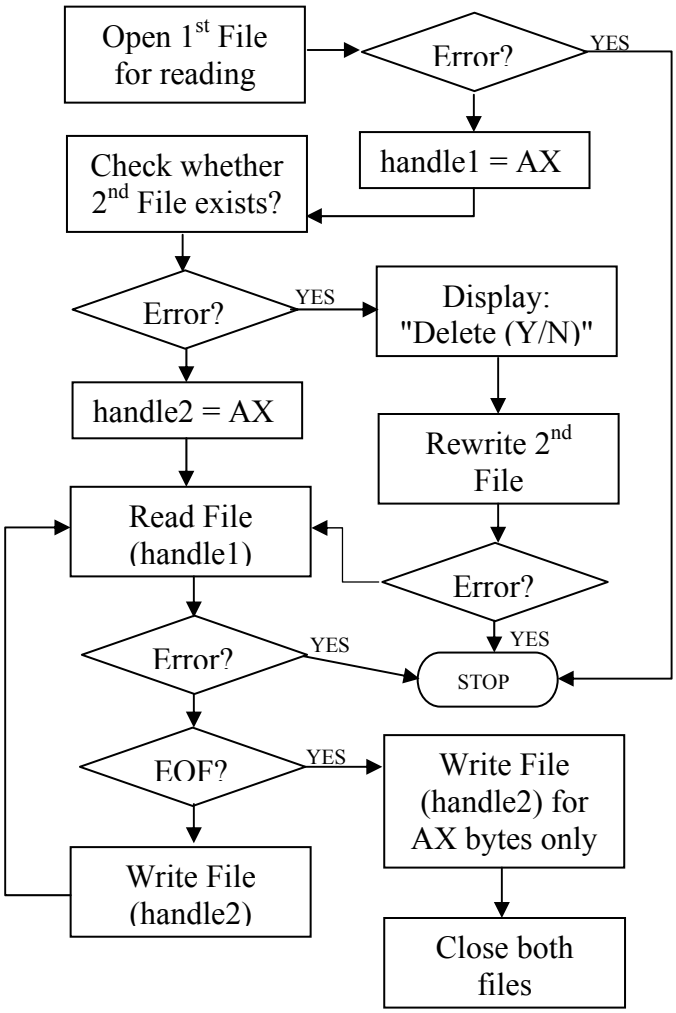
Labwork #12 FILE

Instructor: I Putu Danu Raharja.

Objectives:
To introduce to the students to use File MS-DOS services.

1. Write an Assembly Language program that prompts a user to input two filenames. The program will read the first file, convert its content into uppercase and save it into the second file. Before copying file, the program should check whether or not the second file has been already there. If the second file exists, the program should ask the user whether or not the file will be deleted. Save your work in LAB12.ASM



12.1 DOS LOGICAL STRUCTURES FOR DISK OPERATIONS

DOS uses the following logical structures for disk operations:

1. The Program Segment Prefix (PSP)
2. The File Control Block (FCB)
3. The Disk Transfer Area (DTA)
4. The File Allocation Table (FAT)
5. The Root Directory

12.2 THE PROGRAM SEGMENT PREFIX (PSP)

The PSP is an area that is accessible to both the Operating System and the application program. The DOS loader program, COMMAND.COM, calculates the lowest address in user RAM at which a program can safely be located. The loader than builds a 256 (i.e., 100H) bytes area called the PSP. At the end of the PSP, the loader loads the application program and transfers control to it. When the user program receives control, both the DS and ES registers are pointing to the start of the PSP.

Some portions of the PSP are used internally by DOS and are, therefore, of little use to the programmer. However, there are portions that can be used by the programmer. The most important of these is the *command tail*. This is an area located at offset **81H** of the PSP through offset FFH. DOS copies the remainder of what was typed in the original command line, after the programs name, to the command tail. At offset 80H of the PSP, DOS places the count of the number of characters included in the command tail (excluding the terminating 0DH). The command tail area provides a convenient way to pass information from the command line to the application program.

12.3 THE FILE CONTROL BLOCK (FCB)

The FCB is an area of RAM used by DOS to temporarily store information while performing certain types of disk read/write operations (i.e., file I/O). The normal FCB is a 37-byte data structure usually located in the application program's data segment. The following figure shows the structure of an FCB:

Byte	Information
0	Drive number, 0 = default drive 1 = drive A 2 = drive B, …
1〈…….〉8	Filename, left justified with trailing blanks, for example [myfile]. Invalid characters in filenames are "/[] : ; | < > + =, . All other characters are valid.
9〈-〉11	Filename extension, left justified with trailing blanks (can be all blanks).
12〈-〉13	Current block number relative to the start of the file. A block has 128 records.
14〈-〉15	Logical record size. Default size, set by DOS is 128.
16〈-〉19	Size of the file in bytes.
20〈-〉21	Date the file was created or updates. Bits are mapped as follows. 〈……21…….〉 〈…….20…….〉 7654321 0 765 43210 year(0-119) Month day 1980-2099
22〈……..〉31	Reserved for system use.
32	Number of current record (range 0 to 127). Not initialized by DOS.
33〈……〉36	Record number relative to the start of the file. Not initialized by DOS.

A programmer can create an FCB by initializing the required fields in the programs data segment. For example:

```
·DATA
  FCB  DB   0                ; Code for default drive
       DB   'TESTFILE'       ; 8-character file name
       DB   'DAT'            ; 3-character file extention
       DB   25  DUP(0)
```

An alternative way of creating a valid FCB is using DOS function **29H**.

Old file I/O functions require to access the FCB. These functions are thus known as FCB-type functions. DOS Version 2.0 introduced "file-handle" I/O functions that do not require accessing the FCB or the DTA (Data Transfer Area) and are thus easy to use. The only exceptions are function 4EH (Find First Matching File) and function 4FH (Find Next Matching File).

12.4 THE DISK TRANSFER AREA

The DTA is an area of RAM that is used by some DOS disk read/write (i.e., file I/O) functions for holding data. When control is transferred to an application program, DOS locates a 128-byte DTA at offset 80H through FFH of the PSP. This is the default DTA.

The application program can use DOS function 1AH to relocate the DTA from the PSP to the program's data area. The function requires DS:DX to hold the Segment: Offset address of the new 128-byte DTA. Example:

```
·DATA
        NEWDTA  DB   128  DUP(?)
        . . .
·CODE
        . . .
        mov     AX  , @data
        mov     DS  ,  AX
        mov     DX, OFFSET NEWDTA
        mov     AH  ,  1AH
        int     21H
        . . .
```

The DTA played a much larger role in connection with the now-superseded FCB-functions. Under the currently used handle-functions, its role is relegated to supporting functions 4EH (Find First Matching File) and function 4FH (Find Next Matching File).

12.5 THE FILE ALLOCATION TABLE (FAT)

DOS creates and manages an area of the disk called the *File Allocation Table* (FAT). DOS uses FAT to record the disk's type, to assign the available disk storage space and to keep track of the use of this space. Thus FAT provides a map of how files are stored on a disk.

12.6 THE ROOT DIRECTORY

The root directory can be viewed as an ordered index of the files and subdirectories on the disk or diskette. Each entry in the root directory contains information for locating the file's or subdirectory's space allocation in the FAT.

Directory Entries:

Offset	Length	Description	Format	Comments
0h	8 bytes	File name: 00h= name never used 05h= first character of name is really E5h E5h= file was used, but has been erased 2Eh= entry is a directory (if second byte also 2Eh, cluster field contains cluster # of parent directory	ASCII chars	Must be padded with spaces to fill field.
8h	3 bytes	File type (Extension)	ASCII chars	Must be padded with spaces to fill field.
Bh	byte	File attribute byte	(see table below)	
Ch	10 bytes	RESERVED		
16h	word	Time file last updated	Unsigned 16-bit integer: Hr*2048 + Min*32 + Sec + 2	
18h	word	Date file last updated	Date= (Yr – 1980)*512 + Mon*32 + Day	
1Ah	word	Starting cluster number	Word binary integer	
1Ch	double word	File size	Double word binary integer	

File Attribute Byte:

Bit Number								Meaning if set to 1	Meaning if set to 0
7	6	5	4	3	2	1	0		
							√	Read-only file	Read/write file
						√		Hidden file	Visible file
					√			System file	Normal file
				√				Volume name	Normal file
			√					Directory name	Normal file
		√						File changed since last backup	File unchanged since last backup
√	√							RESERVED	RESERVED

12.7 DEVICES

A device is any peripheral hardware capable of input and/or output.

12.8 FILE DESCRIPTORS

A file descriptor is an ASCIIZ string (i.e. a string terminated by a 0 byte) of maximum length 128 bytes containing *part* or *all* of the following: a drive, path, filename, and extension. Only the filename is **non-optional**. The following are valid examples:

```
F1  DB  'B:FILE1.ASM', 0
F2  DB  'C:\FILE2.DOC', 0
F3  DB  'A:\PROGS\PROG041.ASM', 0
F4  DB  '*.TXT',0
F5  DB  'PROG?.*', 0
```

12.9 FILE HANDLES

DOS uses *handle*s to access files and devices. A handle is a unique 16-bit code used by DOS to identify an open file or a device. There are five standard I/O devices. Each of these devices has a pre-assigned handle by which it can be referenced. With the exception of the error output device, each the devices supports redirection at the DOS command prompt:

Handle	Device
0	Standard input (keyboard)
1	Standard output (video screen)
2	Standard error (video screen)
3	Standard auxiliary (COM1)
4	Standard printer (LPT1)

12.10 SOME FILE-RELATED DOS FUNCTIONS

A. INT 21h function 3CH : Create a file

1. The file is automatically opened for both reading and writing.
2. An existing file is truncated to 0 bytes.
3. It requires create access right on networks.
4. Pathname must be in ASCIIZ form.
5. Volume and subdirectory bits are ignored in attribute byte.

Prior to calling function	Upon return from the function
AH = 3Ch CH = 0 CL = file attribute byte DS:DX = pointer to pathname string	AX = file handle (CF=0) or error code (CF=1) CF=1 if error occurs

If the file was opened successfully the Carry Flag is cleared and DOS returns a 16-bit handle in AX. The Carry Flag is set one if error occurs. Some of errors code is listed in the following table:

Error Code	Meaning
2	File not found
3	Path not found
4	Too many open files
5	Access denied => directory is full or file exists with read-only attribute or file has same name as a subdirectory.
6	Invalid handle
15	Invalid drive
21	Drive not ready

Example:

```
·DATA
PATHNM1  DB  'A:\NEWFILE.DOC',0
HANDLE1  DW  ?

·CODE
. . .
mov    AH ,  3CH
mov    CX ,  00H
mov    DX ,  OFFSET  PATHNM1
int    21H
jc     Trap_ERROR
mov    [HANDLE1],  AX
. . .
```

Example: Creating a Hidden, Read-only file:

```
·DATA
PATHNM1  DB  'A:\NEWFILE.DOC',0
HANDLE1  DW  ?

·CODE
. . .
mov  AH ,  3CH
mov  DX ,  OFFSET  NEWFILE
```

```
MOV CX, 03H
int 21H
jc DISPLAY_ERROR
mov [HANDLE] , AX
. . .
```

This is a good technique for storing important data, such as a password file or a control file, to protect them from being accidentally modified or erased. A hidden, read-only file does not show up in the directory, it cannot be found by normal DOS searches, and it cannot be deleted from the DOS command line. It may, however, be opened for input by a program that knows it exists. The only way the file can be modified is by changing its attribute by DOS function 43H.

B. INT 21h function 5Bh : Create a new file & Protecting existing file:

Function 3CH destroys an existing file. To avoid this, Use DOS function 5BH (Create new file). It aborts and returns error code in AX if the file already exists:

Prior to calling function	Upon return from the function
AH = 5Bh CH = 0 CL = file attribute byte DS:DX = pointer to pathname string	AX = file handle (CF=0) or error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
PATHNM2 DB 'A:\NEWFILE.DOC',0
HANDLE2 DW ?

·CODE
. . .
mov   AH , 5BH
mov   CX , 00H
mov   DX , OFFSET PATHNM2
int   21H
jc    DISPLAY_ERROR
mov   [HANDLE2] , AX
```

C. INT 21h function 3Dh : Open an existing file

1. It requires read/write access right on networks.
2. Pathname must be in ASCIIZ form.

3. If you have created a new file using service 5Bh or 3Ch, you don't need to use this service.

Prior to calling function	Upon return from the function
AH = 3Dh AL = Access code : 0 Input (Read only) 1 Output (Write only) 2 Input-output DS:DX = pointer to pathname string	AX = file handle (CF=0) or error code (CF=1) CF=1 if error occurs

If the file was opened successfully the Carry Flag is cleared and DOS returns a 16-bit handle in AX, otherwise the Carry Flag is set an error code is returned in AX.

Example:

```
·DATA
PATHNM2  DB  'A:\NEWFILE.DOC',0
HANDLE3  DW  ?

·CODE
. . .
mov    AH ,  3DH                ; Request open file ,
mov    AL ,  00H                ; read only
mov    DX ,  OFFSET  PATHNM3
int    21H
jc     DISPLAY_ERROR
mov    [HANDLE3] ,  AX
```

D. INT 21H function 3EH : Close a file

Prior to calling function	Upon return from the function
AH = 3Eh BX = Handle of the file to be closed.	AX = Error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
HANDLE1  DW  ?

·CODE
. . .
mov    AH ,  3EH                ; Request close ,
mov    BX ,  [HANDLE1]          ; file handle
int    21H
```

```
        jc      DISPLAY_ERROR
```

E. INT 21H function 3Fh : Read from a file or a device

The function reads bytes from a file or device into a memory buffer. It is the programmer's responsibility to make the buffer large enough to hold the data. Data is read from a file starting at the position of the file pointer. After a successful read the file pointer is updated to point to the byte following the last byte read. If CF = 0 and AX = 0 an attempt has been made to read *beyond the end of file*; otherwise if CF = 0 and AX > 0 then AX contains the number of bytes read (Thus another way of detecting end of file is CF = 0 and AX < CX). If CF = 1, the error code returned in AX.

Prior to calling function	Upon return from the function
AH = 3Fh BX = Handle of the file to be read CX = number of bytes to read DS:DX = pointer to empty buffer for data	AX = Bytes read or Error code (CF=1) CF=1 if error occurs Buffer contains data read.

Example: This code fragment will read a file for 512 bytes in turn and put in to IN_BUFFER. After a successful read the file pointer is updated to point to the byte following the last byte read and CF=0. If CF=0 and AX=CX, it means the code may continue reading the remaining data.

```
        .DATA
        HANDLE2     DW  ?
        IN_BUFFER   DB  512  DUP(?)

        .CODE
        . . .
@@1:    mov  AH ,  3FH
        mov  BX ,  [HANDLE2]
        mov  CX ,  512                ;Number of bytes to read
        mov  DX ,  OFFSET  IN_BUFFER
        int  21H
        jc  DISPLAY_ERROR
        cmp  AX ,  CX
        JB  EOF_REACHED
        ; manipulate the data
        . . .
        jmp   @@1                     ;Read the next 512 bytes
        . . .
```

Example: The following code fragment will re-read a record of 10 bytes size.

```
·DATA
HANDLE2     DW  ?
IN_BUFFER   DB  512  DUP(?)

·CODE
. . .
MOV   AH ,  42H              ; Position file pointer
mov   AL ,  01H              ; relative to the current position
mov   BX ,  [HANDLE2]
mov   CX ,  0FFFFh
mov   DX ,  -10              ; CX:DX= -10 bytes
int   21h
jc    DISPLAY_ERROR1

mov   AH ,  3FH              ; Read
mov   CX ,  10               ; 10 bytes
mov   DX ,  OFFSET  IN_BUFFER
int   21H
jc    DISPLAY_ERROR2
cmp   AX ,  CX
jb    EOF_REACHED
```

Reading from the keyboard: Set BX to 0 (the keyboard handle), set CX to the maximum number of characters to be read, set DX to the address of the input buffer. The reading terminates when ENTER is pressed. The character count placed in CX should include the CR/ LF characters that DOS appends to the string.

Example:

```
·DATA
IN_BUFFER   DB  512  DUP(?)

·CODE
...
mov   AH ,  3FH
mov   BX ,  00H
mov   CX ,  127
mov   DX ,  OFFSET  IN_BUFFER
int   21H
jc    DISPLAY_ERROR2
...
```

F. INT 21H function 40H : Write to a file or a device

The function writes bytes from a memory buffer into a file or device. Data is written into the file starting at the position of the file pointer. After a successful write the file pointer is updated to point to the byte following the last byte written. If CF = 0 and AX <

CX some type of input-output error occurred; for example the destination disk may be full . If CF = 0 and AX = CX then AX contains the number of bytes written. If CF = 1, the error code returned in AX.

Prior to calling function	Upon return from the function
AH = 40h BX = Handle of the file to be read CX = number of bytes to write DS:DX = pointer to empty buffer containing data	AX = Bytes read or Error code (CF=1) CF=1 if error occurs Buffer contains unchanged data.

Example:

```
·DATA
IN_BUFFER   DB  256  DUP(?)
HANDLE      DW ?

·CODE
...
mov   AH ,  40H              ; Write to file or device
mov   BX ,  [HANDLE]         ; whose handle is in BX
mov   CX ,  100H             ; number of bytes to write
mov   DX ,  OFFSET  BUFFER   ; from buffer
int   21H
jc    DISPLAY_ERROR1
cmp   AX ,  100H             ; All bytes written ?
jne   DISPLAY_ERROR2         ; no: disk is full
```

Writing to the video screen: Set BX to 1 (the video screen handle), set CX to the number of characters to be displayed, set DX to the address of the output buffer.

Example:

```
·DATA
OUT_BUFFER  DB  '8086 ASSEMBLY LANGUAGE'

·CODE
. . .
mov   AH ,  40H
mov   BX ,  01H
mov   CX ,  4
mov   DX ,  OFFSET  OUT_BUFFER
int   21H
jc    DISPLAY_ERROR
```

G. INT 21H function 43H : Get/Set file attribute

Load the address of the file-descriptor in DS:DX. To retrieve the attribute clear AL to 0; the attribute is returned in CX. To change the attribute set AL to 1. The Carry Flag is set if the function fails and the error codes is returned in AX.

Get File Attributes:

Prior to calling function	Upon return from the function
AH = 43h AL = 0 DS:DX = pointer to pathname string.	AX = Error code (CF=1) CH = 0 CL = Attributes byte CF=1 if error occurs

Set File Attributes:

Prior to calling function	Upon return from the function
AH = 43h AL = 1 CH = 0 CL = Attributes byte DS:DX = pointer to pathname string.	AX = Error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
PATHNM2  DB  'TEST.DOC',0
HANDLE2  DW  ?

·CODE
. . .
mov   AH ,  43H              ; Request
mov   AL,  01H               ; set normal
mov   CX  ,  00H             ; attribute
mov   DX ,  OFFSET  PATHNM2
int   21H
jc    DISPLAY_ERROR
```

H. INT 21H function 41H : Delete a file

1. It requires create access rights on networks
2. Pathname must be in ASCIIZ format.
3. Wild card characters (*, ?) are not allowed in the file descriptor.
4. The function cannot delete a file with read-only attribute. To delete a file with read-only attribute, first call DOS function 43H to change its attribute.

Prior to calling function	Upon return from the function
AH = 41h DS:DX = pointer to pathname string.	AX = Error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
PATHNM3  DB  'TEST.DOC',0
HANDLE3  DW  ?

·CODE
. . .
mov   AH , 41h
mov   DX ,  OFFSET  PATHNM3
int   21H
jc    DISPLAY_ERROR
```

I. INT 21H function 42H : Move file pointer

Prior to calling function	Upon return from the function
AH = 42h AL = movement method (see table below) CX = High order of offset to move pointer (in bytes) DX = Low order of offset to move pointer	AX = Low order position (CF=0), or Error code (CF=1) DX = High order position (CF=0) CF=1 if error occurs

The function sets the position of the file pointer for the next read or write operation. The value in CX:DX is interpreted as a 32-bit signed number which specifies an offset according to one of three methods. Which method is to be used is given by the value in AL:

AL	Contents of CX: DX	Type of offset
0	Offset from the beginning of the file	positive
1	Offset from the current position of the file pointer	positive or negative
2	Offset from the end of the file	positive or negative

With method 0, specifying an offset of 0 positions the file pointer at the beginning of the file. Similarly with method 2, specifying an offset of 0 positions the file pointer at the end of the file.

The input registers are:

```
MOV  AH ,  42H
```

MOV AL , method-code (0, 1, or 2)
MOV BX , file-handle
MOV CX , offset , high
MOV DX , offset , low

A valid operation clears the carry flag and returns the new file pointer location in DX:AX relative to the start of the file (regardless of which method code was used). An invalid operation sets the carry flag and returns one of the following error codes in AX:

Error code	Meaning
1	Invalid method code
6	Invalid handle

Note: If the value in CX:DX is too large, the file pointer could be moved past the beginning or end of the file. This is not an error in itself, but it will cause an error when the next file read or write is executed.

Examples:

(1) Position the file pointer 105,247 bytes from the beginning of the file:

```
mov   AH ,  42H
mov   AL ,  00H
mov   BX ,  [HANDLE1]
mov   EDX, 105247
shld  ECX, EDX, 16      ; Now CX:DX = 105,247
int   21H
jc    DISPLAY_ERROR
```

(2) Position the file pointer at the end of the file and hence determine file size:

```
mov   AH ,  42H
mov   AL ,  02H
mov   BX ,  [HANDLE]
xor   CX ,  CX
xor   DX ,  DX
int   21H
; DX:AX = file size
jc    DISPLAY_ERROR
```

(3) Position the file pointer -10 bytes from the current file pointer position:

```
mov   AH ,  42H
mov   AL ,  01H
mov   BX ,  [HANDLE]
mov   EDX, -10
shld  ECX, EDX, 16      ; Now CX:DX = -10
int   21H
jc    DISPLAY_ERROR
```

J. INT 21H function 56H : Rename a file

1. It requires create access rights on networks.

2. Both file-descriptors should not contain wild card characters (*, ?).

3. If the directory path is not the same, but the file name and type specified are, the file is "moved" to the new directory.

4. The disk drives must be the same in both file-descriptors.

Prior to calling function	Upon return from the function
AH = 56h DS:DX = pointer to old pathname. ES:DI = pointer to new pathname	AX = Error code (CF=1) CF=1 if error occurs

Example:
(1) The following program fragment renames PROG1.ASM in the current directory to PROG2.ASM:

```
·DATA
      OLDNAME  DB  'PROG1.ASM', 0
      NEWNAME  DB  'PROG2.ASM', 0
·CODE
.STARTUP
mov   ES ,  AX
. . .
mov   AH ,  56H
mov   DX ,  OFFSET  OLDNAME
mov   DI ,  OFFSET  NEWNAME
int   21H
jc    DISPLAY_ERROR
```

(2) The following program fragment moves 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
.STARTUP
mov   ES ,  AX
. . .
mov   AH ,  56H
mov   DX ,  OFFSET  OLDNAME
mov   DI ,  OFFSET  NEWNAME
int   21H
jc    DISPLAY_ERROR
```

K. INT 21H function 39H : Create a subdirectory

1. This function creates a subdirectory, just like the DOS command MKDIR .

2. It requires create access rights on networks.

3. DS:DX should contain the address of the ASCIIZ string defining the subdirectory.

4. If the drive is not specified in the ASCIIZ string the default drive is assumed.

Prior to calling function	Upon return from the function
AH = 39h DS:DX = pointer to pathname string.	AX = Error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
      PATH  DB  'PROG1.ASM', 0
·CODE
. . .
mov   AH ,  39H
mov   DX ,  OFFSET  PATH
int   21H
jc    DISPLAY_ERROR
```

L. INT 21H function 3AH : Remove a subdirectory

1. This function deletes a subdirectory, just like the DOS command RMDIR.

2. DS:DX should contain the address of the ASCIIZ string defining the subdirectory.

3. If the drive is not specified in the ASCIIZ string the default drive is assumed.

4. The function cannot delete the current directory or a subdirectory containing files.

5. It requires create access rights on networks.

Prior to calling function	Upon return from the function
AH = 3Ah DS:DX = pointer to pathname string.	AX = Error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
      PATHNAME  DB  '\ASM\PROGS\PROG1.ASM', 0
·CODE
. . .
mov   AH ,  3AH
mov   DX ,  OFFSET  PATHNAME
int   21H
jc    DISPLAY_ERROR
```

M. INT 21H function 3BH : Change current directory

1. This function changes the current directory to the specified one, just like the DOS command CHDIR.
2. Pathname must be in ASCIIZ form.
3. Pathname string is limited to 64 characters and may not contain a network path.

Prior to calling function	Upon return from the function
AH = 3Bh DS:DX = pointer to pathname string.	AX = Error code (CF=1) CF=1 if error occurs

Example:

```
·DATA
      PATHNAME  DB  '\ASM\PROGS\PROG1.ASM', 0
·CODE
. . .
mov   AH ,  3BH
mov   DX ,  OFFSET  PATHNAME
int   21H
jc    DISPLAY_ERROR
```

N. INT 21H function 47H : Get current directory path

Prior to calling function	Upon return from the function
AH = 47h DL = Logical drive number DS:SI = pointer to 64-bytes buffer. Buffer = empty	AX = Error code (CF=1) CF=1 if error occurs. Buffer = ASCIIZ pathname (CF=0)

1. DL should contain the drive code (0 = default, 1 = A, 2 = B, etc.).
2. DS:SI should contain the address of a 64-byte buffer. In this buffer DOS places the file-descriptor for the current directory path EXCLUDING the drive letter and the leading backslash. If the requested directory is the root, the value returned is the byte 00H.

Example:

```
·DATA
      BUFFER      DB 64 DUP(?)
·CODE
. . .
mov   AH ,  47h
mov   DL ,  0           ; Default drive
```

```
int   21H
jc    DISPLAY_ERROR
```

O. INT 21H function 0EH : Set default disk drive

To set the default, or logged, disk drive, call this function and pass it the disk drive code in DL (0 = A, 1 = B, 2 = C, etc.). If the number in DL is too large, DOS ignores the request but does not set the carry flag.

Example:

```
MOV  AH , 0EH
MOV  DL , 2
INT  21H
```

P. INT 21H function 19H : Get default disk drive

To find which drive is currently the default drive, call this function. DOS returns the code of the default drive in AL (0 = A, 1 = B, etc.)

Example:

```
MOV  AH , 19H
INT  21H
MOV  [CURRENT_DRIVE] , AL
```

Q. INT 21H Function 1AH: Set Disk Transfer Area

Function 1AH is used to set the DTA to a location in the data segment. Otherwise the DTA defaults to offset 80H from the start of the PSP. There are at least two advantages of resetting the DTA:

1. When an EXE program is loaded, both DS and ES point to the PSP. But we usually reset DS and ES to point to the data segment, thereby losing the original PSP segment address.
2. The default DTA is the same location as the DOS command tail. Thus the original command tail would be overwritten the first time DOS performed a disk transfer.

Example:

```
·DATA
      MYDATA      DB  43 DUP(?)
·CODE
. . .
.STARTUP
mov   DX, OFFSET  MYDATA
mov   AH, 1Ah
int   21h
```