

Laboratory Manual

CS/EEE/ECE/INSTR F241 Microprocessor

Programming & Interfacing

DOS INTERRUPTS – BASIC KEYBOARD & DISPLAY

CONSOLE INTERRUPTS

Meetha.V.Shenoy

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,

PILANI – Pilani CAMPUS

LAB 4 – DOS INTERRUPTS – BASIC KEYBOARD & DISPLAY CONSOLE INTERRUPTS

4.1. Introduction to DOS & BIOS Interrupts

The Basic Input/Output System (BIOS) 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? This lab exercise will provide insights to these questions.

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 use INT 21 H (DOS Interrupts) – Here, we use INT 21H to access the value of the key pressed and display characters.

Note: In this lab, for executing instructions one at a time, you will use the 't' command. However, for executing INT 21H instruction, you will use the 'p' command [proceed command] instead of the 't' command.

Command	Function	Effect on INT Instructions
T (Trace)	Executes the next single instruction , stepping inside procedure calls or interrupts.	Steps inside the interrupt routine, showing each instruction within the interrupt handler.
P (Proceed)	Executes the next instruction normally , but skips over procedure calls and interrupts.	Executes the interrupt but does not step into its internal instructions.

4.2 Keyboard Interrupts

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

Program Snippet

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

Outcome expected: The user will get a prompt to enter a character/key via the keyboard. ASCII value corresponding to the character entered will be stored in AL register. The steps to execute are given below.

Full program and steps to execute:

```
.model tiny
.data
.code
```

```
.startup
    MOV AH,01H
    int 21h
.exit
End
```

Use “ml filename.asm” command to assemble the code and then use debugx to debug the code. Note that INT 21H should be executed using ‘p’ command instead of ‘t’ command. ‘p’ is proceed command.

```
D:\>debugx i1.com
-t
AX=01FF BX=0000 CX=0008 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0863 ES=0863 SS=0863 CS=0863 IP=0102 NU UP EI PL ZR NA PE NC
0863:0102 CD21          INT     21
-p
1 AX=0131 BX=0000 CX=0008 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0863 ES=0863 SS=0863 CS=0863 IP=0104 NU UP EI PL ZR NA PE NC
0863:0104 B44C          MOV     AH,4C
```

You can observe that when you enter p command, the code provides you with a prompt to enter the character. If ‘1’ is entered, AL gets filled with 31h (ASCII value corresponding to ‘1’).

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 debugx and the ASCII key you press will be stored in AL register. In this case, the entered character will not be visible on the screen when you type it

Purpose: Input a string from keyboard (STDIN)

Program Snippet

```
.data
max1    db      10              ; 10 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 9 as the 10th will be Enter key)
inp1     db      10 dup(0)      ; Reserve 10 locations for input string
.code
.startup
    LEA    DX,max1
    MOV    AH, 0Ah
    INT    21

.exit
end
```

After the execution of interrupt, act 1 will contain the number of characters read, and the characters entered will be stored at inp1. The characters will be terminated by a carriage return or enter (ASCII code 0Dh).

Note: while storing count at max1, the count for carriage return or enter should be included. (Note: act1 will have the actual number of characters entered, excluding enter key)

```
D:\>debugx i3.com
-u 100
0863:0100 8D160C01      LEA     DX,[010C]
0863:0104 B40A          MOV     AH,0A
0863:0106 CD21          INT     21
0863:0108 B44C          MOV     AH,4C
0863:010A CD21          INT     21
```

Program is stored at ds:010c

```
-d ds:010c
0863:0100      -                0A 00 00 00      ....
0863:0110  00 00 00 00 00 00 00 00-59 2B F6 8B 07 D4 58 8E  ....Y+...X.
0863:0120  C2 26 89 4C 2C 50 FF 76-06 B8 F9 09 BA 2D 15 52  .&.L,P.v...-.R
0863:0130  50 1E 07 9A 42 07 5A 02-8B E5 5D CB 55 3B EC FF  P...B.Z...l.U...
0863:0140  76 06 9A 2D 14 EF 06 8B-E5 5D CB 55 8B EC 83 3E  v...-...l.U...>
0863:0150  06 F7 03 73 43 8A 46 0B-3A F4 F0 FF 76 07 8A 0F  H...F...B...
```

max1

After entering string as 'hello1234'

```
-d ds:010c
0863:0100      -                0A 09 68 65      ..he
0863:0110  6C 6C 6F 31 32 33 34 0D-59 2B F6 8B 0E D4 58 8E  llo1234.Y+...X.
0863:0120  C2 26 89 4C 2C 50 FF 76-06 B8 F9 09 BA 2D 15 52  .&.L,P.v...-.R
```

act1 entered string

Purpose: Output a character to display (STD OUT)

Program Segment

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

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

Purpose: Output a string on display (STDOUT)

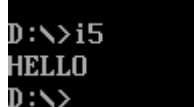
Program Snippet

```
.data
str1    db          'HELLO $'      ; all strings must terminate with '$' ASCII value (24h)
.code
```

.startup

```
    lea        dx, str1
    mov        ah, 09h
    int        21h
```

When the program is executed the string “HELLO” will be displayed on screen. Note: if your executable created using the ‘ml i5. asm’ is ‘i5.com’, you can just type ‘i5’ to run the code instead of the debugx filename.com



```
D:\>i5
HELLO
D:\>
```

Remove the ‘\$’ sign from the str1 declaration, assemble the code again and execute. What happens? Use of ‘\$’ is hence critical.

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.

Hint: Follow the steps

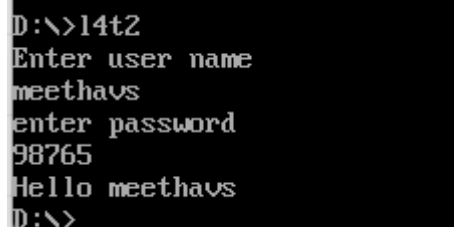
- Use instructions to Input a string from keyboard
- Use instructions to Output a character to display – Display 0AH (Note 0DH is the ascii equivalent of next line character)
- Use instructions to Output a character to display – Display 0DH (Note 0DH is the ascii equivalent of enter character)
- Use instructions to Output a string on display

Task2: Write an ALP that does the following

- (1) Display the string “Enter User Name” and wait for the user to enter the same on the next line
- (2) Takes in the user entered string . The string may be visible on the screen as and when it is entered.
- (3) Compare the user name with the user name stored in memory in the data segment
- (4) If there is no match, program should exit.
- (5) If there is a match, the program should display the string “Enter Password” and allow the user to enter the password in the next line. The string may be visible on the screen as and when it is entered.
- (6) Takes in password entered by the user and compares with the password already stored in memory
- (7) If there is no match, the program should exit.
- (8) If there is a match it should display “Hello *Username*”

Note: User name is always 8 characters. Password is always 5 digits.

Sample output when the entered user name and password matches with the one entered in memory



```
D:\>14t2
Enter user name
meethavs
enter password
98765
Hello meethavs
D:\>
```

Sample output when the entered password doesn't match with the one saved in memory

```
D:\>l4t2
Enter user name
meethavs
enter password
87654
D:\>
```

Task3: Write an ALP that does the following.

Repeat Task 2 with the following modification.

In step 5, When password is being entered, the entered password should not be displayed, instead it should display '*' for every key pressed.

Sample output when the entered user name and password matches with the one entered in memory

```
D:\>l4t3
Enter user name
meethavs
enter password
*****
Hello meethavs
D:\>
```