File I/O in Real-Address Mode

This article explains how to create, open, read,and write files in 16-bit Real-address mode. It is the full text of Chapter 12 from the previous edition of *Assembly Language for Intel-Based Computers*. Copyright 1999 Prentice-Hall Publishing, all rights reserved.

12.1 FILE MANIPULATION

12.1.1 Introduction

Having developed a good understanding of disk file organization, let's now examine the multitude of function calls relating to files. DOS uses the technique, borrowed from the UNIX operating system, of using handles to access files and devices. In most cases, there is no distinction between files and devices such as keyboards and video monitors. A *handle* is a 16-bit number used to identify an open file or device. There are five standard device handles recognized by DOS. Each of these supports redirection at the command prompt except the error output device:

0	Keyboard (standard input)
1	Console (standard output)
2	Error output
3	Auxiliary device (asynchronous)
4	Printer

These handles are predefined and do not have to be opened before being used. For example, one can write to the console using handle 1 without any advance preparation. Each function has a common characteristic: if it fails, the Carry flag is set and an error code is returned in AX. You can use this error code to display an appropriate message to the program's user.

***Basic File Functions*.** Let's start by looking at a list of the most commonly used file functions, defined by a *function number* placed in AH. All the following functions are available in high-level languages (see Table 1).

The next set of file manipulation routines allows powerful control of files, often beyond that allowed at the command prompt. For example, we can hide or unhide a file, change a normal file to read-only, or change the time and date stamp on a file. We can also search for all files matching a file specifier with a wildcard character such as *.ASM.

12.1.2 Get/Set File Attribute (43h)

Function 43h can be used to either retrieve or change the attribute of a file. We set a flag in AL to decide which action to perform. The following input registers are used:

AH	43h
AL	(0 = get attribute, 1 = set attribute)
CX	New attribute (if AL = 1)
DS:DX	Points to an ASCIIZ string with a file specification

The Carry flag is set if the function fails, and the error return codes are 1 (function code invalid), 2 (file not found), 3 (path not found), and 5 (access denied). If AL = 0 (get attribute function), the file attribute is returned in CX. The attribute may also indicate a volume label (08h) or a subdirectory (10h). The following instructions set a file's attributes to hidden and read-only:

```
.data
filename  db  "TEST.DOC",0
```

```
.code
mov   ah,43h
mov   al,1                    ; set file attribute
mov   cx,3                    ; hidden, read-only
mov   dx,offset filename
int   21h
jc    display_error
```

You may want to refer to the discussion of file attributes earlier in Section 11.1.4. Sample values are shown in the following table. In addition, the archive bit (5) may have been set:

Table 1. Basic File Functions.

Function	Description
1Ah	Set disk transfer address.
3Ch	Create file. Create a new file or set the length of an existing file to 0 bytes.
3Dh	Open file. Open an existing file for input, output, or input-output.
3Eh	Close file handle.
3Fh	Read from file or device. Read a predetermined number of bytes from a file into an input buffer.
40h	Write to file or device. Write a predetermined number of bytes from memory to a file.
41h	Delete file.
42h	Move file pointer. Position the file pointer before reading or writing to a file.
43h	Get/Set file attribute.
4Eh	Find first matching file.
4Fh	Find next matching file.
56h	Rename file.
57h	Get/set file date and time.

Attribute	Value
Normal file	00
Read-only file	01
Hidden file	02
Hidden, read-only file	03
System file	04
Hidden, system, read-only file	07

One reason this function is important is that it allows you to hide a file so it won't appear when the DIR, DEL, and COPY commands are used. You can also give a file a read-only attribute to prevent it from being changed. In fact, the only way to delete or update a read-only file at the DOS command prompt is to *first* change its attribute to normal.

12.1.3 Delete File (41h)

To delete a file, set DS:DX to the address of an ASCIIZ string containing a file specification. The specification can contain a drive and path name, but wildcard characters are not allowed. For example, the following code deletes SAMPLE.OBJ from drive B:

```
.data
filespec  db  "B:SAMPLE.OBJ",0

.code
mov   ah,41h                  ; delete file
mov   dx,offset filespec
int   21h
jc    display_error
```

If DOS fails and the Carry flag is set, the possible error codes are 2 (*file not found*), 3 (*path not found*), and 5 (*access denied because the file has a read-only attribute*). To delete a file that has a read-only attribute, you must first call Function 43h (*change file mode*) to change its attribute.

12.1.4 Rename File (56h)

Function 56h renames a file if you pass a pointer to the current name in DS:DX and a pointer to the new name in ES:DI. Both names must be ASCIIZ strings, without any wildcard characters. This function can also be used to move a file from one directory to another because you can specify a different path for each filename. Moving a file is different from copying it; the file no longer exists in its original place. If the Carry flag is set, the possible error codes are 2 (*file not found*), 3 (*path not found*), 5 (*access denied*), and 11h (*not same device*). Error 11h occurs when one refers to filenames on different disk drives. The following routine renames prog1.asm to prog2.asm:

```
.data
oldname  db  "prog1.asm",0
newname  db  "prog2.asm",0

.code
mov   ah,56h                  ; rename file
mov   dx,offset oldname
mov   di,offset newname
int   21h
jc    display_error
```

The following statements move prog1.asm from the current directory to the \asm\progs directory:

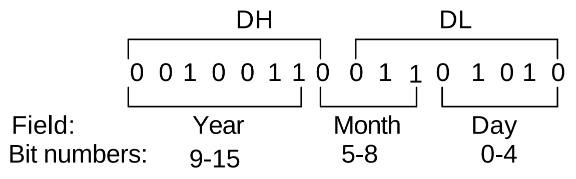
```
.data
oldname  db  "prog1.asm",0
newname  db  "\asm\progs\prog1.asm",0

.code
mov   ah,56h                  ; rename file
mov   dx,offset oldname
mov   di,offset newname
int   21h
jc    display_error
```

12.1.5 Get/Set File Date/Time (57h)

Function 57h can be used to read or modify the date and time stamps of a file. Both are automatically updated when a file is modified, but there may be occasions when you wish to set them to some other value.

The file must already be open before calling this function. If you wish to read the file's date and time, set AL to 0 and set BX to the file handle. To set the date and time, set AL to 1, BX to the file handle, CX to the time, and DX to the date. The time and date values are bit-mapped exactly as they are in the directory. Here, we show the date:



The seconds are stored in increments of 2. A time of 10:02:02, for example, would be mapped as

```
0101000001000001
```

The year value is assumed to be added to 1980, so the date April 16, 1992 (920416) would be stored as

```
0001100010010000
```

If you simply want to get a file's date and time, Function 4Eh (*find first matching file*) is easier to use because it does not require the file to be open.

12.1.6 Find First Matching File (4Eh)

To search for a file in a particular directory, call Function 4Eh (*find first matching file*). Pass a pointer to an ASCIIZ file specification in DS:DX and set CX to the attribute of the files you wish to find. The file specification can include wildcard characters (* and ?), making this function particularly well suited to searches for multiple files. For example, to look for all files with an extension of ASM in the C:\ASM\PROGS directory, we would use the following:

```
.data
filespec  db  "C:\ASM\PROGS\*.ASM",0

.code
mov   ah,4Eh                  ; find first matching file
mov   cx,0                    ; find normal files only
mov   dx,filespec
int   21h
jc    display_error
```

If a matching file is found, a 43-byte file description is created in memory at the current *disk transfer address* (DTA). The location defaults to offset 80h from the PSP, but we usually reset it to a location within the data segment, using Function 1Ah (*set disk transfer address*). The following is a description of the DTA when a matching file has been found:

Offset	File Information
0-20	Reserved by DOS
21	Attribute
22-23	Time stamp
24-25	Date stamp
26-29	Size (doubleword)
30-42	File name (null-terminated string)

This function provides a convenient way to get the time and date stamp of a file without having to open it. If the search fails, the Carry flag is set and AX equals either 2 (*invalid path*) or 18 (*no more files*). The latter means that no matching files were found.

12.1.7 Find Next Matching File (4Fh)

Once Function 4Eh has found the first matching file, all subsequent matches can be found using Function 4Fh (*find next matching file*). This presumes that a file specification with a wildcard character is being used, such as PROG?.EXE or *.ASM. Function 4Fh uses the same disk transfer address as Function 4Eh and updates it with information about each new file that is found. When Function 4Fh finally fails to find another matching file, the Carry flag is set. For a list of the file information in the DTA, see the explanation of Function 4Eh (*find first matching file*). To call Function 4Fh, you need only place the function number in AH:

```
mov   ah,4Fh                  ; find next matching file
int   21h
jc    no_more_files
```

12.1.8 Set Disk Transfer Address (1Ah)

The *disk transfer address* (DTA) is an area set aside for the transfer of file data to memory. Originally, it was used by early DOS file functions, where file control blocks were used to access disk files. Later, its primary use was to provide a buffer for functions 4Eh (*find first matching file*) and 4Fh (*find next matching file*).

Function 1Ah can be used to set the disk transfer address to a location in the data segment. Otherwise, the DTA defaults to offset 80h from the start of the PSP. Most of the time, we reset the DTA to a buffer inside our program because the default location in the

PSP is used for other purposes (such as the program's command line parameters). The following statements, for example, set the DTA to a buffer called **myDTA**:

```
mov   ah,1Ah                   ; set DTA
mov   dx,offset myDTA          ; to buffer in data segment
int   21h
```

12.2 APPLICATION: DISPLAY FILENAMES AND DATES

Using what we have learned about finding matching files and file date/time formats, we can apply these to a program called **Date Stamp** (Example 1) that looks for a file or group of files and displays each name and date. This should provide some insight on how the DIR command works in DOS. We would also like to be able to enter a file specification on the program's command line that includes wildcard characters. The Date Stamp program does the following:

- It retrieves the filename typed on the command line. If no name is found, a message is displayed showing the program syntax.
- It finds the first matching file. If none is found, an appropriate message is displayed before returning to DOS.
- It decodes the date stamp and stores the day, month, and year in variables.
- It displays the filename and date.
- It finds the next matching file. The last three steps are repeated until no more files are found.

Example 1. The Date Stamp Program.

```
title Date Stamp Program                  (DAT.ASM)

; This program displays the name and date stamp for
; each file matching a file specification entered
; on the DOS command line. Uses macros and a
; structure.
; Last update: 10/14/2002

INCLUDE Irvine16.inc

FileControlBlock struc
           db 22 dup(?) ; header info - not used
```

```
  fileTime dw ?         ; time stamp of file
  fileDate dw ?         ; date stamp of file
  fileSize dd ?         ; size of file: not used
  fileName db 13 dup(0) ; name of file found by DOS
FileControlBlock ends

mWriteint macro value, radix:=<10>
    push  ax
    push  bx
    mov   ax,value
    mov   bx,radix
    call  WriteDec
    pop   bx
    pop   ax
endm

mWritestring macro aString
    push  dx
    mov   dx,offset aString
    call  WriteString
    pop   dx
endm
;--------------------------------------------------------
.data
filespec db  40 dup(0)       ; DOS command line
heading  db "Date Stamp Program             (DAT.EXE)"
         db  0dh,0ah,0dh,0ah,0
helpMsg  db "The correct syntax is:  "
         db "DAT [d:][path]filename[.ext]",0dh,0ah,0
DTA      FileControlBlock <>
;--------------------------------------------------------
.code
DOS_error PROTO

main proc
     mov   bx,ds
     mov   ax,@data
     mov   ds,ax
     mov   es,ax
     mov   dx,offset filespec      ; get filespec from
     call  Get_Commandtail         ; the command line
     jc    A2                      ; quit if none found
     mWritestring heading
     call  findFirst        ; find first matching file
     jc    A3               ; quit if none found

A1:  call  decodeDate       ; separate the date stamp
     call  display_filename
     mov   ah,4Fh           ; find next matching file
     int   21h
     jnc   A1               ; continue searching
     jmp   A3               ; until no more matches

A2:  mWritestring helpMsg    ; display help

A3:  exit
main endp

; Find first file that matches the file
; specification entered on command line.

findFirst proc
     mov   ah,1Ah        ; set transfer address
     mov   dx,offset DTA
     int   21h
     mov   ah,4Eh        ; find first matching file
     mov   cx,0          ; normal attributes only
     mov   dx,offset filespec
     int   21h
     jnc   B1            ; if DOS error occurred,
     call  DOS_error     ; display a message
B1:  ret
findFirst endp

; Translate the encoded bit format of a file's
;   date stamp.

.data
month    dw  ?    ; temporary storage for
day      dw  ?    ; month, day, year
year     dw  ?
.code
decodeDate proc
     mov   bx,offset DTA.fileDate
     mov   dx,[bx]            ; get the day
     mov   ax,dx
     and   ax,001Fh           ; clear bits 5-15
     mov   day,ax
     mov   ax,dx              ; get the month
     shr   ax,5               ; shift right 5 bits
     and   ax,000Fh           ; clear bits 4-15
     mov   month,ax
     mov   ax,dx              ; get the year
     shr   ax,9               ; shift right 9 bits
     add   ax,1980            ; year is relative to 1980
     mov   year,ax            ; save the year
     ret
decodeDate endp

; Write both filename and date stamp to console.

display_filename proc
     mWritestring DTA.fileName
     call  fill_with_spaces
     mWriteint month
     call  write_dash         ; display a "-"
     mWriteint day
     call  write_dash         ; display a "-"
     mWriteint year
     call  Crlf
     ret
display_filename endp

; Pad right side of the filename with spaces.

fill_with_spaces proc
     mov   cx,15        ; max file size plus 3 spaces
     ;mov   di,offset DTA.fileName ; get length
     ;call  Str_length  ; AX = length of filename

     INVOKE Str_length, ADDR DTA.fileName
     sub   cx,ax        ; loop counter
     mov   ah,2         ; display character
     mov   dl,20h       ; space
E1:  int   21h          ; write spaces
     loop  E1           ; until CX = 0
     ret
fill_with_spaces endp

write_dash proc        ; write a hyphen
     push  ax
     push  dx
     mov   ah,2
     mov   dl,'-'
     int   21h
     pop   dx
     pop   ax
     ret
write_dash endp
end main
```

Main Procedure. The **main** procedure calls routines to retrieve the command tail and find the first matching file. From that point on, it is essentially a loop that decodes and displays the date and looks for other matching files.

FindFirst Procedure. The **FindFirst** procedure calls Function 1Ah to set the disk transfer address, where file information is stored when matching files are found. We call Function 4Eh to find the first matching file and return to main. The Carry flag is set if no matching files are found.

DecodeDate Procedure. The **DecodeDate** procedure is the most complex one because each field (day, month, year) must be masked and shifted to the right. As each value is isolated, it is stored in a variable. The day of the week occupies bits 0-4, so we clear bits 5-15 and move the result to **day**. The month number is stored in bits 5-8, so AX is shifted 5 bits to the right. We clear all other bits and store the result in **month**. The year number is stored in bits 9-15, so we shift AX 9 bits to the right. We add 80 because the date is always relative to 1980.

12.3 FILE I/O SERVICES

12.3.1 Create File (3Ch)

To create a new file or to truncate an existing file to 0 bytes, Function 3Ch should be used. The file is automatically opened for both reading and writing, but that can be changed by calling Function 43h (change file mode) after the file is open. DS:DX must point to an ASCIIZ string with the name of the file, and CX should contain one or more of the following attribute values:

00h	Normal file
01h	Read-only file

02h	Hidden file
04h	System file (rarely used)

A sample routine that creates a file with a normal attribute is shown here. The file is created on the default drive in the current directory. We would pass the offset of the filename to the procedure in DX:

```
CreateFile proc               ; Input: DX points to filename
     push  cx
     push  dx
     mov   ah,3Ch             ; function: create file
     mov   cx,0               ; normal attribute
     int   21h                ; call DOS
     pop   dx
     pop   cx
     ret
CreateFile endp
```

The following statements show how CreateFile might be called:

```
.data
newfile db  "NEWFILE.DOC",0
handle  dw  ?

.code
mov   dx,offset newfile       ; pass the filename offset
call  CreateFile              ; create the file
jc    display_error           ; error? display a message
mov   handle,ax               ; no error: save the handle
```

If the file is opened successfully, a 16-bit file handle is returned in AX. The value is 5 if this is the first file opened, but it is larger when other files are already open.

Protecting Existing Files. One disadvantage of using Function 3Ch (*create file*) is that one might inadvertently destroy an existing file with the same name. There are a couple of solutions to this problem. You can attempt to open the file for input, using Function 3Dh (*open file*). If the Carry flag is set and AX = 2 (*file not found*), you can safely use the *create file* function.

Another solution is to use Function 5Bh (*create new file*). It aborts and returns error 50h if the file already exists. For example:

```
.data
filename  db  "FILE1.DOC",0
```

```
.code
mov   ah,5Bh                  ; create new file
mov   cx,0                    ; normal attribute
mov   dx,offset filename
int   21h
jc    error_routine
```

Error Codes. If DOS sets the Carry flag, the error number it returns should be 3, 4, or 5. Error 3 (*path not found*) means the file specifier pointed to by DX probably contains a nonexistent directory name. For example, you may have specified the following, when in fact the subdirectory name is ASM, not ASMS:

```
file1  db 'C:\ASMS\FILE1.ASM',0
```

Error 4 (*too many open files*) occurs when you have exceeded the maximum number of open files set by DOS. By default, DOS allows only eight open files. Since the first five of these are in use by DOS (for standard file handles), that leaves only three files for use by application programs. You can change this number with the FILES command in the CONFIG.SYS file (activated when you boot the system). For example,

```
files=32
```

After deducting the five handles used by DOS, there would be 27 handles available for programs to use. But DOS still allows each *program* to have a maximum of 20 open files. It is possible to change this maximum value by calling INT 21h, Function 67h: BX should contain the number of desired handles (1 to 65,534). The following statements set the maximum to 30 files per program:

```
mov  ah,67h
mov  bx,30
int  21h
```

Error 5 (*access denied*) indicates that you may be trying to create a file that already exists and has a read-only attribute. You may be trying to create a file with the same name as a subdirectory, or you may also be trying to add a new entry to a root directory that is already full.

In some versions of DOS, Error 2 (*file not found*) is generated if you leave a carriage return at the end of a filename.

12.3.2 Open File (3Dh)

Function 3Dh opens an existing file in one of three modes: input, output, or input-output. AL contains the file mode to be used, and DS:DX points to a filename. Normal and hidden files can be opened. If the open is successful, a valid file handle is returned in AX:

```
.data
filename      db  'A:\FILE1.DOC',0
infilehandle  dw  ?

.code
mov  ah,3Dh                    ; function: open file
mov  al,0                      ; choose the input mode
mov  dx,offset filename
int  21h                       ; call DOS
jc   display_error             ; error? display a message
mov  infilehandle,ax           ; no error: save the handle
```

File Mode. The file mode placed in AL can have one of three values:

AL	Mode
0	Input (read only)
1	Output (write only)
2	Input-output

To open a file in output mode for sequential writing, Function 3Ch (*create file*) is probably best. On the other hand, to read and write data to a file, Function 3Dh (*open file*) is best. Random-access file I/O requires Function 3Dh.

Error Codes. If CF = 1, AX contains one of the following error codes: Error 1 (*invalid function number*) means you are trying to share a file without having loaded the SHARE program. Error 2 (*file not found*) indicates that DOS was not able to find the requested file. Error 3 (*path not found*) means you specified an incorrect directory name in the filename's path. Error 4 (*too many open files*) indicates that too many files are currently open. Error 5 (*access denied*) means the file may be set to read-only, or it may be a subdirectory or volume name.

12.3.3 Close File (3Eh)

To close a file, call Function 3Eh and place the file's handle in BX. This function flushes DOS's internal file buffer by writing any remaining data to disk and makes the file handle available to other files. If the file has been written to, it is saved with a new file size, time stamp, and date stamp. The following instructions close the file identified by **infilehandle**:

```
.data
infile  db  'B:\FILE1.DOC',0
infilehandle   dw  ?

.code
```

```
mov  ah,3Eh                    ; close file handle
mov  bx,infilehandle
int  21h
jc   display_error
```

The only possible error code is 6 (*invalid handle*), which means the file handle in BX does not refer to an open file.

12.3.4 Read From File or Device (3Fh)

In Chapter 5 we showed how to use Function 3Fh to read from standard input, which ordinarily is the keyboard. This function is very flexible because it can easily read from a disk file. First, you have to call Function 3Dh to open the file for input; then, using the file handle obtained by this call, you can call Function 3Fh and read from the open file.

After calling this function, if the Carry flag is set, the error code is either 5 or 6. Error 5 (*access denied*) probably means the file was open in the output mode, and error 6 (*invalid handle*) indicates that the file handle passed in BX does not refer to an open file. If the Carry flag is clear after the operation, AX contains the number of bytes read.

The information returned by Function 3Fh is useful when checking for end of file. If there is no more data in the file, the value in AX is less than the number of bytes that were requested (in CX). In the following code example, we jump to a label called **Exit** if the end of the file has been reached:

```
.data
bufferSize = 512
filehandle dw ?
buffer  db bufferSize dup(0)

.code
mov  ah,3Fh                    ; read from file or device
mov  bx,filehandle             ; BX = file handle
mov  cx,bufferize              ; number of bytes to read
mov  dx,offset buffer          ; point to buffer
int  21h                       ; read the data
jc   Display_error             ; error if CF = 1
cmp  ax,cx                     ; compare to bytes requested
jb   Exit                      ; yes: quit reading
```

12.3.5 Write to File or Device (40h)

Function 40h is used when writing to a device or a file. Place a valid file handle in BX, place the number of bytes to write in CX, and point DS:DX to the buffer where the data are stored. DOS automatically updates the file pointer after writing to the file, so the next call to Function 40h will write beyond the current position. In the following example, we write the contents of **buffer** to the file identified by **handle**:

```
.data
buffer    db 100h dup(?)             ; output buffer
handle   dw   ?                      ; file handle

.code
write_to_file:
    mov  ah,40h                      ; write to file/device
    mov  bx,handle                   ; file handle returned by OPEN
    mov  cx,100h                     ; number of bytes to write
    mov  dx,offset buffer            ; DX points to the buffer
    int  21h                         ; call DOS
    jc   display_error               ; error? display message.
    cmp  ax,100h                     ; all bytes written?
    jne  close_file                  ; no: disk is full
```

If the Carry flag is set, AX contains error code 5 or 6. Error 5 (*access denied*) means the file is open in the input mode, or the file has a read-only attribute. Error 6 (*invalid handle*) means the number in BX does not refer to a currently open file handle. If the Carry flag is clear but AX contains a number that is less than the requested number of bytes, an input-output error may have occurred. For example, the disk could be full.

12.4 RANDOM FILE ACCESS

Random file processing is surprisingly simple in assembly language. Only one new function needs to be added to what we already know—Function 42h (*move file pointer*), which makes it possible to locate any record in a file. Each high-level language tends to have a specific syntax for random file processing. DOS, on the other hand, makes very little distinction between sequential and random files.

Random access is possible only when the records in a file have a *fixed length*. This is because the record length is used to calculate each record's offset from the beginning of the file. A text file usually has *variable-length* records, each delimited by an end-of-line marker (0Dh, 0Ah). There is no practical way to locate individual variable-length records because their offsets are not determined by their lengths.

In the following illustration, **File1** has fixed-length records, so we calculate the beginning of each record by multiplying the record number minus 1 by 20. **File2** stores the same data in a comma-delimited text file. There are comma delimiters between fields, and end-of-line markers (*0Dh,0Ah*) at the end of each record. The position of any one record cannot be calculated because each record has a different length. Record 2 begins at offset 000F, record 3 at offset 0022, and so on:

File1: Record offsets (hexadecimal): 0000,0014,0028,003C:

```
                1               2               3               4
0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0
1000AU       00300H1003BAKER     02000B2001DAVIDSON  40000H3000GONZALEZ  50000A
```

File2: Record offsets (hexadecimal): 0000,000F,0022,0039:

```
                1               2               3               4
0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0
1000,AU,300,H..1003,BAKER,2000,B..2001,DAVIDSON,40000,H..3000,GONZALEZ,50000,A..
```

12.4.1 Move File Pointer (42h)

Function 42h moves the file pointer to a new location (the file must already be open). The input registers are

AH	42h
AL	Method code (type of offset)
BX	File handle
CX	Offset, high
DX	Offset, low

The *offset* can be relative to the beginning of the file, the end of the file, or the current file position. When the function is called, AL contains a *method code* that identifies how the pointer will be set, and CX:DX contains a 32-bit offset:

AL	***Contents of CX:DX***
0	Offset from the beginning of the file
1	Offset from the current location
2	Offset from the end of the file

Result Values. If the Carry flag is set after the function is called, DOS returns either Error 1 (*invalid function number*) or Error 6 (*invalid handle*). If the operation is successful, the Carry flag is cleared and DX:AX returns the new location of the file pointer relative to the start of the file (regardless of which method code was used).

Example: Locating a Record. Suppose we are processing a random file with 80-byte records, and we want to find a specific record. The LSEEK procedure shown in Example 2 moves the file pointer to the position implied by the record number passed in AX. Assuming that records are numbered beginning at 0, we multiply the record number by the record length to find its offset in the file:

Example 2. Locating a Record with the Lseek Procedure.

```
Lseek proc                  ; AX = record number
   push  bx
   push  cx
   mov   bx,80              ; DX:AX = (AX * 80)
   mul   bx
   mov   cx,dx              ; upper half of offset in CX
   mov   dx,ax              ; lower half of offset in DX
   mov   ah,42h
   mov   al,0               ; method: offset from beginning
   mov   bx,handle
   int   21h                ; locate the file pointer
   pop   cx
   pop   bx
   ret
Lseek endp
```

For example, record 9 would be located at offset 720 and record 0 would be located at offset 0:

```
Offset =  9 * 80 = 720
Offset =  0 * 80 = 0
```

The ReadRecord procedure in Example 3 uses Function 3Fh to read 80 bytes from the file. To read a record, we simply place the desired record number in AX and call both Lseek and ReadRecord:

```
mov   ax,record_number
call  Lseek
call  ReadRecord
```

Example 3. The ReadRecord Procedure.

```
ReadRecord proc
   pusha
   mov   ah,3Fh             ; read from file or device
   mov   bx,handle          ; file/device handle
   mov   cx,80              ; number of bytes to read
```

```
    mov   dx,offset buffer
    int   21h
    popa
    ret
ReadRecord endp
```

Example: Append to a File. Function 42h is also used to append to a file. The file may be either a text file with variable-length records or a file with fixed-length records. The trick is to use method code 2, to position the file pointer at the end of the file before writing any new records. The SeekEOF procedure in Example 4 does this.

Example 4. The SeekEOF Procedure.

```
SeekEOF proc
    pusha
    mov   ah,42h                 ; position file pointer
    mov   al,2                   ; relative to end of file
    mov   bx,handle
    mov   cx,0                   ; offset, high
    mov   dx,0                   ; offset, low
    int   21h
    popa
    ret
SeekEOF endp
```

Using a Negative Offset. If the method code in AL is either 1 or 2, the offset value can be either positive or negative, presenting some interesting possibilities. For example, one could back up the file pointer from the current position (using method 1) and reread a record. This would even work for a text file with variable-length records:

```
mov   ah,42h                     ; function: move pointer
mov   al,1
; method: relative to current position
mov   bx,handle
mov   cx,0
mov   dx,-10                     ; back up 10 bytes
int   21h
jc    error_routine              ; exit if there is an error
mov   ah,3Fh                     ; function: read file
mov   cx,10                      ; read 10 bytes
mov   dx,offset inbuf
int   21h
```

12.5 READING A BITMAP FILE

In this section we present a procedure called ShowBMP that loads a Windows-style bitmap from a file and displays it on the screen. The bitmap can have a resolution up to 320x200, with 256 colors. See the program in Example 5.

When the ShowBMP procedure is called, DS:DX must point to a null-terminated filename. Inside the procedure, we call the OpenInputFile procedure from the link library and quit if the procedure cannot open the file. Next, the ShowBMP procedure reads the bitmap file's *header* record. The ReadHeader procedure reads 54 bytes into a buffer and calls the CheckValid procedure to make sure the bitmap header is recognized.

The CheckValid procedure looks for the string "BM" at the start of the file, and if it finds it, returns. The program calls GetBMPInfo to read the bitmap header record. For example, the header contains the offset of the beginning of the graphic image, the number of colors in the bitmap, and the bitmap's horizontal and vertical resoltuion

The ReadPal procedure reads the graphic pallete into memory. The procedure reads a count of the number of colors and loads the complete palette into a variable. The InitVid procedure inializes the video display into graphcis mode, and the LoadBMP procedure load sand displays the bitmap file. The LoadBMP procedure takes into account that BMP files store graphics images upside-down. The file is read one graphics line at a time, which tends to slow the program down.

This program is just a quick demonstration of the technque of loading bitmaps, but with some experimentation, you should be able to load and display a bitmap anywhere on the screen.

Example 5. Reading and Displaying a Bitmap File.

```
; Bitmap Display Program                  (bitmap.asm)

; This program demonstrates the ShowBMP procedure from Section 12.5
; in "Assembly Language for Intel-Based Computers" by Kip R. Irvine.
; (Third Edition)

; Implementation Notes:
; The bitmap size must be no larger than 320x200. It may be 16-color
; or 256-color. Two test files are supplied with this program. Select
; either one by changing the filename variable at label TEST1.
; The program will look for the bitmap file in the same directory as
; the EXE file. The filename cannot be longer than 8 characters, plus
; the BMP extension.

INCLUDE Irvine16.inc
```

```
Open_infile PROTO
Close_file PROTO

.data
; Two demonstration files supplied with this program:
filename1  DB "16color.bmp",0
filename2  DB "256color.bmp",0
vmode      DB ?

.code
main proc
    mov ax,@data
    mov ds,ax

; Get the current video mode and save it in a variable
    mov  ah,0Fh
    int  10h
    mov  vmode,al

TEST1:
    mov   dx,offset filename2          ; select the bitmap file
    call  ShowBMP                      ; show the bitmap

    mov   ah,0                         ; wait for key
    int   16h

; Restore the startup video mode and exit to OS
    mov   ah,0
    mov   al,vmode
    int   10h

    mov ax,4c00h
    int 21h
main endp

;------------------------------------------------------------------
ShowBMP proc

; This procedure procedure sets loads and displays a Windows bitmap
; file (extension BMP). The maximum resolution is 320x200, with
; 256 colors. By Diego Escala, Miami, Florida, used by permission.
; Receives: DS:DX points to an ASCIIZ string containing the BMP file path.
; Returns: nothing
;-----------------------------------------------------------------
pusha                       ; Save registers
call    Open_infile         ; Open file pointed to by DS:DX
jc      FileErr             ; Error? Display error message and quit
mov     bx,ax               ; Put the file handle in BX
call    ReadHeader          ; Reads the 54-byte header containing file info
jc      InvalidBMP          ; Not a valid BMP file? Show error and quit
call    ReadPal             ; Read the BMP's palette and put it in a buffer
push    es
call    InitVid             ; Set up the display for 320x200 VGA graphics
call    SendPal             ; Send the palette to the video registers
call    LoadBMP             ; Load the graphic and display it
call    Close_file          ; Close the file
pop     es

jmp     ProcDone

FileErr:
mov     ah,9
mov     dx,offset msgFileErr
int     21h
jmp     ProcDone

InvalidBMP:
mov     ah,9
mov     dx,offset msgInvBMP
int     21h

ProcDone:
popa                           ; Restore registers
ret
ShowBMP endp

; Check the first two bytes of the file. If they do not
; match the standard beginning of a BMP header ("BM"),
; the carry flag is set.

CheckValid proc
clc
mov     si,offset Header
mov     di,offset BMPStart
mov     cx,2                    ; BMP ID is 2 bytes long.
CVloop:
mov     al,[si]                 ; Get a byte from the header.
mov     dl,[di]
cmp     al,dl                   ; Is it what it should be?
jne     NotValid                ; If not, set the carry flag.
inc     si
inc     di
loop    CVloop

jmp     CVdone

NotValid:
stc

CVdone:
ret
CheckValid      endp

GetBMPInfo      proc
; This procedure pulls some important BMP info from the header
; and puts it in the appropriate variables.

mov  ax,header[0Ah]          ; AX = Offset of the beginning of the graphic.
sub  ax,54                   ; Subtract the length of the header
shr  ax,2                    ; and divide by 4
mov  PalSize,ax              ; to get the number of colors in the BMP
                             ; (Each palette entry is 4 bytes long).
mov  ax,header[12h]          ; AX = Horizontal resolution (width) of BMP.
mov  BMPWidth,ax             ; Store it.
mov  ax,header[16h]          ; AX = Vertical resolution (height) of BMP.
mov  BMPHeight,ax            ; Store it.
ret
GetBMPInfo      endp

InitVid proc
; This procedure initializes the video mode and makes ES point to
; video memory.

mov     ax,13h
```

```
int     10h                     ; Set video mode to 320x200x256.
push    0A000h
pop     es                      ; ES = A000h (video segment).
ret
InitVid endp

LoadBMP proc
; BMP graphics are saved upside-down.  This procedure reads the graphic
; line by line, displaying the lines from bottom to top.  The line at
; which it starts depends on the vertical resolution, so the top-left
; corner of the graphic will always be at the top-left corner of the screen.

; The video memory is a two-dimensional array of memory bytes which
; can be addressed and modified individually.  Each byte represents
; a pixel on the screen, and each byte contains the color of the
; pixel at that location.

mov     cx,BMPHeight            ; We're going to display that many lines
ShowLoop:
push    cx
mov     di,cx                   ; Make a copy of CX
shl     cx,6                    ; Multiply CX by 64
shl     di,8                    ; Multiply DI by 256
add     di,cx                   ; DI = CX * 320, and points to the first
                                ; pixel on the desired screen line.

mov     ah,3fh
mov     cx,BMPWidth
mov     dx,offset ScrLine
int     21h                     ; Read one line into the buffer.

cld                             ; Clear direction flag, for movsb.
mov     cx,BMPWidth
mov     si,offset ScrLine
rep     movsb                   ; Copy line in buffer to screen.

pop     cx
loop    ShowLoop
ret
LoadBMP endp
```

```
; This procedure checks to make sure the file is a valid BMP,
; and gets some information about the graphic.

ReadHeader proc
mov     ah,3fh
mov     cx,54
mov     dx,offset Header
int     21h                     ; Read file header into buffer.

call    CheckValid              ; Is it a valid BMP file?
jc      RHdone                  ; No? Quit.
call    GetBMPInfo              ; Otherwise, process the header.

RHdone:
ret
ReadHeader endp

; Read the video palette.

ReadPal proc
mov     ah,3fh
mov     cx,PalSize              ; CX = Number of colors in palette.
shl     cx,2                    ; CX = Multiply by 4 to get size (in bytes)
                                ; of palette.
mov     dx,offset palBuff
int     21h                     ; Put the palette into the buffer.
ret
ReadPal endp

SendPal proc
; This procedure goes through the palette buffer, sending information about
; the palette to the video registers.  One byte is sent out
; port 3C8h, containing the number of the first color in the palette that
; will be sent (0=the first color).  Then, RGB information about the colors
; (any number of colors) is sent out port 3C9h.

mov     si,offset palBuff       ; Point to buffer containing palette.
mov     cx,PalSize              ; CX = Number of colors to send.
mov     dx,3c8h
mov     al,0                    ; We will start at 0.
out     dx,al
inc     dx                      ; DX = 3C9h.
```

```
sndLoop:
; Note: Colors in a BMP file are saved as BGR values rather than RGB.

mov     al,[si+2]               ; Get red value.
shr     al,2                    ; Max. is 255, but video only allows
                                ; values of up to 63.  Dividing by 4
                                ; gives a good value.
out     dx,al                   ; Send it.
mov     al,[si+1]               ; Get green value.
shr     al,2
out     dx,al                   ; Send it.
mov     al,[si]                 ; Get blue value.
shr     al,2
out     dx,al                   ; Send it.

add     si,4                    ; Point to next color.
                                ; (There is a null chr. after every color.)
loop    sndLoop
ret
SendPal endp

.data
Header          label word
HeadBuff        db 54 dup('H')
palBuff         db 1024 dup('P')
ScrLine         db 320 dup(0)
BMPStart        db 'BM'
PalSize         dw ?
BMPHeight       dw ?
BMPWidth        dw ?
msgInvBMP       db "Not a valid BMP file.",7,0Dh,0Ah,24h
msgFileErr      db "Error opening file.",7,0Dh,0Ah,24h
end main
```

12.6 REVIEW QUESTIONS

1. If a file currently does not exist, what will happen if function 3Dh opens the file in the output mode?
2. If a file is created using function 3Ch, can it be both written to and read from before it is closed? What if it was created with a read-only attribute?

3. If you want to create a new file but do not want to accidentally erase an existing file with the same name, what steps would your program take?
4. For each of the following error codes returned when INT 21h is called, write a single-sentence explanation of what probably caused the error:

Error Number	Function Being Called
03h	56h (Rename file)
05h	41h (Delete file)
06h	57h (Set date/time)
10h	3Ah (Remove directory)
11h	56h (Rename file)
12h	4Eh (Find first matching file)

5. What do the following instructions imply?

```
.data
filename  db  'FIRST.RND',0

.code
mov   ah,3Dh
mov   al,2
mov   dx,offset filename
int   21h
```

6. When a file is closed, do you need to point DX to its filename?
7. What do you think the effect of the following instructions would be?

```
mov   ah,3Eh
mov   bx,0
int   21h
```

8. When function 3Eh (read from file or device) is called, what does it mean when the Carry flag is set and AX = 6?
9. When function 3Eh is called (with CX = 80h), what does it mean when DOS clears the Carry flag and returns a value of 20h in AX?
10. When function 3Eh is used to read from the keyboard and CX = 0Ah, what will be the contents of the input buffer when the following string is input?

```
1234567890
```

11. When function 40h writes a string to the console, must the string be terminated by a zero byte?
12. When using function 40h to write to an output file, does DOS automatically update the file pointer?
13. If you have just read a record from a random file and you want to rewrite it back to the same position in the file, what steps must you take?
14. Is it possible to move the file pointer within a text file?
15. Write the necessary instructions to locate the file pointer 20 bytes beyond the end of the file identified by **filehandle**.
16. What is the offset of the 20th record in a file that contains 50-byte fixed-length records?
17. What is the purpose of buffering input records?
18. Assuming that bits 0-4 hold a department number and bits 5-7 hold a store number within the following bit-mapped field, what are the values shown here?

```
11000101   store =    department =
00101001   store =    department =
01010101   store =    department =
```

19. The following WRITE_BUFFER procedure is supposed to write the contents of **buffer** to the file identified by **filehandle**. The variable **buflen** contains the current length of the buffer. If the disk is full, the procedure should print an appropriate message. What is wrong with the procedure's logic?

```
.data
filehandle   dw  ?
buflen      dw  ?
buffer  db   80 dup(?)
message db  'Disk is full.$'

.code
write_buffer proc
     mov   ah,40h
     mov   bx,filehandle
     mov   cx,buflen
     mov   dx,offset buffer
     int   21h
     jnc   L1
     mov   dx,offset message
     call  display
L1:  ret
write_buffer endp
```

12.7 PROGRAMMING EXERCISES

1. The "Touch" Utility

For a long time, programmers have used a tool called *touch* that reads a file specifier on the command line, including wildcards, and changes the date/time stamp of all matching files to the current date and time. Write this program in assembly language. If, for example, the user types the following command line, all files in the current directory with an extension of ASM will be updated:

```
touch *.asm
```

One way this program might be useful is, when distributing a set of files to customers for the release of a product, you could assign the same date and time to all files.

2. Text Matching Program

Write a program that opens a text file containing up to 60K bytes and performs a case-insensitive search for a string. The string and the filename are typed on the command line. Display each line from the file on which the string appears and prefix each line with a line number. For example:

```
> search line file1.txt

 2: This is line 2.
10: On line 10, we have even more text.
11: This is a single text line that is even longer.
```

3. Enhanced Text Matching Program

Improve the *text matching* program from the previous exercise as follows:

- Allow wildcard characters in the file specification, so multiple files may be scanned for the same string.
- Include a command-line option to display filenames only. The command should be +/–, the same one used by the **grep** utility supplied with Turbo Assembler. A sample command line that displays the names of all ASM files containing the string "xlat" is

```
search -l+ xlat *.asm
```

4. File Listing Program

Write a program that reads a text file into a buffer and displays the first 24 lines of text. Write the text directly to video memory for the best performance. Provide the following keyboard command functions:

Key	**Function**
PgUp	Scroll up 24 lines
PgDn	Scroll down 24 lines
UpArrow	Scroll up 1 line
DnArrow	Scroll down 1 line
Esc	Exit to DOS

5. Random File Creation Program

Write a program that creates a random file containing student academic information, using data entered from the console. Each record is 27 bytes long, and there should be at least 20 records. The record format is shown here:

Field	**Column**
Student number	1
Last name	6
Course taken	19
Number of credits	27
Grade	28

Here is some sample data, to which you should add at least 12 more records:

```
10024ADAMS       ENG 11003A
10123BEAZLIE     CIS 23014B
10200BOOKER      MAC 11325A
10201BOZEK       BUS 30023B
10330CHARLES     MUS 23003C
10405DANIELS     ART 10022A
10524GONZALEZ    CHM 40004A
10645HART        ENG 11003B
```

6. Student File Maintenance Program

Using the file created in the previous exercise, write a random file update program that displays the following menu:

```
        STUDENT FILE MAINTENANCE

       S   Show a single record
       A   Add a new record
       C   Change (edit) a record
       D   Delete a record
```

```
E   Exit program
```

The user may select records by record number. After each of the menu functions is carried out, return to the menu. Test the program with multiple additions, deletions, and changes to records.

7. ***Enhanced Sector Display Program***

   Using the Sector Display program from the Chapter 11 Exercises as a starting point, add the following enhancement: As a sector is displayed, let the operator press [F3] to write the sector to an output file. Prompt for the filename, and if it already exists, append the current sector to the end of the file. This helps to make the program a useful tool for recovering lost sectors on a disk, as the sectors can be reconverted into files.

12.7.1 Manipulating Disk Directories

1. ***Search for Subdirectories***

   Write a procedure that searches for all entries in a disk's root directory with an attribute of 10h (subdirectory name). Display the names.

2. ***Display a Subdirectory***

   Write a procedure that finds the first subdirectory entry in the root directory, moves to the subdirectory, and displays a list of all its files.

3. ***Recursive Subdirectory Display***

   (Requires knowledge of tree searching methods.) Write a recursive procedure called **ShowTree** that locates and displays the name of each subdirectory in the current directory. For each subdirectory, locate and display all its subdirectories. Use a depth-first search method. For example, print out the directory tree in the following manner:

```
A1
    A1B1
       A1B1C1
       A1B1C2
    A1B2
    A1B3
       A1B3C1
       A1B3C2
A2
    A2B1
    A2B2
A3
    A3B1
```

According to this listing, the root directory contains A1, A2, and A3, and A1 contains A1B1, A1B2, and A1B3. Directory A1B1 contains A1B1C1 and A1B1C2, and so on.

4. ***Showing File Times and Sizes***

   Enhance the Date Stamp program from Example 1 earlier in this chapter so that it also displays each file's time and size.

5. ***Sorting by Filename***

   Enhance the Date Stamp program from Example 1 earlier in this chapter by reading the directory into an array, sorting the array by filename, and displaying the array.

6. ***Sort by Date and Time***

   Enhance the Date Stamp program from Example 1 earlier in this chapter by reading the directory into an array, sorting the array by date and time, and displaying the array.

7. ***Purge Multiple Files***

   Write a program that takes a file specification from the command line, displays the name of each matching file, and asks if the file is to be deleted. When the user enters Y next to any filename, delete the file.

8. ***Search for Files by Date***

   Write a program that searches for all files in the current directory that have a date stamp that is earlier than the current system date. Displays the names of the matching files. To obtain the system date, call INT 21h function 2Ah. The year is returned in CX, the month in DH, and the day in DL. For example, October 12, 1990, would be returned as:

   ```
   CX = 07C6h, DH = 0Ah, DL = 0Ch
   ```

9. ***File Hide and Unhide***

   Write two programs: hide.exe, which hides all files matching a file specifier, and unhide.exe, which unhides all matching files. Only files in the current directory are affected. Output from each program should be a listing of the files that have been hidden or unhidden.

   These programs, which have been available as shareware utilities for many years, are tremendously useful. A major feature of HIDE is that you can protect important files from being deleted by the DOS DEL command. Another is that the average computer user does not know how to view the contents of these files. One good application has to do with deleting all files in a directory *except* a particular file. First, hide the chosen file; next, delete all remaining files in the directory; and finally, unhide the original file.

   Both programs should read a file specifier from the command line, which might be a single filename, a complete path, or a wildcard filename, such as *.ZIP.