## Experiment No 1

**Aim: Introduction to Assembly Language Programming:**

To write assembly language program using TASM assembler certain DOS interrupt and its functions are used:

**DOS INT 21H**

INT 21H is provided by DOS, and it is stored in DRAM when the operating system is loaded. The user can invoke this interrupt to perform several useful functions, like inputting data from keyboard, and outputting data to monitor. These interrupt subroutines may be used by issuing a software interrupt call (**INT *type***). The user will have to identify which function is being used by setting the AH register to a specific value. Other registers may also need to be modified. Following are the descriptions of several of the most common INT 21H functions.

1) **INT 21H Function 01H: Inputting a single character from the keyboard with echo.**
   AH = 01H
   AL = inputted ASCII character code
   Code example to input a single character from keyboard and to echo it to the display.

   ```
   MOV AH,01H
   INT 21H
   ```

2) **INT 21H Function 02H: Outputting a single character to the monitor.**
   AH = 02H
   DL = ASCII character code to be displayed
   Code example to output a single character to the monitor.

   ```
   MOV AH,02H
   INT 21H
   ```

3) **INT 21H Function 09H: Outputting a string terminated with $ to the monitor.**
   AH = 09H
   DX = String address can be calculated using OFFSET assembler directive.msg is a string variable which is initialized with string in declared data segment.
   msg variable is declared in data segment as follows:

   ```
   msg DB 0DH, 0AH,"Enter a number1 $"
   ```

   Where
   DB is another assembler directive stands for Define byte
   0DH is hex code for enter key

0AH is hex code for line feed

0DH and 0AH together will create a new line on the screen after displaying current msg for the next line to be displayed.

**Note:**

1. **At the end of string $ is compulsory while declaring.**
2. **Assembly Language Programs are not case sensitive.**

Code example to output a string terminated with $ to the monitor

```
MOV DX, OFFSET  msg
MOV AH, 09H
INT 21H
```

4) **INT 21H Function 4CH: Terminate a process ( EXIT ).**

AH = 4CH

AL = binary return code

This interrupt terminates a process and returns control to DOS or the parent process.

Code example to terminate a program :

```
MOV AH, 4CH
INT 21H
```

**ASSEMBLER DIRECTIVE:**

An assembler is a program used to convert an assembly language program into the equivalent machine code modules. The assembler decides the address of each label and substitutes the values for each of the constants and variables. It then forms the machine code for mnemonics and data in assembly language program.

Assembler directives help the assembler to correctly understand assembly language programs to prepare the codes. Commonly used assembler directives are DB, DD, DW, DUP, ASSUME, BYTE, SEGMENT, MACRO, PROC, OFFSET, NEAR, FAR, EQU, STRUC, PTR, END, ENDM, ENDP etc. Some directives generate and store information in the memory, while others do not.

1) **DB :-** Define byte directive stores bytes of data in memory.
2) **BYTE PTR :-** This directive indicates the size of data referenced by pointer.
3) **SEGMENT :-** This directive is to indicate the start of the segment.
4) **DUP (Duplicate) :-** The DUP directive reserves memory locations given by the number preceding it, but stores no specific values in any of these locations.

5) **ASSUME : -** The ASSUME statement is only used with full segment definitions. This statement tells the assembler what names have been chosen for the code, data, extra and stack segments.
6) **EQU : -** The equate directive equates a numeric ASCII or label to another label.
7) **ORG : -** The ORG (origin) statement changes the starting offset address in a segment.
8) **PROC and ENDP : -** The PROC and ENDP directives indicate start and end of a procedure (Sub routine). Both the PROC and ENDP directives require a label to indicate the name of the procedure. The PROC directive, must also be followed with the NEAR or FAR. A NEAR procedure is one that resides in the same code segment as the program. A FAR procedure may reside at any location in the memory system.