Lecture # 30
High-Level Language Interface

- General and Calling Conventions
- Inline assembly code

Inline Assembly Code

- **Inline Assembly Code**

  Assembly language source code that is inserted directly into high-level language programs

- **__asm Directive in Microsoft Visual C++ :**
- **Inline assembly code for Microsoft visual C++ running in 32-bit protected mode using flat memory model**

  Main advantage is simplicity as there are no external linking issues

- **Suffers from lack of portability**

- **The allowed features when writing inline assembly code**
- **Any instruction for 80x86 instruction set is supported**
- **Register names may be used as operands**
- **Code labels and variables declared outside the __asm block are supported**
- **Numeric constants can be used**

Inline Assembly Code Syntax

- **The PTR operator may be used**
- **EVEN and align directives may be used**

➢ ***The __asm directive***

Marks the beginning of a block of assembly language statements or a single statement

__asm *statement*

```
__asm{                         ; may use either ; or // or */ for
                               ; comments but use C/C++ syntax
    statement_1                // preferably for comments
    statement_2
        …
   statement n
        }
```

Inline Assembly Code

- **Limitations of inline assembly code**
  - **Data definition directives cannot be used**
  - **Assembler operators other than PTR cannot be used**
  - **Cannot reference macro directives**
  - **Cannot reference segments by name**
- ***Register Values***

  Cannot make any assumptions about register values at the beginning of the asm block

  Can modify EAX, EBX, ECX and EDX registers in the line code

- **Example in-line assembly code**

  File encryption example
  Procedure Call Overhead

```
void TranslateBuffer( char * buf,
     unsigned count, unsigned char eChar )
{
    __asm {
          mov   esi,buf
          mov   ecx,count
          mov   al,eChar
    L1:
          xor   [esi],al
          inc   esi
          loop  L1
    }     // asm
}
```

```
int main( int argcount, char * args[] )
{
    // Read input and output files from the command line.
    if( argcount < 3 ) {
          cout << "Usage: encode infile outfile" << endl;
          return -1;
    }
    const int BUFSIZE = 2000;
    char buffer[BUFSIZE];
    unsigned int count;         // character count
    unsigned char encryptCode;
    cout << "Encryption code [0-255]? ";
    cin >> encryptCode;
    ifstream infile( args[1], ios::binary );
    ofstream outfile( args[2], ios::binary );
    cout << "Reading" << args[1] << "and creating"
          << args[2] << endl;
    while (!infile.eof() )
    {
          infile.read(buffer, BUFSIZE);
          count = infile.gcount();
          TranslateBuffer(buffer, count, encryptCode);
          outfile.write(buffer, count);
    }
    return 0;
}
```

```
char array1 [5] = {2, -4, 56, -87, 35};
char sum1, sum2, sum3, sum4, sum5;

 //switch to assembly
       _asm
       {
          MOV    AL, array1[0]      ;+2
          ADD    AL, array1[1]      ;-4
          MOV    sum1, AL           ;-2
 ;-----------------------------------
          MOV    AL, array1[1]      ;-4
          ADD    AL, array1[3]      ;-87
          MOV    sum2, AL           ;-91
 ;-----------------------------------
          MOV    AL, array1[2]      ;+56
          ADD    AL, array1[4]      ;+35
          MOV    sum3, AL           ;+91
 ;-----------------------------------
          MOV    AL, array1[3]      ;-87
          ADD    AL, array1[2]      ;+56
          MOV    sum4, AL           ;-31
 ;-----------------------------------
          MOV    AL, array1[0]      ;+2
          ADD    AL, array1[4]      ;+35
          MOV    sum5, AL           ;+37
       }
```

```
printf ("Sum1 = %d\n", sum1);
printf ("Sum2 = %d\n", sum2);
printf ("Sum3 = %d\n", sum3);
printf ("Sum4 = %d\n", sum4);
printf ("Sum5 = %d\n\n", sum5);
```

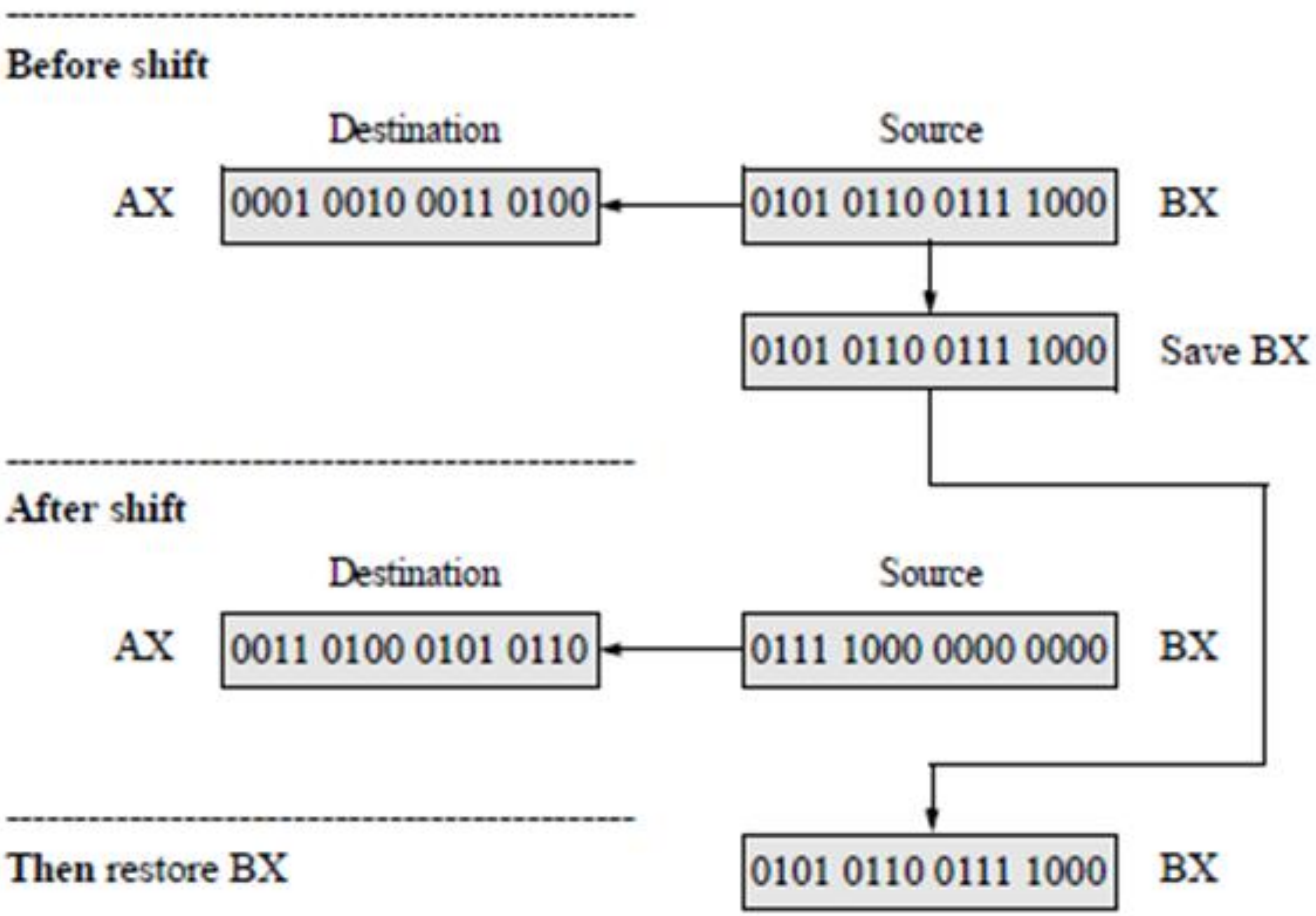
```
Sum1 = -2
Sum2 = -91
Sum3 = 91
Sum4 = -31
Sum5 = 37
```

```
int main (void)

{
//define variables
   unsigned short src_opnd, dst_opnd, src_rslt, dst_rslt;

   printf ("Enter two 4-digit hex numbers - src, dst: \n");
   scanf ("%X %X", &src_opnd, &dst_opnd);

//switch to assembly
      _asm
      {
         MOV   BX, src_opnd
         MOV   AX, dst_opnd

         SHLD  AX, BX, 8   ;shift AX:BX left 8 bits

         MOV   src_rslt, BX
         MOV   dst_rslt, AX
      }

   printf ("\nSource result = %X\n
            Destination result = %X\n\n",
            src_rslt, dst_rslt);

   return 0;
}
```

```
Enter two 4-digit hex numbers - src, dst:
1234 5678

Source result = 1234
Destination result = 7812

Press any key to continue . . . _
------------------------------------------
Enter two 4-digit hex numbers - src, dst:
12AB CDEF

Source result = 12AB
Destination result = EF12

Press any key to continue . . . _
```

Let AX = 1234H
Let BX = 5678H

Then do SHLD AX, BX, 8


Before shift
Destination
Source
AX
0001 0010 0011 0100
0101 0110 0111 1000
BX
0101 0110 0111 1000
Save BX
After shift
Destination
Source
AX
0011 0100 0101 0110
0111 1000 0000 0000
BX
Then restore BX
0101 0110 0111 1000
BX