Using Assembly Language with C/C++

Introduction

The assembly language portion usually solves tasks (difficult or inefficient to accomplish in C/C++) that often include control software for peripheral interfaces and driver programs that use interrupts.

Using Assembly Language with C++ for 16-Bit DOS Applications

The performance of a program often depends on the incorporation of assembly language sequences to speed its execution. The assembly language is also used for I/O operations in embedded systems.

To build a 16-bit **DOS** application, you will need the legacy 16-bit compiler usually found in the directory of the Windows DDK. The compiler is **CL.EXE** and the 16-bit linker program is LINK.EXE, both located in the directory or folder listed. Compilation and linking must be performed at the command line because there is no visual interface or editor provided with the compiler and linker. Programs are generated using either Notepad or DOS Edit.

Basic Rules and Simple Programs

Before assembly language code can be placed in a C/C++ program, some rules must be learned.

- All the assembly code is placed in the **_asm** block.
- It is also extremely important to use lowercase characters for any inline assembly code. If you use uppercase, you will find that some of the assembly language commands and registers are reserved or defined words in C/C++ language.
- Labels can be used, just as in normal assembly program.
- The semicolon adds comments to the listing in the _asm block, just as with the normal assembler.
- The AX, BX, CX, DX, and ES registers are never used by Microsoft C/C++. These registers, which might be considered **scratchpad** registers, are available to use with assembly language. If you wish to use any of the other registers, make sure that you save them with a PUSH before they are used and restore them with a POP afterwards. If you fail to save the registers used by a program, the program may not function correctly and can crash the computer.

Example -1 This program reads one character from the console keyboard, and then filters it through assembly language so that only the numbers 0 through 9 are sent back to the video display.

EXAMPLE –1

```
//Accepts and displays one character of 1 through 9,
//all others are ignored.
void main(void)
{
      _asm
      {
            mov ah,8 ;read key no echo
            int 21h

            cmp al,‘0’ ;filter key code
            jb big

            cmp al,‘9’
            ja big
            mov dl,al ;echo 0 - 9

            mov ah,2
            int 21h
            big:
      }
}
```

To **compile the program**, start the Command Prompt program located in the Start Menu under Accessories. Change the path to where you have your Windows DDK. Make sure you saved the program in the same path and use the extension .c with the file name. To compile the program, type **CL /G3 filename.c**>. This will generate the .exe file for the program. (See Table –1 for a list of the /G compiler switches.) Any errors that appear are ignored by pressing the Enter key. When the program is executed, you will only see a number echoed back to the DOS screen.

Compiler Switch	*Function*
/G1	Selects the 8088/8086
/G2	Selects the 80188/80186/80286
/G3	Selects the 80386
/G4	Selects the 80486
/G5	Selects the Pentium
/G6	Selects the Pentium Pro–Pentium 4

Note: The 32-bit C++ compiler does not recognize /G1 or /G2.

TABLE –1 Compiler (16-bit) G options.

Example –2 shows how to use variables from C with a short assembly language program. In this example, program itself performs the operation X + Y = Z, where X and Y are two one-digit numbers, and Z is the result.

EXAMPLE –2

```
void main(void)
{
      char a, b;
      _asm
      {
            mov ah,1 ;read first digit
            int 21h
            mov a,al

            mov ah,1 ;read a + sign
            int 21h

            cmp al,'+'
            jne end1 ;if not plus

            mov ah,1
            int 21h ;read second number
            mov b,al

            mov dl,'=';display =
            mov ah,2
```

```
           int 21h

           mov ah,0
           mov al,a ;generate sum
           add al,b
           aaa ;ASCII adjust for addition
           add ax,3030h
           cmp ah,'0'
           je down

           push ax
           mov dl,ah ;display 10's position
           mov ah,2
           int 21h
           pop ax

     down: mov dl,al ;display units position
           mov ah,2
           int 21h
           end1:
     }
}
```

What Cannot Be Used from MASM Inside an _asm Block

- The inline assembler does not include the conditional commands from MASM, nor does it include the MACRO feature found in the assembler.
- Data allocation with the inline assembler is handled by C instead of by using DB, DW, DD, etc.

Using Character Strings

Example –3 illustrates a simple program that uses a character string defined with C and displays it so that each word is listed on a separate line. The WHILE statement repeats the assembly language commands until the null (00H) is discovered at the end of the character string. If the null is not discovered, the assembly language instructions display a character from the string unless a space is located. For each space, the program displays a carriage return/line feed combination. This causes each word in the string to be displayed on a separate line.

EXAMPLE–3

```
// Example that displays showing one word per line
void main(void)
{
     char strings[] = "This is my first test application using _asm.\n";
     int sc = -1;
     while (strings[sc++] != 0)
     {
          _asm
          {
               push si
               mov si,sc ;get pointer
               mov dl,strings[si] ;get character

               cmp dl,' ' ;if not space
               jne next

               mov dl,10;display new line
```

```
                mov ah,2
                int 21h
                mov dl,13

           next: mov ah,2 ;display character
                int 21h
                pop si
           }
      }
}
```

Example–4 illustrates a program that creates a procedure displaying a character string. This procedure is called each time that a string is displayed in the program. Note that this program displays one string on each line.

EXAMPLE–4

```
// A program illustrating an assembly language procedure that
// displays C language character strings
char string1[] = “This is my first test program using _asm.”;
char string2[] = “This is the second line in this program.”;
char string3[] = “This is the third.”;
void main(void)
{
      Str (string1);
      Str (string2);
      Str (string3);
}
Str (char *string_adr)
{
      _asm
      {
           mov bx,string_adr ;get address of string
           mov ah,2

       top: mov dl,[bx]
           inc bx
           cmp al,0 ;if null
           je bot
           int 21h ;display character
           jmp top

       bot: mov dl,13 ;display CR + LF
           int 21h
           mov dl,10
           int 21h
      }
}
```

Using Data Structures

Example–5 illustrates a short program that uses a data structure to store names, ages, and salaries. The program then displays each of the entries by using a few assembly language procedures. The ***Crlf procedure*** displays a carriage return/line feed combination. The ***Numb procedure*** displays the integer.

EXAMPLE–5

UNIT 4

```
// Program illustrating an assembly language procedure that displays the
//contents of a C data structure.

typedef struct records
{
      char first_name[16];
      char last_name[16];
      int age;
      int salary;
} RECORD;

// Fill some records
RECORD record[4] =
{ {“Bill” ,”Boyd” , 56, 23000},
{“Page”, “Turner”, 32, 34000},
{“Bull”, “Dozer”, 39. 22000},
{“Hy”, “Society”, 48, 62000}
};
//Program
void main(void)
{
      int pnt = -1;
      while (pnt++ < 3)
      {
            Str(record[pnt].last_name);
            Str(record[pnt].first_name);
            Numb(record[pnt].age);
            Numb(record[pnt].salary);
            Crlf();
      }
}
Str (char *string_adr[])
{
      _asm
      {
            mov bx,string_adr
            mov ah,2
       top: mov dl,[bx]
            inc bx
            cmp al,0
            je bot
            int 21h
            jmp top
      bot: mov al,20h
            int 21h
      }
}

Crlf()
{
      _asm
      {
            mov ah,2
            mov dl,13
            int 21h
            mov dl,10
            int 21h
      }
}
Numb (int temp)
{
      _asm
      {
            mov ax,temp
            mov bx,10
            push bx
       L1: mov dx,0
            div bx
            push dx
            cmp ax,0
            jne L1
            L2: pop dx
            cmp dl,bl
            je L3
            mov ah,2
            add dl,30h
            int 21h
            jmp L2
        L3: mov dl,20h
            int 21h
      }
}
```

An Example of a Mixed-Language Program

Example –6 shows how the program can do some operations in assembly language and some in C language. Here, the only assembly language portions of the program are the Dispn procedure that displays an integer and the Readnum procedure, which reads an integer. Also, the program functions correctly only if the result is positive and less than 64K.

EXAMPLE–6 ***/** IMPORTANT **/***

```
/* A program that functions as a simple calculator to perform addition,
subtraction, multiplication, and division. The format is X <oper> Y =.*/

int temp;
void main(void)
{

      int temp1, oper;
      while (1)
      {
            oper = Readnum(); //get first number and operation
            temp1 = temp;
            if ( Readnum() == '=' ) //get second number
            {
                  switch (oper)
                  {
                      case '+':  temp = temp1 + temp;
                                 break;
                      case '-':  temp = temp1 - temp;
                                 break;
                      case '/':  temp = temp1 / temp;
                                 break;
                      case '*':  temp = temp1 * temp;
                                 break;
                 }
                 Dispn(temp); //display result
            }
            else
                 break;
      }
}

int Readnum()
{
      int a;
      temp = 0;
      _asm
      {
            Readnum1:  mov ah,1
                       int 21h
                       cmp al,30h
                       jb Readnum2
                       cmp al,39h
                       ja Readnum2
                       sub al,30h
                       shl temp,1
                       mov bx,temp
                       shl temp,2
                       add temp,bx
                       add byte ptr temp,al
                       adc byte ptr temp+1,0
                       jmp Readnum1
            Readnum2:  mov ah,0
                       mov a,ax
      }
      return a;
}
Dispn (int DispnTemp)
{
      _asm
      {
                 mov ax,DispnTemp
                 mov bx,10
                 push bx
         Dispn1:mov dx,0
                 div bx
                 push dx
                 cmp ax,0
                 jne Dispn1
```

```
          Dispn2:pop dx
                 cmp dl,bl
                 je Dispn3
                 add dl,30h
                 mov ah,2
                 int 21h
                 jmp Dispn2
          Dispn3:mov dl,13
                 int 21h
                 mov dl,10
                 int 21h
     }
}
```

COMPARING 16-BIT AND 32-BIT APPLICATIONS

- The 32-bit applications are written using Microsoft Visual C/C++ Express for Windows and the 16-bit applications are written using Microsoft C++ for DOS.
- The main difference is that Visual C/C++ Express for Windows (32-bit application) is more common today, rather than 16-bit application
- Visual C/C++ Express (32-bit application) cannot easily call DOS functions such as INT 21H.
- The embedded applications that do not require a visual interface be written in 16-bit C or C++, and applications that incorporate Microsoft Windows use 32-bit Visual C/C++ Express for Windows.
- A 32-bit application is written by using any of the 32-bit registers, and the memory space is essentially limited to 2GB for Windows.
- Embedded applications (16-bit application) use direct assembly language instructions to access I/O devices in an embedded system. In the Visual interface (32-bit application), all I/O is handled by the Windows operating system framework.

Console applications in WIN32 run in native mode, which allow assembly language to be included in the program without anything other than the _asm keyword***. Windows forms applications*** are more challenging because they operate in the managed mode, which does not run in the native mode of the microprocessor. Managed applications operate in a pseudo mode that does not generate native code.