19 Interrupt Programming

“Think before you speak.”

Interrupt is the one which temporarily suspends the execution of the current program and executes a specific subroutine called interrupt routine and then resumes the execution of actual program. Many people think that the interrupt instruction 'INT' is one of the "basic" instructions in assembly language. But it is not so. The 'INT' instruction just calls or invokes a specific routine i.e., interrupt routine.

19.1 Logical outline of interrupt routine

The following code shows the logical outline of an interrupt routine. (Please understand that it is only a prototype)

```
int10h( REGISTER AX, REGISTER BX, ...... )
{
   switch( AH )         /* AH holds function number */
   {
      case 0x0:
                switch( AL )  /* AL holds sub function number */
                {
                  case 0x0:
                              MOV ....
                              INC ....
                              break;
                  case 0x1:
                              :
                              break;
                }
                break;
      case 0x1:
                if(BX == 0)
                  {
                     MOV ....
                      :
                  }
                break;
      case 0x2:
                :
                break;
}
```

Here, you see that the behavior of the interrupt routine is determined by the argument that passes through (Some book authors use the term *input values instead of argument*. But professional programmers use the term argument). The value passed through the register AH is referred as function value. In special cases, value is also passed through AL register to the sub-function. Sometimes we would also pass values through other registers.

Some interrupt routines don't take any argument, which means we don't need to pass value through registers. For example, the interrupt for Print Screen int 5h doesn't take any argument. The prototype of int 5h hence looks like:

```
int5h( void )
{
   MOV ...
      :
      :
}
```

Usually interrupt numbers, function numbers and sub-function numbers are represented in hexadecimal rather that in decimal.

19.2 Interrupt Classification

Each and every motherboard must have a chip containing software, which is known as BIOS or ROM BIOS. Basic Input/Output system (BIOS) is a collection of programs burned (or embedded) in an EPROM (Erasable Programmable Read Only Memory) or EEPROM (Electrically Erasable ROM). We can call these programs by what is known as *interrupts*. By the way you should know that BIOS programs are not much compatible, because they are written typically for the hardware and they manage the hardware. (Different machines may use different hardware). Usually most of the BIOS functions are compatible.

Operating System is nothing but program that operates computer. It is actually an extension of BIOS. Thus Disk Operating System (DOS) functions and BIOS functions collectively interact with the hardware. Besides interacting with hardware, DOS programs preside more useful functions such as file maintenance (create file, delete file, rename file, etc). These functions can be called by interrupts. Experts find that DOS programs are good for 'DISK' related functions, than 'Input / Output' related functions. Yes, DOS also has got *few* 'Input / Output' related functions. But these 'Input / Output' related functions are not much used by programmers. They prefer BIOS functions for 'Input / Output' related functions. There is a drawback with DOS functions; it is not re-entrant (where as BIOS functions are re-entrant). If a routine can be called again before it is finished, it is said to be re-entrant. TSR programmers very often get suffered by DOS's re-entrancy problem.

19.3 Programming with interrupts

We have seen that we can call DOS functions or BIOS functions with what is known as interrupts. Turbo C provides various ways to send arguments and to generate interrupts. Let's write a simple function `GetVideoMode( )` to get the current video mode with various styles.

To get the current video mode, we have to generate int 10h and we should pass 0Fh in AH register as an argument. After generating interrupts, current video mode is stored in AL register.

19.3.1 Inline Assembly Style

```
typedef char BYTE;
BYTE GetVideoMode(  void )
{
   asm {
         mov ah, 0Fh;
         int 10h;
      }
   /* AL holds current video mode and is returned */
} /*--GetVideoMode( )-------*/
```

19.3.2 Pure Assembly Style

We can also write a pure assembly file (getvid.asm) and assemble the file with TASM as

```
C:\WAR>TASM -mx getvid
```

Now we will get getvid.obj. We can link this obj file with the main program.

```
; File name: Getvid.asm
.MODEL small, C
.CODE
     PUBLIC GetVideoMode
GetVideoMode PROC NEAR
     MOV AH, 0Fh
     INT 10h        ; AL register holds current video mode
     XOR AH, AH   ; Set AH register to 0
                  ; Now, AX holds value of AL
     RET          ; value in AX get returned
GetVideoMode ENDP
END
```

19.3.3 geninterrupt( ) style

```
typedef char BYTE;
BYTE GetVideoMode( void )
{
   _AH = 0x0F;
```

```
      geninterrupt( 0x10 );
      return(_AL);
   } /*--GetVideoMode( )-------*/
```

19.3.4 int86( ) style

```
   BYTE GetVideoMode( void )
   {
      union REGS inregs, outregs;
      inregs.h.ah = 0x0F;
      int86( 0x10, &inregs, &outregs );
      return( outregs.h.al );
   } /*--GetVideoMode( )-------*/
```

The function related to `int86( )` are `int86x( )`, `intdos( )` & `intdosx( )`. And those functions return the value of AX after completion of the interrupt. If an error occurs, carry flag is set to 1 and `_doserrno` is also set to error code.

19.3.5 intr( ) style

```
   BYTE GetVideoMode( void )
   {
      struct REGPACK regs;
      regs.r_ax = 0x0F00;
      intr( 0x10, &regs );
      return( (BYTE)regs.r_ax );
   } /*--GetVideoMode( )-------*/
```

Here you have to note that `intr( )` functions doesn't return anything, there is no way to represent AL or AH register separately.

19.3.6 Benchmarking

We can find that the inline assembly style and pure assembly style are faster than any other above methods. Big software companies use "Pure Assembly Style". They create library file with assembly language and link them wherever necessary. Inline assembly is my choice, because it provides more readability, C style usage and flexibility. For example in C, we can directly enter octal or hexadecimal or decimal number as

```
         int a = \101 ; /* Octal   */
         int b = \x65 ; /* Hexa    */
         int c = 65   ; /* decimal */
```

But we cannot directly enter binary values in C (But it is possible in Assembly!). One solution for this is to use `strtol( )` as:

```
   int a;
   char str[] = "0000010";  /* binary */
```

```
        char *endptr;
        /* radix should be 2 for binary in strol... */
        a = strtol( str, &endptr, 2 );
```

Fortunately inline style provides more flexibility and an easy way for entering binary values:

```
      asm MOV AX, 00000010b;      (or)  asm {
      a = _AX;                                 MOV AX, 00000010b ;
                                               MOV a, AX ;
                                            }
```

The suffix 'b' tells that it is a binary number.

That's why I prefer the flexible inline style. But if you are a beginner and if you don't know much of assembly, I suggest you to use `int86( )` style as it provides good error handling mechanism. You can even use other styles, if you are comfortable with them!

19.4 Myth & Mistakes

Q: *"Use of standard library functions increase the size of the EXE file. But this interrupt function doesn't increase the size of the EXE file". Is this statement true?*

A: No. This statement has no sense at all. This myth is introduced in Indian Programming World by few book authors. TC's library functions also use interrupts and it was also written by "Programmers". The only difference you can find between interrupt programming and using compiler's library is flexibility i.e., our own functions will be more convenient as it is written by us.

Q: *Can I use standard library's `gotoxy()` ?*

A: The standard library according to ANSI standard doesn't have `gotoxy( )`. `gotoxy( )` is provided by Turbo C and you can use it.

Exercises

1. Write a program that find out the life of battery found on your motherboard.

Suggested Projects

1. Write diagnostic software that finds the status of your peripherals and motherboard.