

Lab Session 12

High Level Language Interface

Objectives

- General Conventions, Model Directive
- Implementing Inline Assembly Code

.Model Directive

.MODEL directive determines

- memory model type
- procedure naming scheme
- parameter passing convention

```
.MODEL memorymodel [,modeloptions]
```

Memory Model

Model	Description
Tiny	A single segment, containing both code and data. This model is used by programs having a .com extension in their filenames.
Small	One code segment and one data segment. All code and data are near, by default.
Medium	Multiple code segments and a single data segment.
Compact	One code segment and multiple data segments.
Large	Multiple code and data segments.
Huge	Same as the large model, except that individual data items may be larger than a single segment.
Flat	Protected mode. Uses 32-bit offsets for code and data. All data and code (including system resources) are in a single 32-bit segment.

Memory Options

Language specifier -> determines calling and naming conventions for procedures and public symbols

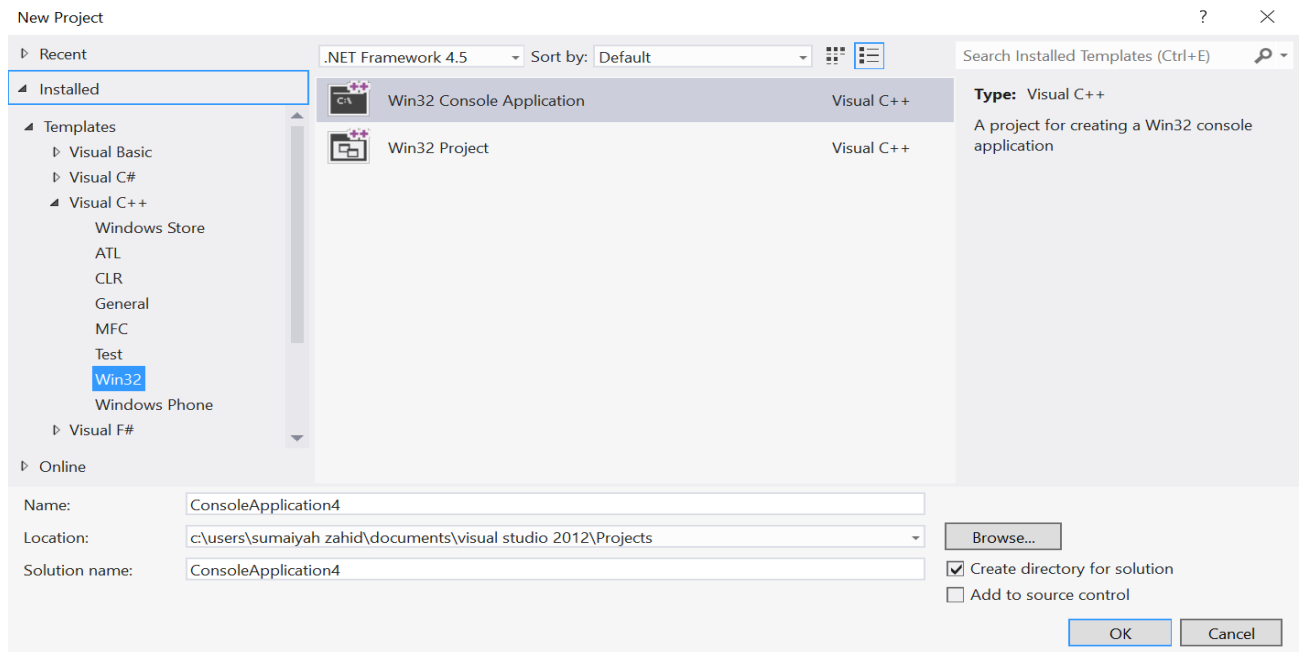
Stack distance -> can be NEARSTACK (the default) or FARSTACK

We mostly uses `.model flat, STDCALL`

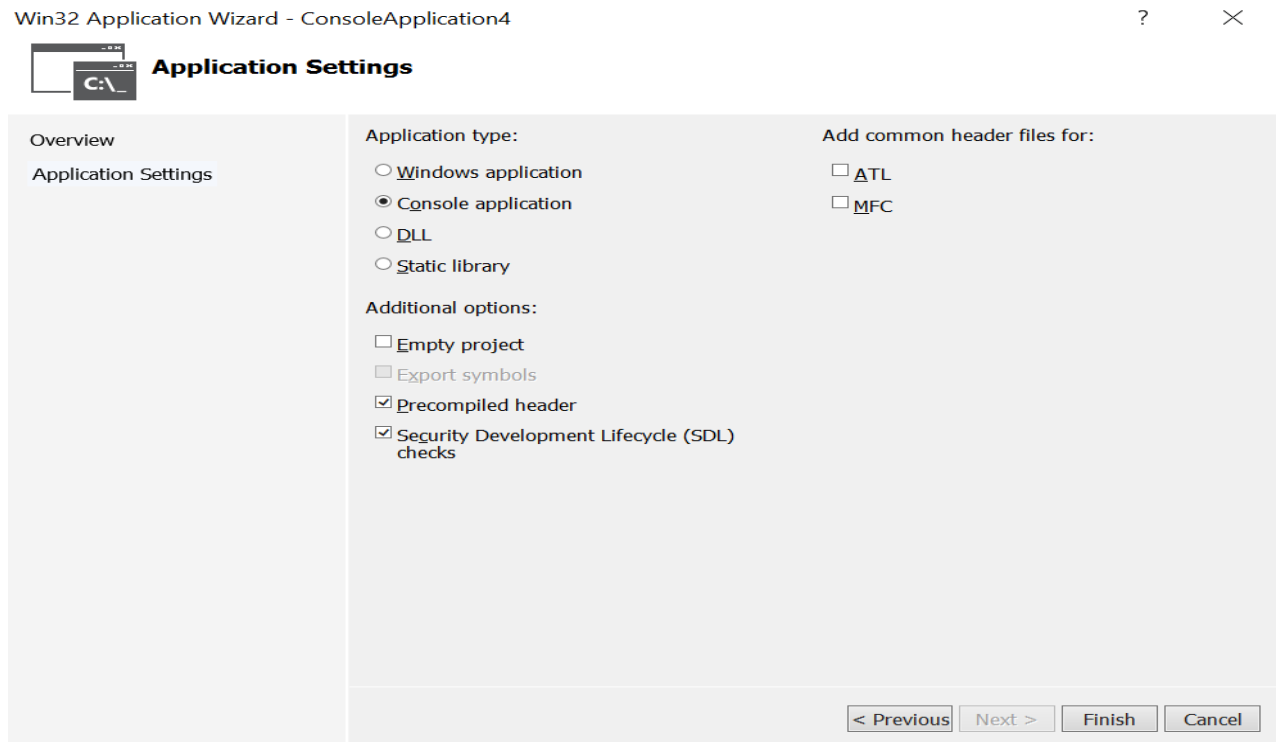
STDCALL is the language specifier used when calling MS-Windows functions.

Steps to follow

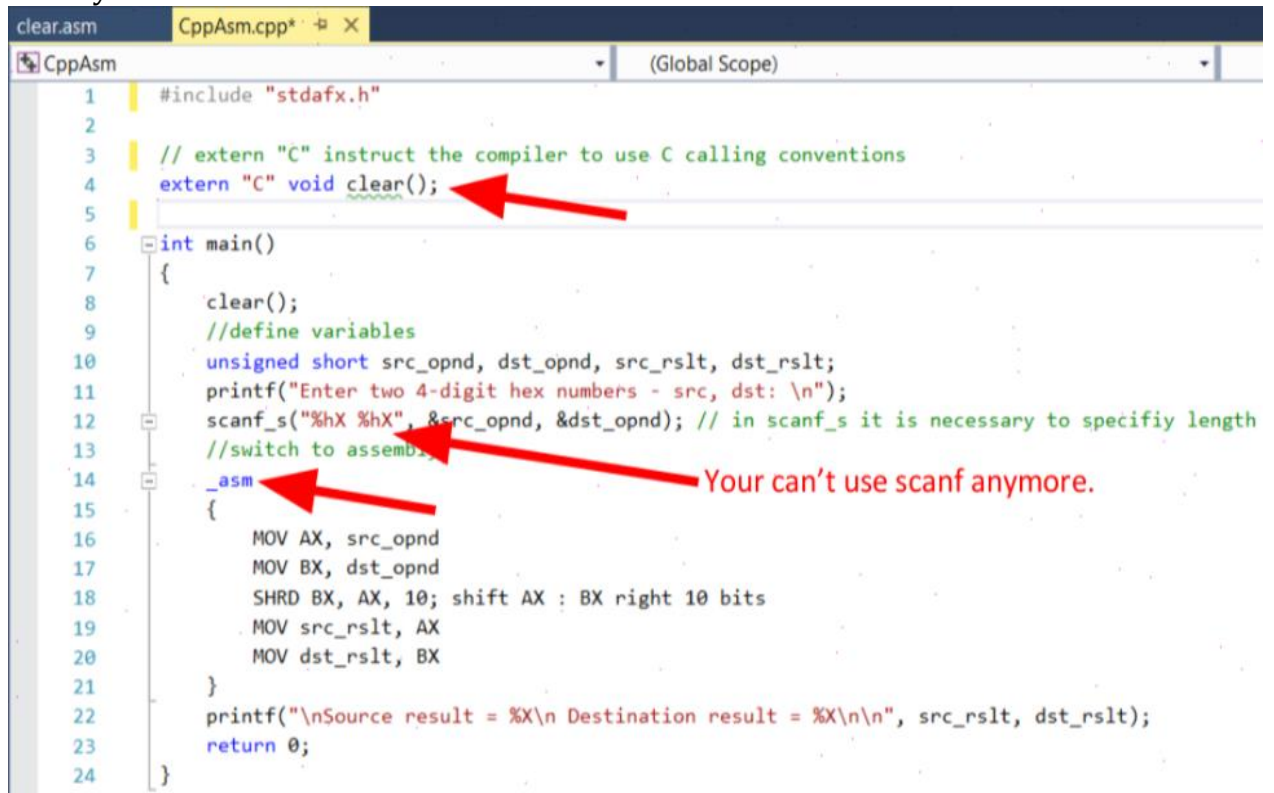
1. Select New Project > Visual C++ > Win32 > Win32 Console Application



2. Make sure to check Pre Compiled Header



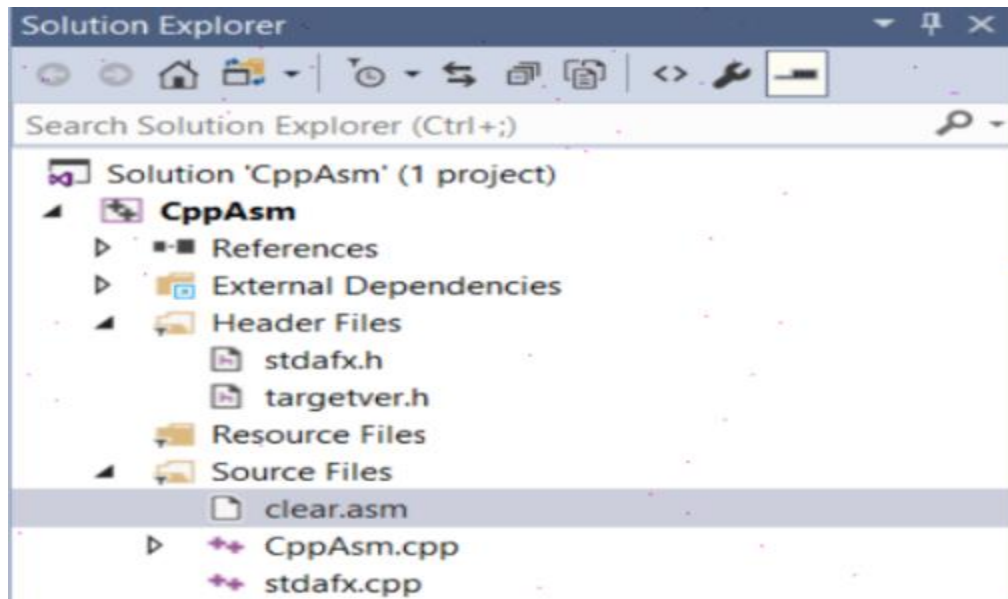
3. Write your C++ code in source file



The screenshot shows a C++ source file named `CppAsm.cpp` in a code editor. The code includes `stdafx.h` and defines an external C function `clear()`. The `main` function calls `clear()` and uses `scanf_s` to read two 4-digit hex numbers. A red arrow points to the `extern "C" void clear();` line. Another red arrow points to the `scanf_s` line, with a text annotation stating "You can't use scanf anymore." A third red arrow points to the `_asm` block, which contains assembly instructions for shifting bits.

```
1  #include "stdafx.h"
2
3  // extern "C" instruct the compiler to use C calling conventions
4  extern "C" void clear();
5
6  int main()
7  {
8      clear();
9      //define variables
10     unsigned short src_opnd, dst_opnd, src_rslt, dst_rslt;
11     printf("Enter two 4-digit hex numbers - src, dst: \n");
12     scanf_s("%hX %hX", &src_opnd, &dst_opnd); // in scanf_s it is necessary to specify length
13     //switch to assembly
14     _asm
15     {
16         MOV AX, src_opnd
17         MOV BX, dst_opnd
18         SHRD BX, AX, 10; shift AX : BX right 10 bits
19         MOV src_rslt, AX
20         MOV dst_rslt, BX
21     }
22     printf("\nSource result = %X\n Destination result = %X\n\n", src_rslt, dst_rslt);
23     return 0;
24 }
```

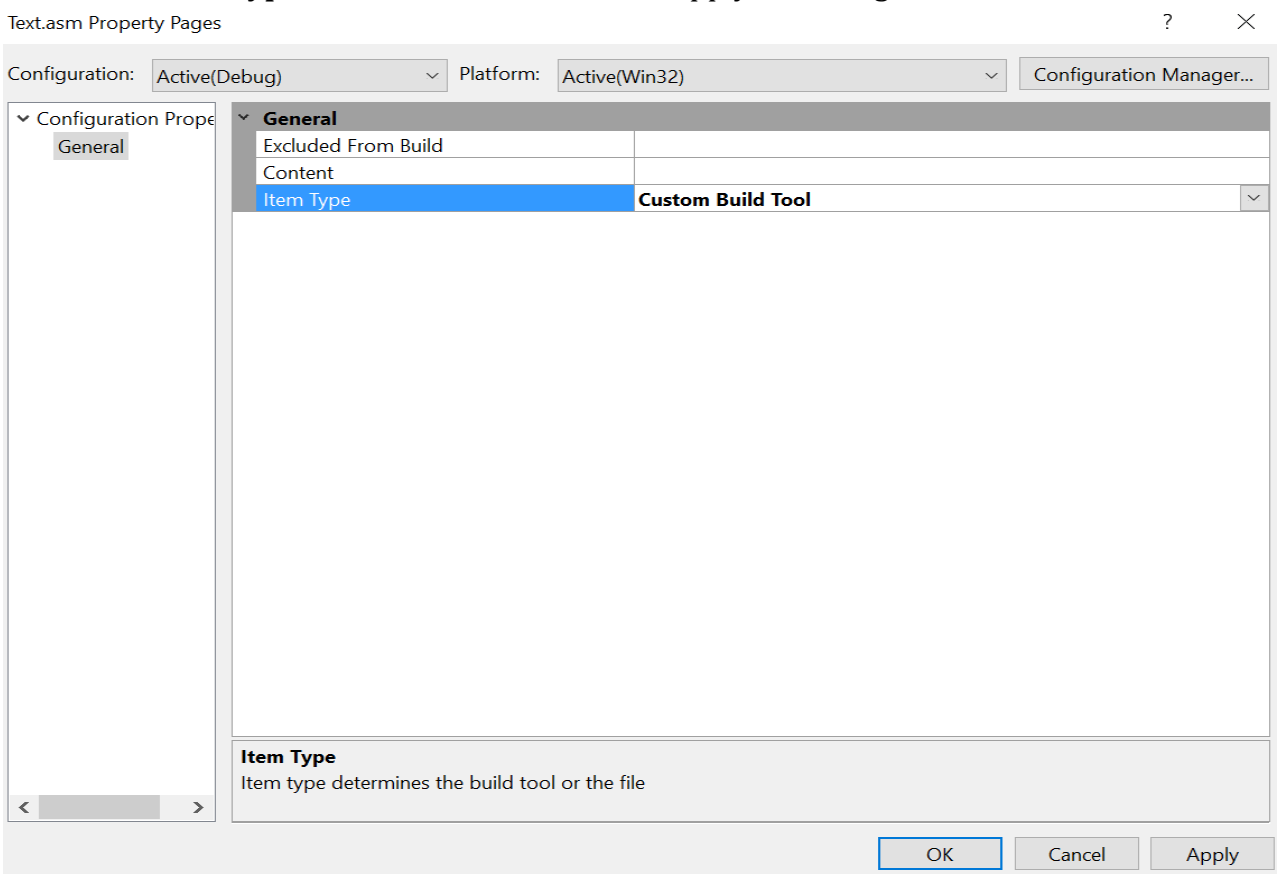
4. Add .asm file to the project



5. Write Assembly code in your .asm file

```
clear.asm  CppAsm.cpp
1  .686                ;Target processor.  Use instructions for Pentium class machines
2
3  .MODEL FLAT, C      ;Use the flat memory model. Use C calling conventions
4
5  .STACK 2048         ;Define a stack segment of 1KB (Not required for this example)
6
7  .DATA               ;Create a near data segment.  Local variables are declared after
8                      ;this directive (Not required for this example)
9      var_1 dword 10
10     str_1 byte 50,100,34,5,6,78,12,45,67
11     str_2 byte 5000 dup(?)
12
13 .CODE               ;Indicates the start of a code segment.
14
15
16 clear PROC
17     xor eax, eax
18     xor ebx, ebx
19     ret
20 clear ENDP
21 END
```

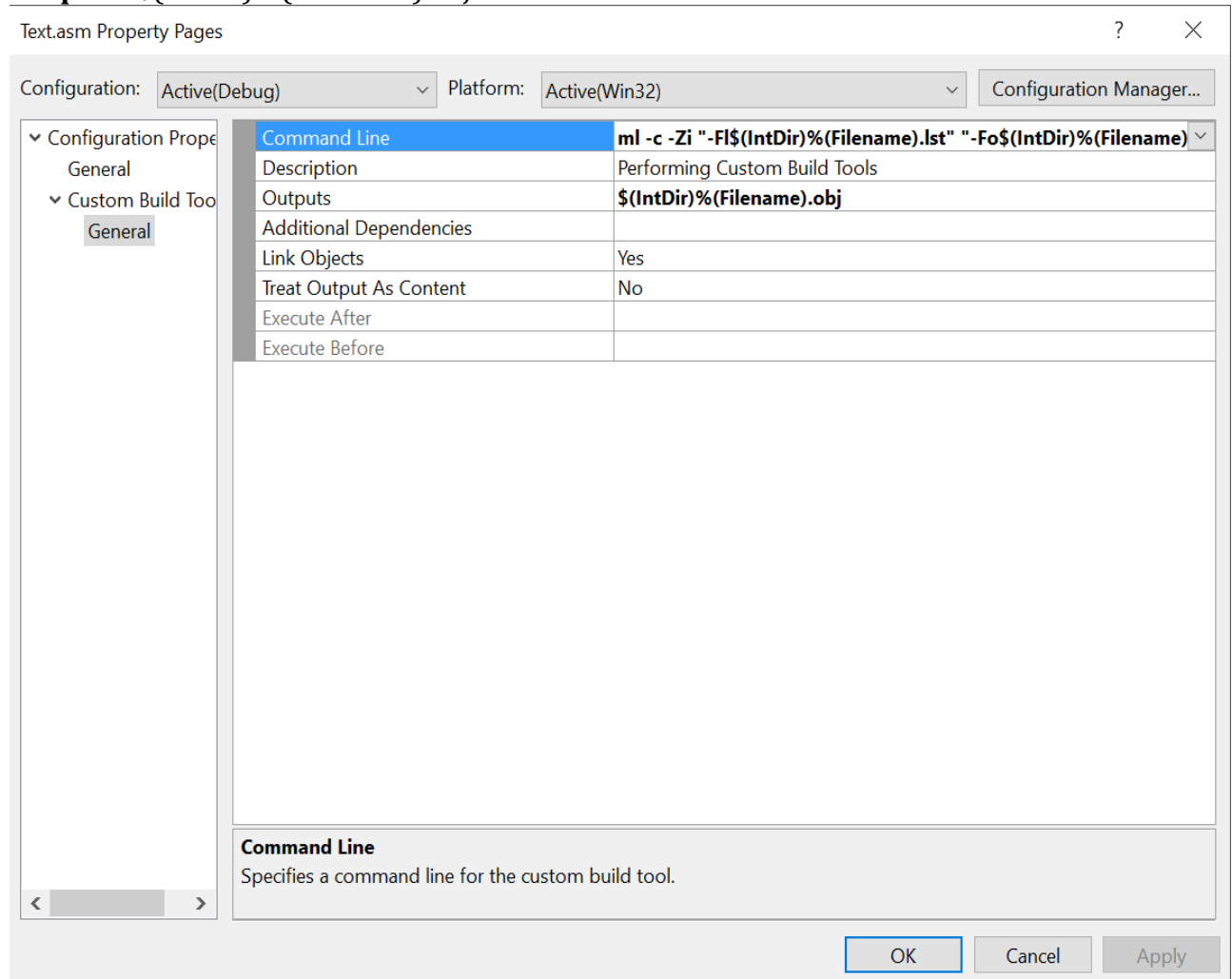
6. Right click on .asm file and select Properties. Select **Configuration Properties > General > Item Type > Custom Build Tool** and Apply the changes.



7. Select **Configuration Properties > Custom Build Tool > General**

Command Line: `ml -c -Zi "-Fl$(IntDir)%(Filename).lst" "-Fo$(IntDir)%(Filename).obj" "%(FullPath)"`

Outputs: `$(IntDir)%(Filename).obj`



Example:

C++ Code:

```
#include "stdafx.h"
```

```
// extern "C" instruct the compiler to use C calling conventions
extern "C" void clear();
```

```
int main()
{
```

```
    clear();
    //define variables
    unsigned short src_opnd, dst_opnd, src_rslt, dst_rslt;
    printf("Enter two 4-digit hex numbers - src, dst: \n");
    scanf_s("%hX %hX", &src_opnd, &dst_opnd); // in scanf_s it is necessary to
specifiy length
```

```

//switch to assembly
_asm
{
    MOV AX, src_opnd
    MOV BX, dst_opnd
    SHRD BX, AX, 10; shift AX : BX right 10 bits
    MOV src_rslt, AX
    MOV dst_rslt, BX
}
printf("\nSource result = %X\n Destination result = %X\n\n", src_rslt, dst_rslt);
return 0;
}

```

Assembly Code:

```

.686                ;Target processor. Use instructions for Pentium class machines

.MODEL FLAT, C       ;Use the flat memory model. Use C calling conventions

.STACK 2048          ;Define a stack segment of 1KB (Not required for this example)

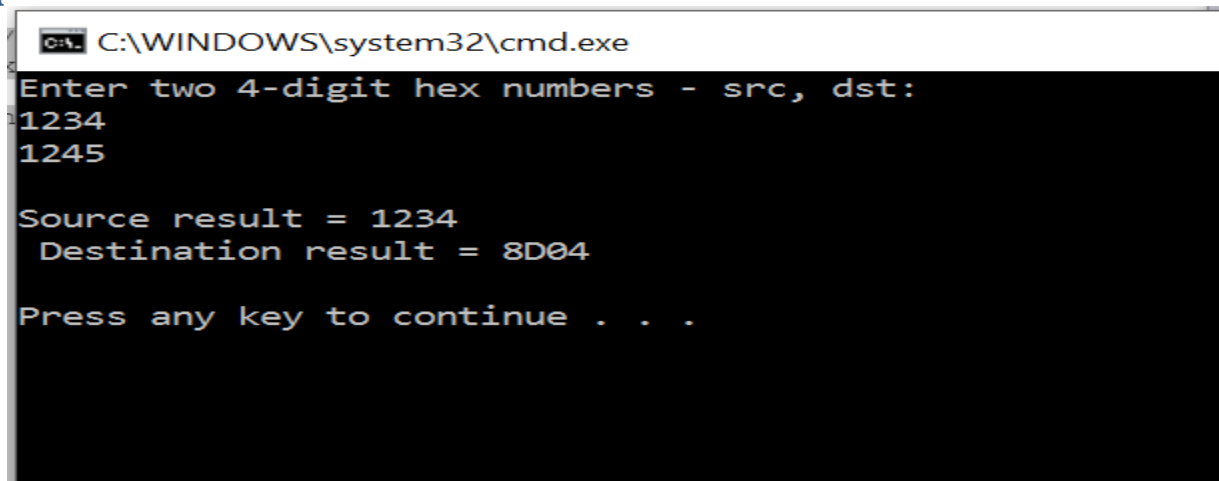
.DATA                ;Create a near data segment. Local variables are declared after
                    ;this directive (Not required for this example)
    var_1 dword 10
    str_1 byte 50,100,34,5,6,78,12,45,67
    str_2 byte 5000 dup(?)

.CODE                ;Indicates the start of a code segment.

clear PROC
    xor eax, eax
    xor ebx, ebx
    ret
clear ENDP
END

```

Output:



```

C:\WINDOWS\system32\cmd.exe
Enter two 4-digit hex numbers - src, dst:
1234
1245

Source result = 1234
Destination result = 8D04

Press any key to continue . . .

```

ACTIVITIES:

1. Write a program in C++ which takes input from user and contains a procedure in assembly named **ThreeProd** that displays the product of three numeric parameters passed in a function argument.
2. Write a program in C++ which takes input from user and contains a procedure in assembly named **GCD** which calculates their GCD.