INT 21h

DOS FUNCTIONS

- The Intel CPU recognizes two types of interrupts namely hardware interrupt when a peripheral devices needs attention from the CPU and software interrupt that is call to a subroutine located in the operating system. The common software interrupts used here is INT 21H for DOS services.

- The PC which was designed by the IBM used INTEL 8088/8086 as processor and Microsoft Disk Operating System(MSDOS) as the OS.
- DOS service can be accessed using INT instruction in our program.
- Each interrupt service is given an 8 bit number. We have to use 8 bit number with INT instruction which executes ISR for that service.

- INT 21H: It is called the DOS function call for keyboard operations follow the function number. The service function number is provided in AH register.

- INT 21h / AH=1 - read character from standard input, with echo, result is stored in AL. if there is no character in the keyboard buffer, the function waits until any key is pressed. example:

```
mov ah, 1
int 21h
```

- INT 21h / AH=2 - write character to standard output.

  entry: DL = character to write, after execution AL = DL.
- example:
```
mov ah, 2
mov dl, 'a‘
int 21h
```

- INT 21h / AH=7 - character input without echo to AL. if there is no character in the keyboard buffer, the function waits until any key is pressed.
- example:

```
mov ah, 7
int 21h
```

- INT 21h / AH=9 - output of a string at DS:DX. String must be terminated by '$'.
- example: msg db "hello world $"

```
mov dx, offset msg
mov ah, 9
int 21h
```

- INT 21h / AH=0Ah - input of a string to DS:DX, first byte is buffer size, second byte is number of chars actually read. this function does not add '$' in the end of string. to print using INT 21h / AH=9 you must set dollar character at the end of it and start printing from address DS:DX + 2.

Max.Size	**Actual size**	**Str.**	**…**	**…**	**$**

```
example:    print buffer db 10,?, 10 dup(' ')
                  mov dx, offset buffer
                  mov ah, 0ah
                  int 21h
            print: xor bx, bx
                  mov bl, buffer[1]
                  mov buffer[bx+2], '$'
                  mov dx, offset buffer + 2
                  mov ah, 9
                  int 21h
```

- **INT 21h / AH= 39h - make directory.**
- entry: **DS:DX -> ASCIZ pathname; zero terminated string, for example:**

```
filepath DB "C:\mydir", 0  ; path to be created.
mov dx, offset filepath
mov ah, 39h
int 21h
```

- Return: **CF clear if successful AX destroyed. CF set on error AX = error code.**
- Note: all directories in the given path must exist except the last one.

- **INT 21h / AH= 3Ah - remove directory.**
- Entry: **DS:DX -> ASCIZ pathname of directory to be removed.**
- Return:
- **CF is clear if successful, AX destroyed CF is set on error AX = error code.**
- Notes: directory must be empty (there should be no files inside of it).

INT 21h / AH= 3Ch - create or truncate file.

- entry: **CX = file attributes:**

- mov cx, 0 ; normal - no attributes.
- mov cx, 1 ; read-only.
- mov cx, 2 ; hidden.
- mov cx, 4 ; system
- mov cx, 7 ; hidden, system and read-only!
- mov cx, 16 ; archive
- **DS:DX -> ASCIZ filename.**

- returns:

- **CF clear if successful, AX = file handle. CF set on error AX = error code.**

- filename db "myfile.txt", 0
- handle dw ?

- mov ah, 3ch
- mov cx, 0
- mov dx, offset filename
- mov ah, 3ch
- int 21h
- jc err
- mov handle, ax
- jmp k
- err:
- ; ....
- k:

- **INT 21h / AH= 3Dh - open existing file.**
- Entry: **AL = access and sharing modes:**

```
mov al, 0   ; read
mov al, 1   ; write
mov al, 2   ; read/write
```

- **DS:DX -> ASCIZ filename.**

- Return:
- **CF clear if successful, AX = file handle.**
- **CF set on error AX = error code.**

- note 1: file pointer is set to start of file.
- note 2: file must exist.

- example:

- filename db "myfile.txt", 0
- handle dw ?

- mov al, 2
- mov dx, offset filename
- mov ah, 3dh
- int 21h
- jc err
- mov handle, ax
- jmp k

INT 21h / AH= 3Eh - close file.

- Entry: **BX = file handle**

- Return:

- **CF clear if successful, AX destroyed.**
- **CF set on error, AX = error code (06h).**

INT 21h / AH= 3Fh - read from file

- Entry:

- **BX = file handle.**
- **CX = number of bytes to read.**
- **DS:DX -> buffer for data.**

- Return:
- CF is clear if successful - AX = number of bytes actually read; 0 if at EOF (end of file) before call.
- CF is set on error AX = error code.

INT 21h / AH= 40h - write to file.

- entry: **BX = file handle.**
- **CX = number of bytes to write.**
- **DS:DX -> data to write.**

- return:

- **CF clear if successful; AX = number of bytes actually written.**
- **CF set on error; AX = error code.**

- note: If CX is zero, no data is written, and the file is truncated or extended to the current position
- Data is written beginning at the current file position, and the file position is updated after a successful write
- The usual cause for AX < CX on return is a full disk.

INT 21h / AH= 41h - delete file (unlink).

- Entry:
- **DS:DX -> ASCIZ filename (no wildcards, but see notes).**
- return:
- **CF clear if successful, AX destroyed. AL is the drive of deleted file (undocumented).**
- **CF set on error AX = error code.**
- Note: DOS does not erase the file's data; it merely becomes inaccessible because the FAT chain for the file is cleared
- Deleting a file which is currently open may lead to file system corruption.

INT 21h / AH= 42h - SEEK - set current file position.

- Entry: **AL = origin of move:**

  0 - start of file.

  1 - current file position.

  2 - end of file.
- **BX = file handle.**
- **CX:DX = offset from origin of new file position.**

- Return: CF clear if successful, DX:AX = new file position in bytes from start of file.
- CF set on error, AX = error code.

- **INT 21h / AH= 56h - rename file / move file.**
- Entry:
- **DS:DX -> ASCIZ filename of existing file.**
- **ES:DI -> ASCIZ new filename.**
- Return:
- **CF clear if successful.**
- **CF set on error, AX = error code.**
- Note: allows move between directories on same logical drive only; open files should not be renamed!

- **INT 21h / AH=4Ch - return control to the operating system (stop program).**

XLAT/XLATB

- Used to translate a byte from one code to another code.

  Syntax:XLAT

  (AL)<-(BX+AL)
- The byte stored in AL register is replaced by the byte stored at location BX+AL in the lookup table with BX as the base of the lookup table and contains offset address of the lookup table stored in data segment.

THANK YOU