

Files

Function 3Dh: open file

- **opens an existing file for reading, writing or appending**
- **bits are numbered from right to left**

input:

AH = 3Dh

AL = bits 0-2 Access mode

= bits 4-5 Sharing mode

DS:DX = segment:offset of ASCII pathname

output

CF = 0 ; function is successful

AX = handle

CF = 1 error has occurred

AX = error code

- **an ASCIIZ string is like an ASCII string with a zero on the end instead of a dollar sign**
- **file handle must be saved**
 - use a register (but can't use again and waste of a register)
 - use a variable

Files

Function 3Eh: close file

- **closes an opened file**

input:

AX = 3Eh

BX = file handle

= bits 4-5 Sharing mode

output

CF = 0 ; function is successful

AX = destroyed

CF = 1 error has occurred

AX = error code

**06h file not opened or
unauthorised file handle**

- **don't call this function with a zero handle because that will close the standard input (the keyboard) and you won't be able to enter anything**

Files

Function 3Fh: read file/device

- **reads bytes from a file or device to a buffer**

input:

AX = 3Fh

BX = handle

CX = number of bytes to be read

DS:DX = segment:offset of a buffer

output

CF = 0 ; function is successful

AX = no. of bytes read

CF = 1 error has occurred

AX = error code

- **if CX = 0 and AX = 0 => file pointer already at end of file**
- **this function can be used to get input from the keyboard**
 - use a handle of 0
 - it stops reading after the first carriage return or has read specified no. of chars
 - good, easy method to only let the user enter a certain amount of chars

Files

Function 3Ch: create file

- **creates a new empty file**

input:

AX = 3Ch

CX = file attribute

DS:DX = segment:offset of ASCII pathname

output

CF = 0 ; function is successful

AX = handle

CF = 1 error has occurred

AX = error code

- **if a file of the same name exists then it will be lost**
- **make sure that there is no file of the same name**
- **you can use function 4Eh to check**

Files

Function 4Eh: find first matching file

- **searches for the first file that matches the filename given**

input:

AX = 4Eh

CX = file attribute mask

DS:DX = segment:offset of ASCII pathname

output

CF = 0 ; function is successful

DTA [Disk Transfer Area] =

FindFirst data block

An example of checking if file exists:

File DB "C:\file.txt",0 ;name of file that we want

mov dx,OFFSET File ;address of filename

mov cx,3Fh ;file mask 3Fh - any file

mov ah,4Eh ;function 4Eh - find first file

int 21h ;call DOS service

jc NoFile

;print message saying file exists

NoFile:

;continue with creating file

String Instructions

MOVS*: move string: byte/word/double word at DS:SI to ES:DI

- e.g. movsb, movsw, movsd

CMPS*: compare string: byte/word/double word at DS:SI to ES:DI

- e.g. cmpsb, cmpsw, cmpsd

SCAS*: search string: search for AL, AX or EAX in string at ES:DI

- e.g. scasb, scasw, scasd

STOS*: move byte/word/double word from AL, AX or EAX to ES:DI

- e.g. stosb, stosw, stosd

LODS*: move byte/word/double word from DS:DI to AL, AX or EAX

- e.g. lodsb, lodsw, lodsd

- these string commands can be used with the rep instruction which repeats the instruction CX times

How to Find the DOS Version

Method 1:

```
mov ah,30h ;function 30h - get MS-DOS version
int 21h ;call DOS function
```

- returns the major version number in AL and the minor version number in AH
- e.g. version 4.01: AL=4 and AH=01

Method 2:

```
mov ah,33h ;function 33h - actual DOS version
mov al,06h ;subfunction 06h
int 21h ;call interrupt 21h
```

- only works for DOS version 5+
- returns data in BL and BH

Multiple Pushes and Pops

All on one line:

```
push ax bx cx dx      ; save registers
```

```
pop dx cx bx ax       ; restore registers
```

PUSHA

- pushes all (modifiable) registers onto the stack
- less typing, smaller instruction, faster

POPA

- analogous to PUSHA

PUSHAD/POPAD

- do the same for 32-bit registers

Shifts: faster Multiplication and Division

- **mults and divs are very slow**
- **use shifting instead:**
 - SHL unsigned multiple by two
 - SHR unsigned divide by two
 - SAR signed divide by two
 - SAL same as SHL
- **syntax: SHL operand1, operand2**
- **on 8086, operand2 can only be 1**
- **on 286/386, operand2 cannot be higher than 31**

Loops

- **loop command is better (?) than jump instructions**
- **place number of iterations in the CX register**
- **loop**
 - decrements CX by 1 (i.e. $CX = CX - 1$)
 - does short jump to label if not zero

Loop:

```
mov cx, 100
```

```
label: .....
```

```
; code here
```

```
loop label
```

Jump

```
mov cx, 100
```

```
label: ....
```

```
; code here
```

```
dec cx ; cx = cx -1
```

```
jnz label; continue until done
```

```
* DEC/JNZ faster on 486+
```

Using the Debugger

- **We can use the debugger to see the contents of registers and segments**
 - F5 Size Window
 - F7 Next Instruction
 - F9 Run
 - Alt F4 Step Backwards
- **All values are in hex**
- **see listing 11: debug.asm**

More Output in Text Modes

- **Listing12 - listing 17:**
 - 12: move the cursor to display a string of text where you want it to go
 - 13: writes to the screen using the file function 40h/int 21h
 - 14: uses function 13h/int 10h to write a string anywhere on the screen in a specified colour (setup harder)
 - 15: writes to the screen using rep stosw to put the writing in video memory
 - 16: writes a string to video memory
 - 17: uses mode 13h which is only available on VGA, MCGA+ cards (good for graphics)
 - check if mode 13h is supported and set video mode
 - function 0Ch can be used to write graphics pixel
 - 18: plotting pixels in mode 13h/int 10h
 - optimisations possible