Laboratory Manual

CS/EEE/INSTR F241Microprocessor
Programming & Interfacing
DOS INTERRUPTS – BASIC KEYBOARD & DISPLAY
CONSOLE INTERRUPTS

Anupama KR & Meetha.V.Shenoy
BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI – KK BIRLA GOA CAMPUS

LAB 3 – DOS INTERRUPTS – BASIC KEYBOARD & DISPLAY CONSOLE INTERRUPTS

3.1 Introduction to DOS & BIOS Interrupts

The BIOS is specific to the individual computer system. It contains the default resident hardware dependent drivers for several devices including the console display, keyboard, and the boot disk device. Have you ever wondered how you are able to type characters and view them even when no OS is loaded, like when the computer is booting?

The BIOS operates at a more primitive level than the DOS kernel. In a X86 based systems, there are total of 256 possible interrupts, out of which some interrupts are reserved for DOS, BIOS services, which the programmer can access by loading the function number into appropriate registers and then invoking the interrupt.

In providing the DOS functions, the MS-DOS kernel communicates with the BIOS device drivers through the BIOS functions. DOS then provides something like a wrapper over the primitive BIOS function. By providing the wrapper, DOS makes the functions easier to use and more reliable, but in turn, reduce the flexibility and power of the functions. By using the BIOS interrupts directly, the programmer bypasses the DOS kernel.

In the next two experiments you will using INT 21 H (DOS Interrupts) – Here we use INT 21H to access value of key pressed and display charcters.

3.2 Keyboard Interrupts

Purpose: Input a character from keyboard (STDIN) with Echo

Program Segment

```
MOV     AH, 01h                 ; AH -01 parameter for INT 21h
INT     21h
```

You have to run the program using debug/debug32 and the ASCII key you press will be stored in AL register.

Purpose: Input a character from keyboard (STDIN) without Echo

Program Segment

```
MOV     AH, 08h                 ; AH -08 parameter for INT 21h
INT     21h
```

You have to run the program using debug/debug32 and the ASCII key you press will be stored in AL register. The difference is the character will not be visible on the screen when you type it

Purpose: Input a string from keyboard (STDIN)

Program

```
.data
max1    db      32              ; 32 is max no. of chars that a user can type in (max possible – 255)
act1    db      ?               ; actual count of keys that user types will be stored here after int has
                                ; executed (Note this cannot exceed the value specified in max1 –
                                ; actual  keys you enter will 31 as the 32nd will be Enter key)
inp1    db      32 dup(0)       ; Reserve 32 locations for input string
.code
.startup
        LEA     DX,max1
        MOV     AH, 0Ah
        INT     21h
```

After the interrupt, act 1 will contain the number of characters read, and the characters themselves will start at inp1 The characters will be terminated by a carriage return (ASCII code 0Dh), although this will not be included in the count (Note: this will not be included in the ACT1 but you have to count Enter also when you are specifying it in max1)

Purpose: Output a character to display (STD OUT)

Program Segment

```
MOV     DL, 'A'
MOV     AH, 02h
INT     21h
```

After Interrupt is executed character 'A' will be displayed on the screen.

Purpose: Output a string on display (STDOUT)

Program

```
.data
str1    db      'HELLO $'       ; all strings must terminate with '$' ASCII value (24h)
.code
.startup
        lea     dx, str1
        mov     ah, 09h
        int     21h
```

When interrupt is executed the string "HELLO" will be displayed on screen.

Remove the '$' sign. What happens?

Task1: Write an ALP that will take in a string of maximum 20 characters from user and display it on the next line on the screen (ASCII equivalent for newline is 0Dh (enter) followed by 0Ah (next line))

Task2: Write an ALP that does the following

(1) Display the string “Enter User Name” and goes to the next line
(2) Takes in the user entered string compares with user name value already stored in memory
(3) If there is no match it should exit.
(4) If there is a match it should display the string “Enter Password” and goes to next line
(5) Takes in password entered by the user and compares with password already stored in memory
(6) If there is no match it should exit’
(7) If there is a match it should display “Hello *Username*”

Note:

While the username is being entered it can be displayed but when password is being entered user pressed key should be displayed instead it should display “*” for every key pressed.

The user name size is fixed to 10 characters and password to 8 characters.