if all you want is to read the FAT ID byte.

In addition to setting status word, the driver must set the pointer to the BPB on return.

The information relating to the BPB for a particular piece of media is kept in the boot sector for the media. The following chart illustrates the format of the boot sector:

	3-BYTE	Near JUMP to boot code
	8 BYTES	OEM name and version
BPB	WORD	Bytes per sector
	BYTE	Sectors per allocation unit
	WORD	Reserved sectors
	BYTE	Number of FATs
	WORD	Number of root directory entries
	WORD	Number of sectors in logical image
	BYTE	Media descriptor
	WORD	Number of FAT sectors
	WORD	Sectors per track
	WORD	Number of heads
	WORD	Number of hidden sectors

The three words at the end (sectors per track, number of heads, and number of hidden sectors) are intended to help the BIOS understand the media. Sectors per track may be redundant (could be calculated from total size of the disk). Number of heads is useful for supporting different multi-head drives which have the same storage capacity but different numbers of surfaces. Number of hidden sectors may be used to support drive-partitioning schemes.

Media Descriptor Byte

The last two digits of the FAT ID are called the media descriptor byte. Currently, the media descriptor byte has been defined for a few media types, including 5-1/4" and 8" standard disks.

Although these media bytes map directly to FAT ID bytes (which are constrained to the 8 values F8H-FFH), media bytes can, in general, be any value in the range 00H-FFH.

READ or WRITE

Command codes = 3,4,8,9, and 12

ES:BX (Including IOCTL)

13-BYTE	Request Header
BYTE	Media descriptor from DPB
DWORD	Transfer address
WORD	Byte/sector count
WORD	Starting sector number (Ignored on character devices)

In addition to setting the status word, the driver must set the sector count to the actual number of sectors (or bytes) transferred. No error check is performed on an IOCTL I/O call. The driver must correctly set the return sector (byte) count to the actual number of bytes transferred.

The following applies to block device drivers:

Under certain circumstances the BIOS may be asked to perform a write operation of 64K bytes that seems to be a "wrap-around" of the transfer address in the BIOS I/O packet. This request arises because of an optimization added to the write code in MS-DOS. It occurs only on user writes that are within a sector size of 64K bytes on files "growing" past the current end of file. It is allowable for the BIOS to ignore the balance of the write that wraps around, if it so chooses. For example, a write of 10000H bytes' worth of sectors with a transfer address of XXXX:1 could ignore the last two bytes. A user program can never request I/O of more than FFFFH bytes and cannot wrap around (even to 0) in the transfer segment. Therefore, in this case, the last two bytes can be ignored.

NON-DESTRUCTIVE READ NO WAIT

Command code = 5

ES:BX

13-BYTE Request Header	
BYTE	Read from device

If the character device returns busy bit = 0 (characters in buffer), then the next character that would be read is returned. This character is not removed from the input buffer (hence the term non-destructive read). This call allows MS-DOS to look ahead one input character.

STATUS

Command codes = 6 and 10

ES:BX

13-BYTE Request Header

All the driver must do is set the status word and the busy bit.

FLUSH

Command codes = 7 and 11

ES:BX

13-BYTE Request Header

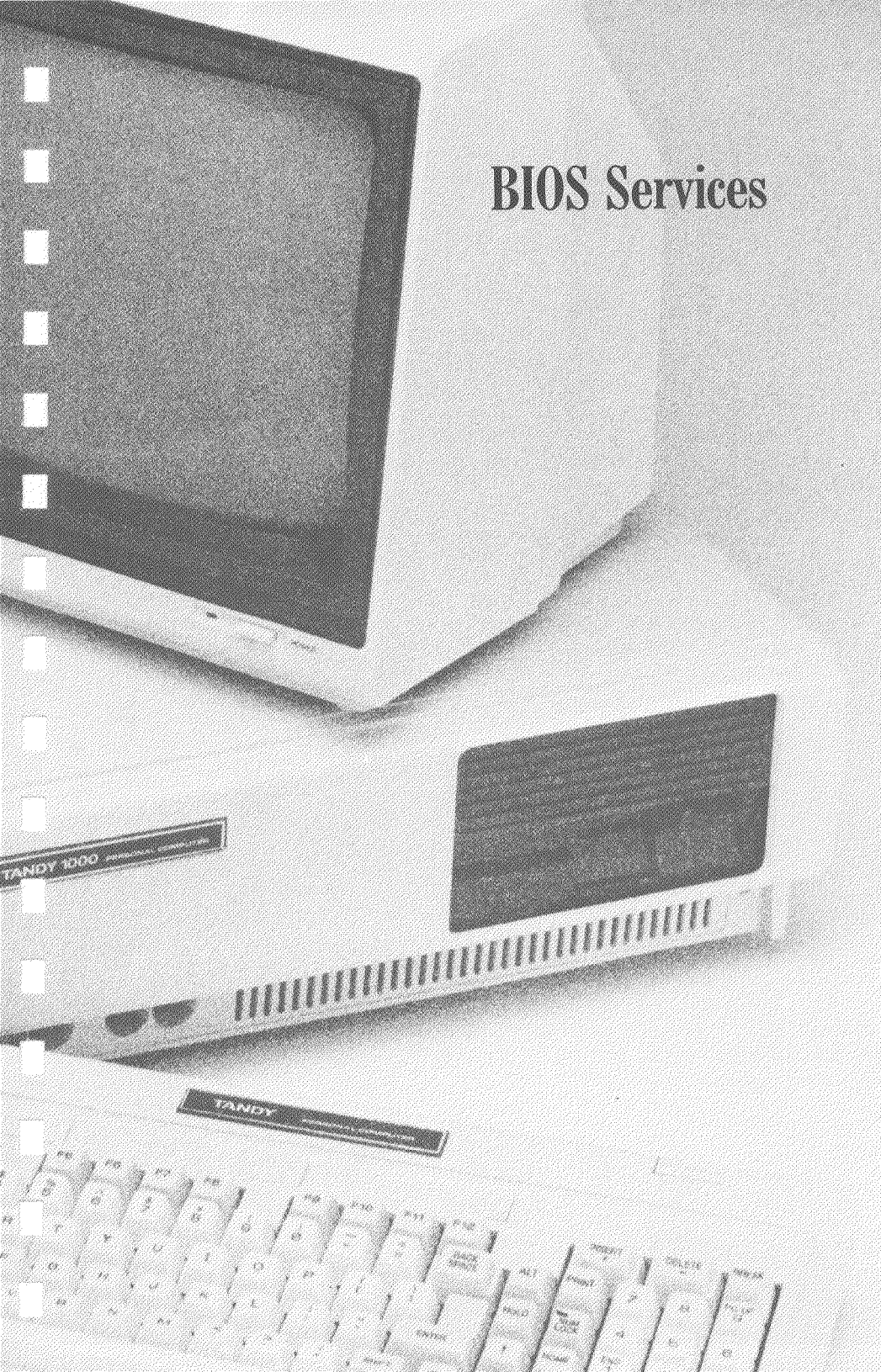
The FLUSH call tells the driver to flush (terminate) all pending requests. This call is used to flush the input queue on character devices.

The CLOCK Device

To allow for a real time clock board to be integrated into the system for TIME and DATE, there is a special device (determined by the attribute word) called the CLOCK device. The CLOCK device defines and performs functions like any other character device. Most functions will be: "set done bit, reset error bit, return."

When a read or write to this device occurs, exactly 6 bytes are transferred. The first two bytes are a word, which is the count of days since 1-1-80. The third byte is minutes, the fourth hours, the fifth 1/100 seconds, and the sixth seconds. Reading the CLOCK device gets the date and time; writing to it sets the date and time.

BIOS Services



TANDY 1000
TANDY

Chapter 6

BIOS SERVICES

Device I/O Services

Introduction

The BIOS (Basic Input/Output System) is the lowest-level interface between other software (application programs and the operating system itself) and the hardware. The BIOS routines provide various device input/output services, as well as boot strap and print screen and other services. Some of the services that BIOS provides are not available through the operating system, such as the graphics routines.

All calls to the BIOS are made through software interrupts (that is, by means of assembly language "INT x" instructions). Each I/O device is provided with a software interrupt, which transfers execution to the routine.

Entry parameters to BIOS routines are normally passed in CPU registers. Similarly, exit parameters are generally returned from these routines to the caller in CPU registers. To insure BIOS compatibility with other machines, the register usage and conventions are, for the most part, identical.

The following pages describe the entry and exit requirements for each BIOS routine. To execute a BIOS call, load the registers as indicated under the "Entry Conditions." (Register AH will contain the function number in cases where a single interrupt can perform more than one operation.) Then issue the interrupt given for the call. The example, the following can be used to read a character from the keyboard:

```
MOV AH,0
INT 16H
```

Upon return, AL contains the ASCII character and AH the keyboard scan code.

Note: All registers except those used to return parameters to the caller are saved and restored by the BIOS routines.

Below is a quick reference list of software interrupts for all device I/O and system status services.

Service	Software Interrupts
Keyboard	16 hex (22 dec)
Video Display	10 hex (16 dec)
Serial Communications	14 hex (20 dec)
Line Printer	17 hex (23 dec)
System Clock	1A hex (26 dec)
Floppy Disk	13 hex (19 dec)
Equipment	11 hex (17 dec)
Memory Size	12 hex (18 dec)

Keyboard

16 hex (22 dec)

Function Summary:

AH = 0: Read Keyboard (destructive with wait)
AH = 1: Scan Keyboard (nondestructive, no wait)
AH = 2: Get Current Shift Status

Function Descriptions:

Read Keyboard

Read the next character typed at the keyboard. Return the ASCII value of the character and the keyboard scan code, removing the entry from the keyboard buffer (destructive read).

Entry Conditions:

AH = 0

Exit Conditions:

AL = *ASCII value of character*
AH = *keyboard scan code*

Scan Keyboard

Set up the zero flag (Z flag) to indicate whether a character is available to be read from the keyboard or not. If a character is available, return the ASCII value of the character and the keyboard scan code. The entry remains in the keyboard buffer (nondestructive read).

Entry Conditions:

AH = 1

Exit Conditions:

Z = no character is available
NZ = a character is available, in which case:
AL = *ASCII value of character*
AH = *keyboard scan code*

Get Shift Status

Return the current shift status.

Entry Conditions:

AH = 2

Exit Conditions:

AL = *current shift status (bit settings: set = true, reset = false)*
bit 0 = RIGHT SHIFT key depressed
bit 1 = LEFT SHIFT key depressed
bit 2 = CTRL (control) key depressed
bit 3 = ALT (alternate mode) key depressed
bit 4 = SCROLL state active
bit 5 = NUMBER lock engaged
bit 6 = CAPS lock engaged
bit 7 = INSERT state active

Video Display

These routines provide an interface to the video display, which is the output half of the console (CON) device. MS-DOS considers the video display to be the default standard output (STDOUT) device.

Software Interrupts:

10 hex (16 dec)

Function Summary:

Control Routines:
- AH = 0: Set CRT Mode
- AH = 1: Set Cursor Type
- AH = 2: Set Cursor Position
- AH = 3: Get Cursor Position
- AH = 4: Read Light Pen Position
- AH = 5: Select Active Page
- AH = 6: Scroll Up
- AH = 7: Scroll Down

Text Routines:
- AH = 8: Read Attribute/Character
- AH = 9: Write Attribute/Character
- AH = 10: Write Character Only'

Graphics Routines:
- AH = 11: Set Color Palette
- AH = 12: Write Dot
- AH = 13: Read Dot

Other Routines:
- AH = 14: Write TTY*
- AH = 15: Get CRT Mode
- AH = 16: Set Palette Registers

*Screen width is determined by the mode previously set. Some "control" characters (ASCII 00H-1FH) perform the usual special terminal function. These include (but are not limited to) BEL (07H), BS (08H), LF (0AH), and CR (0DH).

Function Descriptions:

Set CRT Mode

Entry Conditions:

AH = 0
AL = *mode value, as follows:*

Alpha Modes
- AL = 0: 40x25 black and white
- AL = 1: 40x25 color
- AL = 2: 80x25 black and white
- AL = 3: 80x25 color

Graphics Modes
- AL = 4: 320x200 color graphics
- AL = 5: 320x200 black and white graphics with 4 shades
- AL = 6: 640x200 black and white graphics with 2 shades
- AL = 7: Reserved

Additional Modes
- AL = 8: 160x200 color graphics with 16 colors
- AL = 9: 320x200 color graphics with 16 colors
- AL = A: 640x200 color graphics with 4 colors

Note: If the high order bit of the AL register is 1 then the video buffer is not cleared.

Set Cursor Type

Set the cursor type and attribute.

Entry Conditions:

AH = 1
CH = *bit values:*
bits 5-6:
bits 4-0 = start line for cursor within character cell
CL = *bit values:*
bits 4-0 = end line for cursor within character cell

Set Cursor Position

Write (set) cursor position.

Entry Conditions:

AH = 2
BH = *page number (must be 0 for graphics modes)*
DH = *row (0 = top row)*
DL = *column (0 = leftmost column)*

Get Cursor Position

Read (get) cursor position.

Entry Conditions:

AH = 3
BH = *page number (must be 0 for graphics modes)*

Exit Conditions:

DH = *row of current cursor position (0 = top row)*
DL = *column of current cursor position (0 = leftmost column)*
CH = *cursor type currently set [1]:*
bits 5-6:

Setting bits 5 and 6 to 00, produces a visible, blinking cursor. Any other values result in an invisible cursor.

bits 4-0 = *start line for cursor within character cell*

CL = *bit values:*
bits 4-0 = *end line for cursor within character cell*

See Set Cursor Type (AH = 1) above.

Read Light Pen Position

Reads light pen position.

Entry Conditions:

AH = 4

Exit Conditions:

AH = 0: light pen switch not activated
AH = 1: light pen values in registers
DH = *row of current light pen position*
DL = *column of current light pen position*
CH = *raster line (0-199)*
BX = *pixel column (0-319 or 0-639)*

Select Active Page

Select active display page (valid in alpha mode only).

Entry Conditions:

AH = 5
AL = 0 through 7: *new page value for modes 0, 1*
AL = 0 through 3: *new page value for modes 2, 3*
AL = 80H: *read CRT/CPU page registers*
AL = 81H: *set CPU page register to value in BL*
AL = 82H: *set CRT page register to value in BH*
AL = 83H: *set both CRT and CPU page registers in CL and BH*

Exit Conditions:

bit 7 of AL = 1: BH = *contents of CRT page register*
BL = *contents of CPU page register*

Scroll Up

Scroll active page up.

Entry Conditions:

AH = 6
AL = *number of lines to scroll (0 means blank entire window)*
CH = *row of upper left corner of scroll window*
CL = *column of upper left corner of scroll window*
DH = *row of lower right corner of scroll window*
DL = *column of lower right corner of scroll window*
BH = *attribute (alpha modes) or color (graphics modes) to be used on blank line*

Attributes:

Color modes:

foreground color:

bit 0 = blue
bit 1 = green
bit 2 = red

bit 3 = intensity
All bits off = black

background color:

bit 4 = blue
bit 5 = green
bit 6 = red

bit 7 = blink
All bits off = white

Scroll Down

Scroll active page down.

Entry Conditions:

AH = 7
AL = *number of lines to scroll (0 means blank entire window)*
CH = *row of upper left corner of scroll window*
CL = *column of upper left corner of scroll window*
DH = *row of lower right corner of scroll window*
DL = *column of lower right corner of scroll window*
BH = *attribute (alpha modes) or color (graphics modes) to be used on blank line. See Scroll Up (AH = 6) for attribute values and Set Color Palette (AH = 11) for color values.*

Read Attribute or Color/Character

Read a character and its attribute or color at the current cursor position.

Entry Conditions:

AH = 8
BH = *display page number (not used in graphics modes)*

Exit Conditions:

AL = *character read*
AH = *attribute of character (alpha modes only)*

Write Attribute or Color/Character

Write a character and its attribute or color at the current cursor position.

Entry Conditions:

AH = 9
BH = *display page number (not used in graphics modes)*
CX = *number of characters to write*
AL = *character to write*
BL = *attribute of character (for alpha modes) or color of character (for graphics modes; if bit 7 of BL is set, the color of the character is XOR'ed with the color value). See Scroll Up (AH = 6) for attribute values and Set Color Palette (AH = 11) for color values.*

Write Character Only

Write character only at current cursor position.

Entry Conditions:

AH = 10
BH = *display page number (valid for alpha modes only*
CX = *number of characters to write*
AL = *character to write*
BL = *color of character (graphics mode)*

Set Color Palette [3]

Select the color palette.

Entry Conditions:

AH = 11
BH = 0 Set background color (0-15) to color value in BL.
BL = *color value* (0 = black / 1 = blue / 2 = green / 3 = cyan / 4 = red / 5 = magenta / 6 = yellow / 7 = gray / 8 = dark gray / 9 = light blue / 10 = light green / 11 = light cyan / 12 = light red / 13 = light magenta / 14 = light yellow / 15 = white)

or
BH = 1 Set default palette ot the number (0 or 1) in BL.

In black and white modes:

BL = 0: 1 for white
BL = 1: 1 for black

In 4 color graphics modes:

BL = 0 (1 = green / 2 = red / 3 = yellow)
BL = 1 (1 = cyan / 2 = magenta / 3 = white)

In 16 color graphics modes:

(1 = blue / 2 = green / 3 = cyan / 4 = red / 5 = magenta / 6 = yellow / 7 = light gray / 8 = dark gray / 9 = light blue / 10 = light green / 11 = light cyan / 12 = light red / 13 = light magenta / 14 = yellow / 15 = white)

Note: For alpha modes palette entry 0 indicates the border color. For graphics mode palette entry 0 indicates the border and the background color.

Write Dot

Write a pixel (dot).

Entry Conditions:

AH = 12
DX = *row number*
CX = *column number*
AL = *color value (When bit 7 of AL is set, the resultant color value of the dot is the exclusive OR of the current dot color value and the value in AL.)*

Read Dot

Read a pixel (dot).

Entry Conditions:

AH = 13
DX = *row number*
CX = *column number*

Exit Conditions:

AL = *color value of dot read*

Write TTY

Write a character in teletype fashion. (Control characters are interpreted in the normal manner.)

Entry Conditions:

AH = 14
AL = *character to write*
BL = *foreground color (graphics mode)*

Get CRT Mode

Get the current video mode.

Entry Conditions:

AH = 15

Exit Conditions:

AL = *current video mode; see Set CRT Mode (AH = 0) above for values*
AH = *number f columns on screen*
BH = *current active display page*

Set Palette Registers

Sets palette registers.

Entry Conditions:

AH = 16
AL = 0 Set palette register
BL = *number of the palette register (0-15) to set*
BH = *color value to store*
AL = 1 Set border color register
BH = *color value to store*
AL = 2 Set palette color value to store and border registers
ES:DX :points to a 17 byte list.
bytes 0-15 = *values for palette registers 0-15*
byte 16 = *value for the border register*

Note: CS,SS,DS,ES,BX,CX,DX are preserved.

Serial Communications

These routines provide asynchronous byte stream I/O from and to the RS-232C serial communications port. This device is labeled the auxiliary (AUX) I/O device in the device list maintained by MS-DOS.

Software Interrupts:

14 hex (20 dec)

Function Summary:

AH = 0: Reset Comm Port
AH = 1: Transmit Character
AH = 2: Receive Character
AH = 3: Get Current Comm Status
DX = communication port number (0 or 1).

Function Descriptions:

Reset Comm Port

Reset (or initialize) the communication port according to the parameters in AL, DL, and DH.

Entry Conditions:

AH = 0
AL = *RS-232C parameters, as follows:*

7 6 5	*4 3*	*2*	*1 0*
Baud Rate	*Parity*	*Stop Bits*	*Word Length*

000 = 110 baud
001 = 150 baud
010 = 300 baud
011 = 600 baud
100 = 1200 baud
101 = 2400 baud
110 = 4800 baud
111 = 9600 baud

x0 = none
01 = odd
11 = even

0 = 1 sb
1 = 2 sb

10 = 7 bits
11 = 8 bits

Exit Conditions:

AX = *RS-232 status; see Get Current Comm Status (AH = 3) following*

Transmit Character

Transmit (output) the character in AL (which is preserved).

Entry Conditions:

AH = 1
AL = *character to transmit*
DX = *port number* (0 or 1)

Exit Conditions:

AH = *RS-232 status; see Get Current Comm Status (AH = 3) below* (If bit 7 is set, the routine was unable to transmit the character because of a timeout error.)

Receive Character

Receive (input) a character in AL (wait for a character, if necessary). On exit, AH will contain the RS-232 status, except that only the error bits (1,2,3,4,7) may be set; the timeout bit (7), if set, indicates that data set ready was not received. Thus, AH is non-zero only when an error occurred.

Entry Conditions:

AH = 2
DX = *port number* (0 or 1)

Exit Conditions:

AL = *character received*
AH = *RS-232 status; see Get Current Comm Status (AH =3) below*

Get Current Comm Status

Read the communication status into AX.

Entry Conditions:

AH = 3
DX = *port number* (0 or 1)

Exit Conditions:

AH = *RS-232 status, as follows (set = true):*
bit 0 = data ready
bit 1 = overrun error
bit 2 = parity error
bit 3 = framing error
bit 4 = break detect
bit 5 = transmitter holding register empty
bit 6 = transmitter shift register empty
bit 7 = timeout occurred

AL = *modem status, as follows (set = true):*
bit 0 = delta clear to send
bit 1 = delta data set ready
bit 2 = trailing edge ring detector
bit 3 = delta receive line signal detect
bit 4 = clear to send
bit 5 = data set ready
bit 6 = ring indicator
bit 7 = receive line signal detect

Line Printer

These routines provide an interface to the parallel line printer. This device is labeled "PRN" in the device list maintained by the operating system.

Software Interrupts:

17 hex (23 dec)

Function Summary:

AH = 0: Print Character
AH = 1: Reset Printer Port
AH = 2: Get Current Printer Status

Function Descriptions:

Print Character

Print a character.

Entry Conditions:

AH = 0
AL = *character to print*
DX = *printer to be used (0-2)*

Exit Conditions:

0AH = *printer status; see Get Current Printer Status (AH = 2) below*
(If bit 0 is set, the character could not be printed because of a timeout error.)

Reset Printer Port

Reset (or initialize) the printer port.

Entry Conditions:

AH = 1
DX = *printer to be used (0-2)*

Exit Conditions:

AH = *printer status/ see Get Current Printer Status (AH = 2) below*

Get Current Printer Status

Read the printer status into AH.

Entry Conditions:

AH = 2

Exit Conditions:

DX = *printer to be used (0-2)*
AH = *printer status, as follows (set = true):*
bit 0 = timeout occurred
bit 1 = [unused]
bit 2 = [unused]
bit 3 = I/O error
bit 4 = selected
bit 5 = out of paper
bit 6 = acknowledge
bit 7 = not busy

System Clock

These routines provide methods of reading and setting the clock maintained by the system. This device is labeled CLOCK in the device list of the operating system. An interface for setting the multiplexer for audio source is also provided.

Software Interrupts:

1A hex (26 dec)

Function Summary:

AH = 0: Get Time Of Day
AH = 1: Set Time Of Day
AH = 80H: Set Up Sound Multiplexer

Function Descriptions:

Get Time Of Day

Get (read) the time of day in binary format.

Entry Conditions:

AH = 0

Exit Conditions:

CX = *high (most significant) portion of clock count*
DX = *low (least significant) portion of clock count*
AL = 0 of the clock was read or written (via AH = 0,1) within the current 24-hour period; otherwise, AL > 0

Set Time Of Day

Set (write) the time of day using binary format.

Entry Conditions:

AH = 1
CX = *high (most significant) portion of clock count*
DX = *low (least significant) portion of clock count*

Sound Multiplexer

Sets the multiplexer for audio source.

Entry Conditions:

AH = 80
AL = *source of sound*
- 00 = 8253 channel 2
- 02 = audio in
- 03 = complex sound generator chip

Disk I/O Support for the Floppy Only System Configuration

Software Interrupt:

13 hex (19 dec)

Function Summary:

AH = 0: Reset Floppy Disk
AH = 1: Return Status of Last Floppy Disk Operation
AH = 2: Read Sector(s) from Floppy Disk
AH = 3: Write Sector(s) to Floppy Disk
AH = 4: Verify Sector(s) on Floppy Disk
AH = 5: Format Track on Floppy Disk

Function Descriptions:

Reset Floppy Disk

Reset the diskette system. Resets associated hardware and recalibrates all diskette drives.

Entry Conditions:

AH = 0

Exit Conditions:

See "Exits From All Calls" below.

Return Status of Last Floppy Disk Operation

Return the diskette status of the last operation in AL.

Entry Conditions:

AH = 1

Exit Conditions:

AL = *status of the last operation; see "Exits From All Calls" below for values*

Read Sector(s) from Floppy Disk

Read the desired sector(s) from disk into RAM.

Entry Conditions:

AH = 2
DL = *drive number (0-1)*
DH = *head number (0-1)*
CH = *track number (0-39)*
CL = *sector number (1 to 9)*
AL = *sector count (1 to 9)*
ES:BX = *pointer to disk buffer*

Exit Conditions:

See "Exits From All Calls" below.
AL = *number of sectors read*

Write Sector(s) to Floppy Disk

Write the desired sector(s) from RAM to disk.

Entry Conditions:

AH = 3
DL = *drive number (0-1)*
DH = *head number (0-1)*
CH = *track number (0-39)*
CL = *sector number (1 to 9)*
AL = *sector count (1 to 9)*
ES:BX = *pointer to disk buffer*

Exit Conditions:

See "Exits From All Calls" below.
AL = *number of sectors written*

Verify Sector(s) on Floppy Disk

Verify the desired sector(s).

Entry Conditions:

AH = 4
DL = *drive number (0-1)*
DH = *head number (0-1)*
CH = *track number (0-39)*
CH = *sector number (1 to 9)*
AL = *sector count (1 to 9)*

Exit Conditions:

See "Exits From All Calls" below.
AL = number of sectors verified

Format on Floppy Disk

Format the desired track.

Entry Conditions:

AH = 5
DL = *drive number (0-1)*
DH = *head number (0-1)*
CH = *track number (0-39)*
CL = *sector number (1-9)*
ES:BX = *pointer to a group of address fields for each track.* Each address field is made up of 4 bytes. These are C, H, R, and N, where:
C = track number
H = head number
R = sector number
N = the number of bytes per sector (00 = 128, 01 = 256, 02 = 512, 03 = 1024)
There is one entry for every sector on a given track.

Exit Conditions:

See "Exits From All Calls" below.

Exits From All Calls:

AH = *Status of operation, where set = true:*

Error Code	Condition
01H	Illegal Function
02H	Address Mark Not Found
03H	Write Protect Error
04H	Sector Not Found
08H	DMA Overrun
09H	Attempt To DMA Across A 64K Boundary
10H	Bad CRC on Disk Read
20H	Controller Failure
40H	Seek Failure
80H	Device Timeout, Device Failed To Respond

[NC] = operation successful (AH = 0)
[C] = operation failed (AH = error status)

Equipment

This service returns the "equipment flag" (hardware configuration of the computer system) in the AX register.

Software Interrupts:

11 hex (17 dec)

The "equipment flag" returned in the AX register has the meanings listed below for each bit:

Reset = the indicated equipment is not in the system
Set = the indicated equipment is in the system

bit Ø	diskette installed
bit 1	not used
bit 2,3	always = 11
bit 4,5	initial video mode Ø1 = 40x25 BW 1Ø = 80x25 BW
bit 6,7	number of diskette drives (only if bit Ø = 1) ØØ = 1 Ø1 = 2
bit 8	Ø = dma present 1 = no dma on system
bit 9, 10, 11	number of RS 232 cards
bit 12	game I/O attached
bit 13	not used
bit 14, 15	number of printers

Memory Size

This service returns the total number of kilobytes of RAM in the computer system (contiguous starting from address Ø) in the AX register.

Software Interrupts:

12 hex (18 dec)

Appendices

EXTENDED SCREEN AND KEYBOARD CONTROL

This appendix describes how you can change graphics functions, move the cursor, and reassign the meaning of any key on the keyboard by issuing special character sequences from within your program. These sequences are valid only when issued through MS-DOS function calls 1, 2, 6, and 9.

Before these special functions can be used, the extended screen and keyboard control device driver must be installed. To do this, place the following command in your CONFIG.SYS file (see Appendix C in the *MS-DOS Commands Reference Manual* for information on the configuration file):

DEVICE = ANSI.SYS

In the control sequences described below, the following apply:

- The symbol " * " represents a decimal number that you provide, specified with ASCII characters.
- The default value is used when no explicit value or a value of zero is specified.
- ESC represents the 1-byte code for ESC (1BH). For example, you could create ESC[5;9H under DEBUG as follows:

  E100 1B "[5;9H"

- Any ESC sequences not recognized by this driver will be passed on to the screen intact.

Cursor Control

Cursor Position (CUP)

ESC [* ; *H

Moves the cursor to the position specified by the parameters. The first parameter specifies the line number and the second parameter specifies the column number. The default value for * is 1. If no parameter is given, the cursor is moved to the home position (upper left corner).

Horizontal and Vertical Position (HVP)

ESC [* ; *f

Moves the cursor in the same way as Cursor Position (CUP), described above.

Cursor Up (CUU)

ESC [*A

Moves the cursor up one or more lines without changing columns. The value of * determines the number of lines moved. The default value for * is 1. This sequence is ignored if the cursor is already on the top line.

Cursor Down (CUD)

ESC [*B

Moves the cursor down one or more lines without changing columns. The value of * determines the number of lines moved. The default value for * is 1. This sequence is ignored if the cursor is already on the bottom line.

Cursor Forward (CUF)

ESC [*C

Moves the cursor forward one or more columns without changing lines. The value of * determines the number of columns moved. The default value for * is 1. This sequence is ignored if the cursor is already in the rightmost column.

Cursor Backward (CUB)

ESC [*D

Moves the cursor back one or more columns without changing lines. The value of * determines the number of columns moved. The default value for * is 1. This sequence is ignored if the cursor is already in the leftmost column.

Device Status Report (DSR)

ESC [6n

The console driver outputs a Cursor Position Report (CPR) sequence on receipt of DSR (see below).

Cursor Position Report (CPR)

ESC [* ; *R

Reports current cursor position through the standard input device. The first parameter specifies the current line and the second parameter specifies the current column.

Save Cursor Position (SCP)

ESC [s

Saves the current cursor position. You can restore this position with the Restore Cursor Position (RCP) sequence (see below).

Restore Cursor Position (RCP)

ESC [u

Restores the cursor position to the value it had when the console driver received the SCP sequence.

Erasing

Erase Display (ED)

ESC [2J

Erases the screen and sends the cursor to the home position (upper left corner).

Erase Line (EL)

ESC [K

Erases from the cursor to the end of the line (including the cursor position).

Erase To End Of Screen

ESC [0J

Erases the screen from the cursor position to the end of the screen.

Erase From Top Of Screen

ESC [1J

Erases screen from the top of the screen to the cursor position.

Modes of Operation

Set Graphics Rendition (SGR)

ESC [* ;...; *m

Sets the character attribute(s) specified by the parameter(s) described below. The attributes remain in effect until the next occurrence of an SGR escape sequence.

Parameter	Meaning
0	All attributes off (normal white on black)
1	Highlight on (high intensity)
5	Blink on
7	Reverse video on
8	Concealed on (invisible)
30	Black foreground
31	Red foreground
32	Green foreground
33	Yellow foreground
34	Blue foreground
35	Magenta foreground
36	Cyan foreground
37	White foreground
40	Black background
41	Red background
42	Green background
43	Yellow background
44	Blue background
45	Magenta background
46	Cyan background
47	White background

Set Mode (SM)

ESC [= *h
or ESC [= h
or ESC = 0h
or ESC [?7h

Sets the screen width or type specified by the parameter.

Parameter	Meaning
0	40 x 25 black and white
1	40 x 25 color
2	80 x 25 black and white
3	80 x 25 color
4	320 x 200 color
5	320 x 200 black and white
6	640 x 200 black and white
7	Wrap around at end of line (new line starts when old line filled)
8	160 x 200 16 color
9	320 x 200 16 color
10	640 x 200 4 color

Reset Mode (RM)

ESC [= *l
or ESC [= l
or ESC [=0l
or ESC [?7l

Parameters are the same as for Set Mode (SM) except that parameter 7 resets the wrap-around mode (characters past end-of-line are thrown away).

Keyboard Key Reassignment

ESC [* ; * ;... * p
or ESC ["string" * p
or ESC [* ;"string"; * ; * ;"string"; * p
or any other combination of strings and decimal numbers

Changes the meaning of a key on the keyboard. The first ASCII code in the control sequence defines which code is being mapped. The remaining numbers define the sequence of ASCII codes generated when this key is intercepted. However, if the first code in the sequence is zero (NUL), then the first and second codes make up an extended ASCII redefinition. (See Appendix B for a list of ASCII and extended ASCII codes.)

Examples:

1. Reassign the Q and q key to the A and a key (and vice versa):

   ESC [65;81p A becomes Q
   ESC [97;113p a becomes q
   ESC[81;65p Q becomes A
   ESC [113;97p q becomes a

2. Reassign the F10 key to a DIR command followed by a carriage return:

   ESC [0;68;"dir";13p

The 0;68 is the extended ASCII code for the F10 key. 13 decimal is a carriage return.

KEYBOARD ASCII AND SCAN CODES

The table in this appendix lists the keys on the Tandy 1000 keyboard in scan code order, along with the ASCII codes they generate. For each key, the following entries are given:

Scan Code — A value in the range 01H-5AH which uniquely identifies the physical key on the keyboard that is pressed.

Keyboard Legend — The physical marking(s) on the key. If there is more than one marking, the upper one is listed first.

ASCII Code — The ASCII codes associated with the key. The four modes are:

Normal — The normal ASCII value (returned when only the indicated key is depressed).

SHIFT — The shifted ASCII value (returned when SHIFT is also depressed).
CTRL — the control ASCII value (returned when CTRL is also depressed).

ALT — The alternate ASCII value (returned when ALT is also depressed).

Remarks — Any remarks or special functions.

The following special symbols appear in the table:

x	Values preceded by "x" are extended ASCII codes (codes preceded by an ASCII NUL, 00).
—	No ASCII code is generated.
*	No ASCII code is generated, but the special function described in the Remarks column is performed. If no comment is included, the key does not generate a code and no function is performed.

Note: All numeric values in the table are expressed in hexadecimal.

QWERTY (USA) — Model 1000

Scan Code	Keyboard Legend		ASCII Codes Normal	SHIFT	CTRL	ALT	As of Oct. 22 1984 Remarks
01	ESC		1B	1B	1B	x8B	
02	1	!	31	21	xE1	x78	
03	2	@	32	40	x03	x79	
04	3	#	33	23	xE3	x7A	
05	4	$	34	24	xE4	x7B	
06	5	%	35	25	xE5	x7C	
07	6	^	36	5E	1E	x7D	
08	7	&	37	26	xE7	x7E	
09	8	*	38	2A	xE8	x7F	
0A	9	(	39	28	xE9	x80	
0B	0	)	30	29	xE0	x81	
0C	-	_	2D	5F	1F	x82	
0D	=	+	3D	2B	xF5	x83	
0E	BACK SPACE		08	08	7F	x8C	
0F	TAB		09	x0F	x8D	x8E	
10	Q		71	51	11	x10	
11	W		77	57	17	x11	
12	E		65	45	05	x12	
13	R		72	52	12	x13	
14	T		74	54	14	x14	
15	Y		79	59	19	x15	
16	U		75	55	15	x16	
17	I		69	49	09	x17	
18	O		6F	4F	0F	x18	
19	P		70	50	10	x19	
1A	[	{	5B	7B	1B	xEB	
1B	]	}	5D	7D	1D	xF0	
1C	ENTER		0D	0D	0A	x8F	Main Keyboard
1D	CTRL		*	*	*	*	Control Mode
1E	A		61	41	01	x1E	
1F	S		73	53	13	x1F	
20	D		64	44	04	x20	
21	F		66	46	06	x21	
22	G		67	47	07	x22	
23	H		68	48	08	x23	
24	J		6A	4A	0A	x24	
25	K		6B	4B	0B	x25	
26	L		6C	4C	0C	x26	
27	;	:	3B	3A	xF6	xF8	
28	'	"	27	22	xF7	xF1	
29	UP ARROW		x48	x85	x90	x91	
2A	SHIFT		*	*	*	*	Left SHIFT
2B	LEFT ARROW		x4B	x87	x73	x92	
2C	Z		7A	5A	1A	x2C	
2D	X		78	58	18	x2D	
2E	C		63	43	03	x2E	
2F	V		76	56	16	x2F	

Scan Code	Keyboard Legend		ASCII Codes Normal	SHIFT	CTRL	ALT	As of Oct. 22 1984 Remarks
30	B		62	42	02	x30	
31	N		6E	4E	0E	x31	
32	M		6D	4D	0D	x32	
33	,	<	2C	3C	xF9	x89	
34	.	>	2E	3E	xFA	x8A	
35	/	?	2F	3F	xFB	xF2	
36	SHIFT		*	*	*	*	Right SHIFT
37	PRINT		10	*	x72	x46	SCR Print Toggle
38	ALT		*	*	*	*	Alternate Mode
39	space bar		20	20	20	20	
3A	CAPS		*	*	*	*	Caps lock
3B	F1		x3B	x54	x5E	x68	
3C	F2		x3C	x55	x5F	x69	
3D	F3		x3D	x56	x60	x6A	
3E	F4		x3E	x57	x61	x6B	
3F	F5		x3F	x58	x62	x6C	
40	F6		x40	x59	x63	x6D	
41	F7		x41	x5A	x64	x6E	
42	F8		x42	x5B	x65	x6F	
43	F9		x43	x5C	x66	x70	
44	F10		x44	x5D	x67	x71	
45	NUM LOCK		*	*	*	*	number lock
46	HOLD		*	*	*	*	Freeze display
47	7	\	37	5C	x93	*	
48	8	~	38	7E	x94	*	
49	9	PG UP	39	x49	x84	*	
4A	DOWN ARROW		x50	x86	x96	x97	
4B	4	|	34	7C	x95	*	
4C	5		35	xF3	xFC	*	
4D	6		36	xF4	xFD	*	
4E	RIGHT ARROW		x4D	x88	x74	xEA	
4F	1	END	31	x4F	x75	*	
50	2	‘	32	60	x9A	*	
51	3	PG DN	33	x51	x76	*	
52	0		30	x9B	x9C	*	
53	-	DELETE	2D	x53	x9D	x9E	
54	BREAK		x00	x00	*	*	scroll lock bit toggle control brk routine (INT 1BH)
55	+	INSERT	2B	x52	x9F	xA0	
56			2E	xA1	xA4	xA5	Numeric keypad
57	ENTER		0D	0D	0A	x8F	Numeric keypad
58	HOME		x47	x4A	x77	xA6	
59	F11		x98	xA2	xAC	xB6	
5A	F12		x99	xA3	xAD	xB7	

* Indicates special functions performed

— means this key combination is suppressed in the keyboard driver

X values preceded by "X" are extended ASCII codes (codes preceded by an ASCII NUL)

† The (ALT) key provides a way to generate the ASCII codes of decimal numbers between 1 and 255. Hold down the (ALT) key while you type *on the numeric keypad* any decimal number between 1 and 255. When you release ALT, the ASCII code of the number typed is generated and displayed.

Note: When the NUM LOCK light is off, the Normal and SHIFT columns for these keys should be reversed.

MS-DOS Memory Map

HEXADECIMAL STARTING ADDRESS (SEGMENT:OFFSET)	DESCRIPTION
000:00	BIOS Interrupt Vectors
000:80	Available Interrupt vectors
0040:00[1]	ROM BIOS Data Area
0050:00	MSDOS and BASIC Data Area
0070:00	I/O.SYS Drivers
0190:00[2]	MS-DOS
05B0:00[2]	Available to user
X800:00[3]	Video RAM in 32K video modes
XC00:00[3]	Video RAM in 16K video modes
B800:00[4]	Video RAM Window (32K)
F000:00	Reserved for system ROM
FC00:00	System BIOS ROM

Notes:

1. Detailed description in following pages.
2. Approximate address; subject to change.
3. "X" is defined as follows:

Memory Size	X Value
128K	1
256K	3
384K	5
512K	7
640K	9
768K	B

4. Video memory accessed through the B800:0 window for all video modes.

ROM BIOS Data Area

The following table gives the starting offset, and length of each BIOS device driver. This area is located at segment 40:00.

Comm card addresses	0000	8 (1 word per card)
Printer addresses	0008	8 (1 word per printer)
Devices installed	0010	2 (16 bits)
Not used	0012	1
Memory size	0013	2 (1 word)
I/O channel RAM size	0015	2 (1 word)
KBD data area	0017	39
Disk data area	003E	11
Video data area	0049	30
Not used	0067	5
Clock data area	006C	5
KBD Break & Reset flags	0071	3
Not used	0074	4
Printer Timeout counter	0078	4 (1 byte per printer)
Comm Timeout counter	007C	4 (1 byte per card)
KBD extra data area	0080	4 (2 words)

The structure and usage of the Video driver RAM data area is as follows:

HEX Offset From Segment 0040:000	**Length and Intended Use**
49H	1 byte - current CRT mode (0-7)
4AH	1 word - screen column width
4CH	1 word - byte length of screen
4EH	1 word - address/offset of beginning of current display page
50H	8 words - row/col coordinates of the cursor for each of up to 8 display pages
60H	1 word - current cursor type (See "set cursor type" for correct encoding)
62H	1 byte - current display page
63H	1 word - base address + 4 of the CRT controller card
65H	1 byte - copy of value written to the Mode Select Register
66H	1 byte - current color palette setting

The equipment check BIOS call (INT 11H) and memory size BIOS call (INT 12H) return information from the following data areas:

HEX Offset From Segment 0040:000	Length and Intended Use
10H	Devices installed word
13H	Memory installed word

The structure and usage of the floppy disk driver RAM data area is as follows:

HEX Offset From Segment 0040:0000	Length and Intended Use
3EH	1 byte - drive recalibration status - bit 3-0, if 0 then drive 3-0 needs recal before next seek bit 7 indicates interrupt occurrence
3FH	1 byte - motor status - bit 3-0 drive 3-0 motor is on/off, bit 7 - current operation is write, requires delay
40H	1 byte - motor turn off time out counter (see Timer ISR)
41H	1 byte - disk status - codes defined below
42H	7 bytes - 7 bytes of status returned by the controller during result phase of operation

Value	Error Condition
01H	Illegal Function
02H	Address Mark Not Found
03H	Write Protect Error
04H	Sector Not Found
08H	DMA Overrun
09H	Attempt to DMA Across a 64K Boundary
10H	Bad CRC on Disk Read
20H	Controller Failure
40H	Seek Failure
80H	Device Timeout, Device Failed to Respond

The structure and usage of the RS232 driver RAM data area is as follows:

HEX Offset From Segment 0040:00	**Length and Intended Use**
00H	4 words - Base address of each one of 4 possible comm cards
7CH	4 words - 1 word timeout count for each of 4 possible comm cards

The structure and usage of the Keyboard driver RAM data area is as follows:

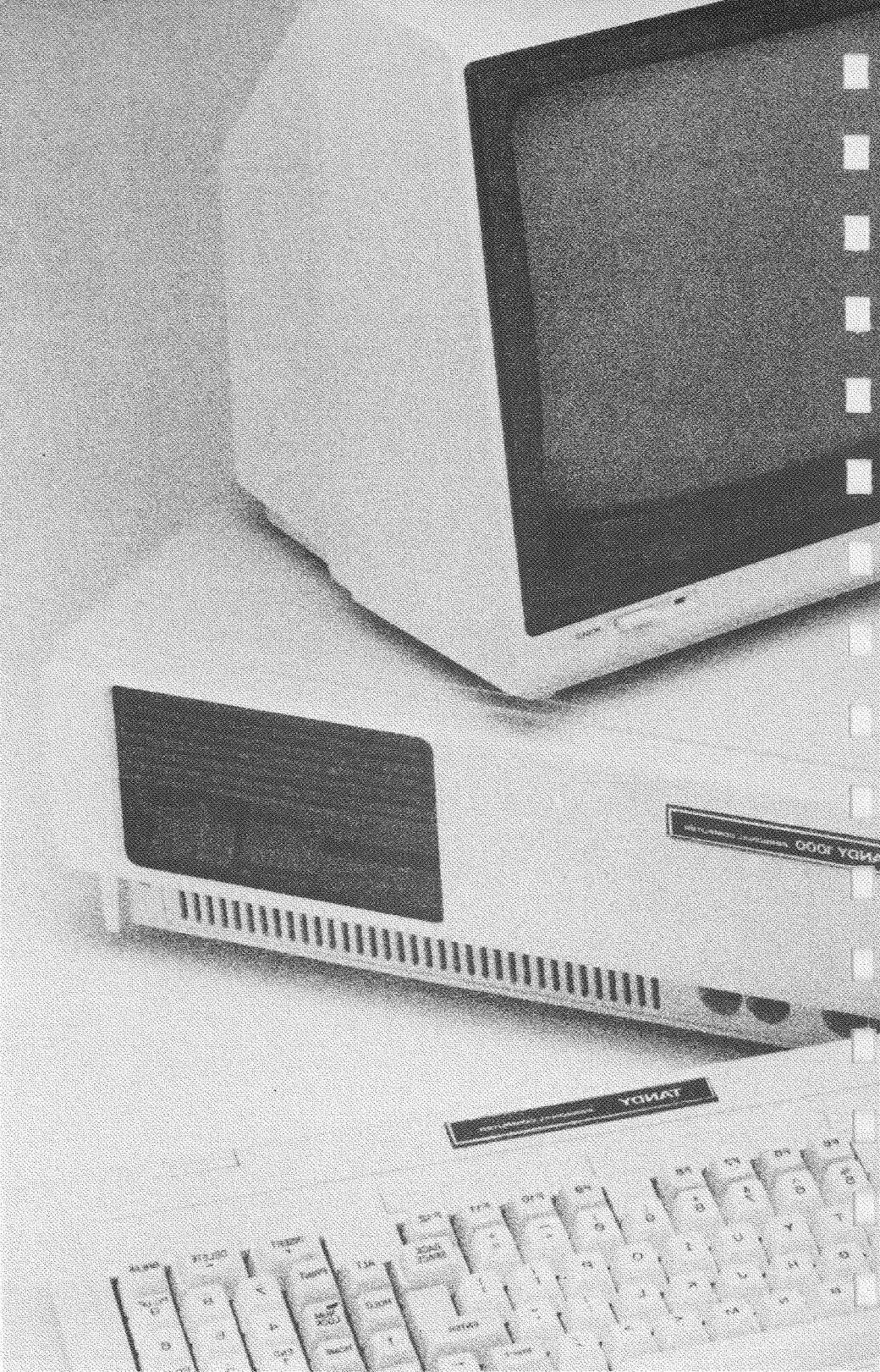
HEX Offset From Segment 0040:0010	**Length and Intended Use**
17	1 byte - Keyboard shift state flag returned by function 02
	bits 7 - INSERT state active, 6 - CAPS LOCK on/off, 5 - NUM LOCK on/off, 4 - SCROLL LOCK on/off, 3 - ALT key depressed, 2 - CTRL key depressed, 1 - Left SHIFT key depressed, 0 - Right SHIFT key depressed
18	1 byte - Secondary shift state flag
	bits INSERT key depressed, 6 - CAPS LOCK depressed, 5 - NUM LOCK depressed, 4 - SCROLL LOCK NUM LOCK depressed, 4 - SCROLL depressed, 4 - SCROLL LOCK depressed, 3 - Pause on/off, depressed, 3 - Pause on/off, 2,1,0 - not used

19	1 byte	- used to store ALT keypad entry
1A	1 word	- pointer to beginning of the keyboard buffer
1C	1 word	- pointer to end of the keyboard buffer
1E	16	- keyboard buffer (enough for words)
	15	- typeahead entries

The structure and usage of the clock service routine is as follows:

HEX Offset From Segment 0040:0000		**Length and Intended Use**
6CH	1 word	- Lease significant 16 bits of clock count
6EH	1 word	- Most significant 16 bits of clock count
70H	1 byte	- Twenty four hour rollover flag

DELETE
INSERT



INDEX

abort 32
absolute disk write 22-23
absolute disk read 20
active display page 198
active page select 195
alloc 123
allocate memory 123
allocation
  disk 165
  memory 123-25
alpha modes 196
  palette 201
ASCII codes, keyboard 225-28
ASCIIZ strings 26
attribute
  retrieving 115
  field 175
  in FCB 167
  of a file 11
  setting 115
auxiliary
  device 35-36
  input 35
  output 36
auxinput 35
auxoutput 36

background color 199
basic input/output system 191
baud rate, com port 204
BIOS services
  call 191
  compatibility 191
  equipment 192, 213
  floppy disk 192, 210-13
  line printer 192, 207-08
  memory size 192
  parameter block (BPB) 178-81
  quick reference 192
  routines 191
  serial communications 192, 204-06

BIOS services (cont.)
  services, keyboard 192, 193-94
  system clock 192, 208-09
  video display 192, 195-203
boot sector 183-84
bootstrap 191
buffer
  input 43
  type-ahead 46
buffered keyboard input 43-44
build BPB 178-183

change attributes 115
change the current directory 105
character
  displaying 34
  inputting 33
  receive 205
  transmit 205
  printing 37
  attribute 195
chdir 105
check keyboard status 45
child process 127-130
chmod 115
clock device 187
clock, system 208-09
close 110
close a file handle 110
close file 51
color
  graphics 196
  palette 195-201
  write attribute 200
column-light pen position 195-98
column, cursor position 197
com port 204-06
  reset 204
com status, current 205
command code field 178
command processor 161
communications, serial 204
con device 195
coninput 40

coninputflush 46
coninputnoecho 41
coninputstatus 45
conio 38
constringinput 43-44
constringoutput 42
control-c
  check 96
  exit address 12
  routine 12
create a file 106-07
create file 62-63
create subdirectory 103
crt mode 195-202
crt/cpu page registers 198
curdsk 66
current block (FCB) 151
current com status 205-06
current disk 66
current printer status 208
current record number (FCB) 152
current shift status, keyboard 193
currentdir 122
cursor 197
cursor control (esc) 218-19

date
  returning 85-136
  setting 86-136
date of last write
  in disk directory 168
  in FCB 152
dealloc 124
default, palette 210
delete a directory entry 112
delete file 56-57
device
  block 173-176
  channels 116-18
  character 173
  clock 187
  creating 176
  dumb and smart 181
  header 174-75

device (cont.)
installing 177
nul 175
reading from 111
types 173
writing to 112
request header 177
I/O control 116-19
direct console
I/O 38-39
input 40
directory 10-11
changing the current 105
creating a sub 103
disk 166-68
format 166-68
returning the current 105
directory entry
moving 134
removing 104
search for first 52
search for next 54
dirsearchfirst 52-53
disk
allocation 165
BIOS format 212
directory 166-68
free space on 98
disk I/O support 210-12
disk transfer address
returning 93
setting 67
display character 34
display string 42
dot
read 202
read/write 195
drive
as block device 173
current selected 66
letter 173
number (in FCB) 151
selecting 48
drivers, device 173-87

dup 120
dup2 121
duplicate a file handle 120

entry conditions, BIOS 191
environment 127-28
equipment flag, BIOS 213
erasing (esc) 220
error codes — for function calls 25-26
esc 217
exec 126-28
exit code 95-130
exit conditions, BIOS 191
exits from BIOS calls 212
extended file control block 153
extended screen and keyboard control 17-23

FAT 168-70
fatal error abort address 17-19
FCB 151-53
file
  attribute 11
  closing 51-110
  creating 62-106
  deleting 56
  moving 134
  opening 49-108
  renaming 64-134
  size 73
  size in FCB 152
file allocation table (FAT) 168-70
file control block (FCB) 151-53
  extended 153
file handle 26
  duplicating 120-121
file size 73-74
  in disk directory 168
filename
  in disk directory 166
  in FCB 151
  parsing 82-84
  separators 82
  terminators 82
filetimes 136

find matching file 131-32
find next matching file 133
findfirst 131-32
findnext 133
flag
  carry 25
  verify 91-134
  equipment 213
floppy disk
  format (BIOS) 212
  I/O 192
  reset 210
  write sectors 211
flush 186
flush buffer, read keyboard 46
fname 82-84
force a duplicate of a file handle 121
format floppy disk (BIOS) 212
free allocated memory 124
function call parameters 181
function calls 25-147
  alphabetical list of 30-31
  calling 27
  categories 25
  CPM/M compatible 27
  descriptions of 32-136
  error codes 25-26
  numeric list of 28-321
  register treatment 28
function request 14

get
  crt mode 195
  cursor position 195-197
get date 85
get disk free space 98
get disk transfer address 93
get interrupt vector 97
get or set a file's date and time 136
get time 88
get version number 94-95
getdta 93
getfreespace 98
getvector 97

getverifyflag 134-35
graphic modes 196-201

I/O control for devices 116-18
I/O device 191
init 182
initialization 161
input
  auxiliary 35
  direct console 38
  keyboard 33
input/output services 191
interface, hardware and software 191
international 99-102
interrupt
  list of 12
  vector, retrieving 97
  vector, setting 77
interrupts 12-24
ioctl 116-18

keep process 95
keyboard
  ASCII and scan codes 225-28
  BIOS services 193-94
  I/O 192
  input 33
  read 41
  status 45

light pen
  position 198
  read position 195
line printer
  I/O 192
  interface 207-08
load and execute a program 126-28
logical record size (FCB) 152
lseek 114

macro 12
  definitions, list of 137-147
media check 178-182
media descriptor byte 184

memory
allocating 123
deallocating 124
modifying allocated block of 125
size 192
size (BIOS) 213
map 229-233
mkdir 103
mode, crt 195-202
modify allocated memory blocks 125
move
a directory entry 134
a file pointer 114
MS-DOS
command processor 161
control blocks 151
disk allocation 165
disk directory 166-68
memory map 229
initialization 161
work areas 151-57
multiplexer, sound 209

name field 176
nondestructive read no wait 186

open 108-09
open a file 108-09
open file 49-50
output-auxiliary 36
output
direct console 38
output-printer 37

page
active 198
registers crt/cpu 198
palette color 195
palette set color 200
parent process 127
parity, com port 204
parse filename 82-84
pen, light 195-198
pixel, column 198

port, com 104
position
light pen, cursor 195
cursor 197
print character 37, 207
printer
port reset 207
status 207-08
printeroutput 37
printing a character 37
program segment 153-56
program segment prefix 157
program terminate 13

quick reference 192
device I/O interrupts 192

random block write 80-81
random block read 78-79
random read 69-70
random write 71-72
raster line 198
rbread 78-79
read
attribute/character 205
dot 195-202
from file or device 111
keyboard 41-46
keyboard (BIOS) 193
light pen position 195-198
random 69
random block 78
sector, floppy disk 210-198
sequential 58
read or write 185
receive character 205
registers
function call, treatment of 27
page 198
palette 195
relative record number (FCB) 152
remove a directory entry 104
rename 134
rename file 64

request header 177
reset 207
  com port 204
reset disk 47
retrieve the return code of a child 130
return code 95-130
return country-dependent information 99-102
return current setting of verify 134-35
return status of last floppy disk operation 210
return text of current directory 122
rmdir 104
row
  cursor position 197
  light pen position 198

scan codes, keyboard 225-28
scan keyboard (BIOS) 193
screen and keyboard control, extended 217-23
screen width 195
scroll 195-198
search for first entry 52-53
search for next entry 54
searchnext 54-55
sectors
  read, floppy disk 210-11
  verify 211-12
  write to floppy disk 211
select
  active page 195
  disk 48
  color palette 200
seqread 58-59
sequential read 58-59
sequential write 60-61
seqwrite 60-61
serial communications 192-204
setblock 125
set
  crt mode 195-96
  cursor position 195-197
  cursor type 195-197
  date 86-87
  disk transfer address 67-68

set (cont.)
graphics rendition (esc) 221
interrupt vector 77
palette registers 195
relative record 75-76
time 89-90; 208-09
set/reset
modes 21
verify flag 91-92
setctrlctrapping 96
setdta 67-68
setrelrec 75-76
setvector 77
setverify 91-92
sgr 221
shift status, keyboard 193-94
software interrupts 191-92
sound multiplexer 209
starting cluster 168-170
status 186
status, printer 208
stdconinput 33
stdconoutput 34
stop bits, com port 204
string displaying 42
system 192
system calls 9-147
calling 9-12
extended example 143-47
returning from 9-12
XENIX compatible 10
system clock 208-09
system status service 192

terminate
a process 129
address 15
but stay resident 24
program 32
terminating a program 13, 32
time
in FCB 152
of day 208-09
of last write in disk directory 167

time (cont.)
  returning 88-136
  set 208-09
  setting 89-136
transmit character 205
TTY 195-202
type, cursor 195

unit code field 177
unlink 113

verify
  flag, setting and resetting 91
  returning 134
  sectors, floppy disk 211-12
version-returning 94
video display (BIOS) 192-195

wait 130
width, screen 195
write
  character 200
  cursor position 197
  dot 195-201
  random 71
  random block 80
  sectors to floppy disk 211
  sequential 60
  to a file or device 112
  TTY 195-202
write attribute/character 195

XENIX 10

RADIO SHACK

A Division of Tandy Corporation
U.S.A.: Fort Worth, Texas 76102
CANADA: Barrie, Ontario L4M 4W5

TANDY CORPORATION

AUSTRALIA
91 Kurrajong Road
Mount Druitt, N.S.W. 2770

BELGIUM
Pare Industriel De Naninne
5140 Naninne

U.K.
Bilston Road Wednesbury
West Midlands WS10 7JN

12/84 - TMG

Printed in U.S.A.