EENG410 – Microprocessors I

Fall 06/07 – Lecture Notes # 12

<u>Outline of the Lecture</u>

- **BIOS and DOS programming in Assembly**
- **BIOS INT 10H**
- **DOS INT 21H**

BIOS AND DOS PROGRAMMING IN ASSEMBLY

- BIOS and DOS contain some very useful subroutines, which can be used through INT (interrupt) instruction.
- The INT instruction works like a FAR call. When it is invoked, it saves CS:IP and the flags on the stack and goes to the subroutine associated with the interrupt.

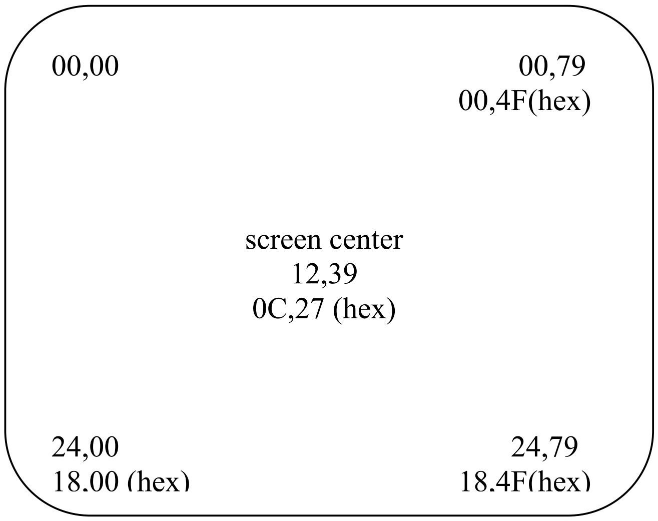
```
INT     xx      ;the interrupt number can be 00 – FFH (256 possible interrupts)
```

BIOS INT 10H PROGRAMMING

- INT 10H subroutines are in the ROM BIOS of the 80x86-based IBM PC.
- Depending on the value put in **AH** many function associated with the manipulation of screen text or graphics is performed.
- Among these functions, clearing the screen, changing the cursor position, change the screen color and drawing lines on the screen.

Monitor screen in text mode

- In normal text mode the screen is divided into 80 columns and 25 rows.
- Top left = 00,00
  Bottom left = 24,00 (decimal)
  Bottom right = 24,79 (decimal)



- **<u>Clearing the screen (INT 10H function 06H)</u>**

- **AH=06** <u>Scroll window up</u>
- To clear the screen with INT 10H the following registers must contain certain values.

AH=06, AL=00, BH=07, CX=0000
DH=24, DL=79

The code:

```
MOV  AH,06    ;AH=06 select the scroll function
MOV  AL,00    ;number of lines to scroll (if AL=00 the entire page)
MOV  BH,07    ;the display attribute (BH=07 normal)
MOV  CH,00    ;row value of the start point
MOV  CL,00    ;column value of the start point
MOV  DH,24    ;row value of the ending point
MOV  DL,79    ;column value of the ending point
INT  10H      ;invoke the interrupt
```

More efficient coding:

```
MOV  AX,0600H   ;scroll entire screen
MOV  BH,07      ;normal attribute
MOV  CX,0000    ;start at 00,00
MOV  DX,184FH   ;end at 24,79 (hex=18,4F)
INT  10H        ;invoke the interrupt
```

- **INT 10H function 02: setting the cursor to a specific location**

AH=02 Set cursor position

BH= page number (BH=00) ; 00 represents the current viewed page.
DH = row
DL = column

Ex: Write the code to set the cursor position to row = 15 (= 0FH) and column = 25 (=19H).

```
    MOV  AH,02         ;set cursor option
    MOV  BH,00         ;page 0
    MOV  DH,15         ;row position
    MOV  DL,25         ;column position
    INT  10H           ;invoke interrupt 10H
```

Ex: Write a program segment to (1) clear the screen and (2) set the cursor at the center of the screen.

```
;clearing the screen
    MOV  AX,0600H      ;scroll the entire page
    MOV  BH,07         ;normal attribute
    MOV  CX,0000       ;row and column of the top left
    MOV  DX,184FH      ;row and column of the bottom right
    INT  10H           ;invoke interrupt 10H

;setting the cursor to the center of the screen
    MOV  AH,02         ;set cursor option
    MOV  BH,00         ;page 0
    MOV  DH,12         ;center row position
    MOV  DL,39         ;center column position
    INT  10H           ;invoke interrupt 10H
```

- **INT 10H function 03: get current cursor position**

AH=03 Read cursor position and size

```
Ex:     MOV  AH,03     ;option 03 of BIOS INT 10H (read cursor position and size)
        MOV  BH,00     ;choose current (00) page
        INT  10H       ;interrupt10H routine
```

After the execution of the above program: DH = current row, DL = current column CX will provide info about the shape of the cursor.

DOS INT 21H PROGRAMMING

- ➢ INT 21H subroutines are provided by DOS Operating system.
- ➢ Depending on the value put in **AH** many functions such as inputting data from the keyboard and displaying it on the screen can be performed.

INT 21H option 09: outputting a string of data to the monitor

- ➢ INT 21H can be used to send a set of ASCII data to the monitor.
- ➢ Register settings before INT 21H is invoked: **AH=09**
  DX = the offset address of the ASCII data to be displayed.

- The address in DX register is an offset address. Data is assumed to be the data segment.
- INT 21H option 09 will display the ASCII data string pointed at by DX until it encounters the dollar sign '$'. Note that this option cannot display '$' character on the screen.

Ex:
```
    ………………..
    DATA_ASC           DB     'I love MICROPROCESSORS','$'
    ……………….
                       MOV    AH,09                    ;option 09 to display string of data
                       MOV    DX,OFFSET DATA_ASC       ;DX offset address of data
                       INT21H                          ;invoke the interrupt
```

INT 21H option 02: outputting a single character to the monitor
- To do that: AH=02 (AH is given 02)
  DL = is loaded with the ASCII character to be displayed.
  INT 21H is invoked.

Ex:
```
        MOV   AH,02          ;option 02 displays one character
        MOV   DL,'Y'         ;DL holds the character to be displayed
        INT   21H            ;invoke the interrupt.
```

* This option can be used to display '$' sign on the monitor.

INT 21H option 01: Keyboard input with echo (inputting a single character with echo)
- This function waits until a character is **input from the keyboard**, then echoes(displays) it to the monitor.
- After the interrupt the character will be in AL.

Ex:
```
        MOV   AH,01          ;option 01 inputs one character
        INT   21H            ;after the interrupt, AL = input character (ASCII)
```

INT 21H option 07: Keyboard input without echo
- This function waits until a character is **input from the keyboard**, then character is not displayed (echoed) to the monitor.
- After the interrupt the character will be in AL.

Ex:
```
        MOV   AH,07          ;keyboard input without echo
        INT   21H            ;after the interrupt, AL = input character (ASCII)
```

INT 21H option 0AH: Inputting a string of data from the keyboard
- This function enables input a string of data from the keyboard and to store it in the data segment.
- The register settings are: AH=0AH
  DX= offset address of the string to be stored (called as the **buffer** area)

- Buffer area must be defined in the data segment.

Ex:
```
        ……………………
                   ORG    0010H
        DATA1      DB     6,?,6 DUP(FF)             ;0010H=06, 0012H – 0017H=FF
        ……………………

                   MOV    AH,0AH                    ;string input option of INT 21H
                   MOV    DX,OFFSET DATA1           ;load the offset address of buffer
                   INT    21H                       ;invoke the interrupt
```

- The following shows the memory contents of offset 0010H: Before input is entered!!

0010	0011	0012	0013	0014	0015	0016	0017
06	00	FF	FF	FF	FF	FF	FF

- When the program is executed and the data is entered through the keyboard, the program will not exit until the return key is pressed. Assume the data entered through the keyboard was, "USA" ,RETURN>

- The contents of memory locations starting at offset 0010H will be:

0010	0011	0012	0013	0014	0015	0016	0017
06	03	55	53	41	0D	FF	FF
		U	S	A	CR		

- The following is the step by step analysis:

0010=06	The size of the buffer must be defined in the first location
0011=03	The keyboard was pressed 3 times, U, S, A (excluding the RETURN)
0012=55	the hex ASCII code for letter U
0013=53	the hex ASCII code for letter S
0014=41	the hex ASCII code for letter A
0015=0D	the hex ASCII code for CR (carriage return)

Note that the value 03 is generated and stored by DOS to indicate the number of characters that entered.

- **INT 16H Keyboard Programming:**

  - In the previous sections it was explained that INT 21H function AH=07, **waits** for the user to input a character.
  - In some programs a task must run continuously while checking a key press? Such cases require to use **INT 16H**.

Checking a key press: AH=01

```
Ex:   MOV  AH,01           ;check for key press
      INT  16H             ;using INT 16H
```

After the execution, **ZF=0**,if there is **a key press**;
ZF=1 if there is **no key press**.

Which key is pressed?

- In order to find out which key is pressed immediately after the above routine (INT 16H function AH=01) the following routine (INT 16H function AH=00) must be called.

```
Ex:   MOV  AH,0            ;get key pressed
      INT  16H             ;using INT 16H
```

- Upon return, AL contains the ASCII character of the pressed key.

Outline of the Lecture

- **MACROS in Assembly Language**
- **MACRO definition**

Macros are predefined functions which involve a group of instructions to perform a special task which can be used repeatedly.

For example:

- in order to print a string to the screen INT 21H together with 2 more instructions can be used (3 lines of code).
- It doesn't make sense to rewrite them every time they are needed.
- In order to reduce the time to write the code and reduce the length of the code macros can be used.
- Macros allow programmer to define the task (set of codes to perform a specific job) once only and invoke it whenever/wherever it is needed.

MACRO definition:

```
name        MACRO     dummy1,dummy2,dummy3,…,dummyN
            …
            …
            …
            ENDM
```

Ex: Write a macro called STRING to which display a string of text to the monitor.

```
STRING      MACRO     DATA1
            MOV       AH,09
            MOV       DX,OFFSET DATA1
            INT       21H
            ENDM
```

The above code is the macro definition. You can invoke the above macro as follows:

```
; from the data segment
MESSAGE1  DB     'What is your name?','$'
:
:

;from the code segment
          :
          STRING        MESSAGE1   ; Assembler will invoke the macro to perform the defined function.
          :
```

Using MACROS in an Assembly Language Program:

- ➢ The Macros are defined outside the Code segment of an Assembly Language program and can be invoked inside the code segment.
- ➢ There can be comments in Macro definition

Example: the following program contains 3 Macro definitions which are: clear the screen, display a string and set the cursor position.

```
;THE FOLLOWING PROGRAM USES MACROS
;-------------------------------------------------
CLSCREEN     MACRO                                   ;THIS MACRO  CLEARS THE SCREEN
             MOV    AX,0600H
             MOV    BH,07
             MOV    CX,0
             MOV    DX184FH
             INT    10H
             ENDM
;-------------------------------------------------
DISPSCREEN   MACRO   STRING                          ;THIS MACRO  DISPLAYS A STRING OF DATA
             MOV    AH,09
             MOV    DX,OFFSET STRING
             INT    21H
             ENDM
;-------------------------------------------------
CURSOR       MACRO   ROW,COLUMN                      ;THIS MACRO SETS THE CURSOR POSITION
             MOV    BH,00
             MOV    AH,02
             MOV    DH,ROW
             MOV    DL,COLUMN
             INT    10H
             ENDM
;-------------------------------------------------
.MODEL SMALL
.STACK  64

.DATA
MESSAGE1            DB      'My name ','$'
MESSAGE2            DB      'is Ali','$'
MESSAGE3            DB      'What is ','$'
MESSAGE4            DB      'your name?','$'

.CODE
MAIN:        MOV       AX,@DATA
             MOV       DS,AX
             CLSCREEN
             CURSOR 2,4
             DISPSCREEN  MESSAGE1
             CURSOR 3,4
             DISPSCREEN  MESSAGE2
             CURSOR 10,4
             DISPSCREEN  MESSAGE3
             CURSOR 11,4
             DISPSCREEN  MESSAGE4
             MOV    AH,4CH
             INT       21H
             END        MAIN
```

LOCAL directive and its use in macros:

- If a label is needed to be used in a macro (e.g. JNZ BACK) the label must be declared as LOCAL to the macro.
- The LOCAL directive must be right after the MACRO directive.
- The local directive can be used to declare all names and labels at once as follows.

```
LOCAL        name1                    OR
LOCAL        name2                  <==>   LOCAL name1,name2,name3
LOCAL        name3
```

;The Following Program Defines a Macro to multiply two words by repeated addition. Macro is used in the main
;procedure below 3 times.

```
;-------------------------------------------------
MULTIPLY        MACRO           VALUE1, VALUE2, RESULT
        LOCAL BACK                          ;
                ;THIS MACRO  COMPUTES RESULT = VALUE1 x VALUE2
                MOV     BX,VALUE1
                MOV     CX,VALUE2
                SUB     AX,AX
                MOV     DX,AX
BACK:           ADD     AX,BX
                ADC     DX,00
                LOOP    BACK
                MOV     RESULT,AX
                MOV     RESULT+2,DX
                ENDM
;-------------------------------------------------
.MODEL SMALL
.STACK  64

.DATA
RESULT1         DW      2 DUP(?)
RESULT2         DW      2 DUP(?)
RESULT3         DW      2 DUP(?)

.CODE
MAIN:           MOV        AX,@DATA
                MOV        DS,AX
                MULTIPLY        2000,500,RESULT1
                MULTIPLY        2500,500,RESULT2
                MULTIPLY        300,400,RESULT3
                MOV     AH,4CH
                INT     21H
                END          MAIN
```

Note: The reason why the LOCAL directive must be used is as follows: When a MACRO is assembled in the program, the body of the MACRO is expanded as many times as the MACRO function is invoked/called. This means that, for example in the above case, the same BACK label will be expanded in the program 3 times. As a result, there will be the same BACK label in 3 different locations. This confuses the processor so it is an error.

However if LOCAL directive is used the label which is defined as LOCAL in a MACRO will be the only one to be considered. So, in the above example when a jump to BACK label is needed it will be the local BACK label not the other two.