Microprocessor Programming

Dr. Tin Ni Ni Kyaw

Ph. D (Kumamoto University, Japan)

Associate Professor

Department of Computer Engineering and Information Technology

Yangon Technological University

Yangon, Myanmar

Course Information

Course Name	Microprocessor Programming
No. of Lectures	12
Each Lecture Duration	150 minutes
Special Requirement or Resources to Deliver the Course	Computer and Microsoft Visual C++
Pre-requisite/co-requisite (if any)	Programming Language (C Programming)

References

Textbook

- James T. Streib, “Guide to Assembly Language, A Concise Introduction”, (2011).

Additional References

- Kip R. Irvine, “Assembly Language for x86 Processors”, 6th Edition, (2011)
- Barry B. Brey, “Intel Microprocessor, Architecture, Programming and Interfacing”, 8th Edition, (2009)

Lesson Plan

Week	Topics
Week 1	Introduction to Assembly Language
Week 2 to Week 6	Input/Output, Arithmetic Instructions, Operators, Selection Structures, Iteration Structures
Week 7	Tutorials
Week 8 to Week 13	Shifting and Rotating, Stack, Procedures, Macro, Arrays, Strings
Week 14	Tutorials
Week 15	Revision

Assessments and Grading

Types of Assessment	Percentage (%)
Tutorial	5%
Practical	15%
Final Examination	80%
Total	100%

Grade	Marks
A+	75~100
A	70~74
A-	65~69
B+	60~64
B	55~59
B-	50~54
C+	45~49
C	42~44
C-	40~41
D	<40

Course Learning Outcomes

- To realize the basic and important concepts of assembly language programming
- To understand the relationship between the assembly language, the high-level language and low-level language
- To be able to construct the assembly programming solutions for the given problems
- To get the knowledge of computer architecture and organization from the point of software view

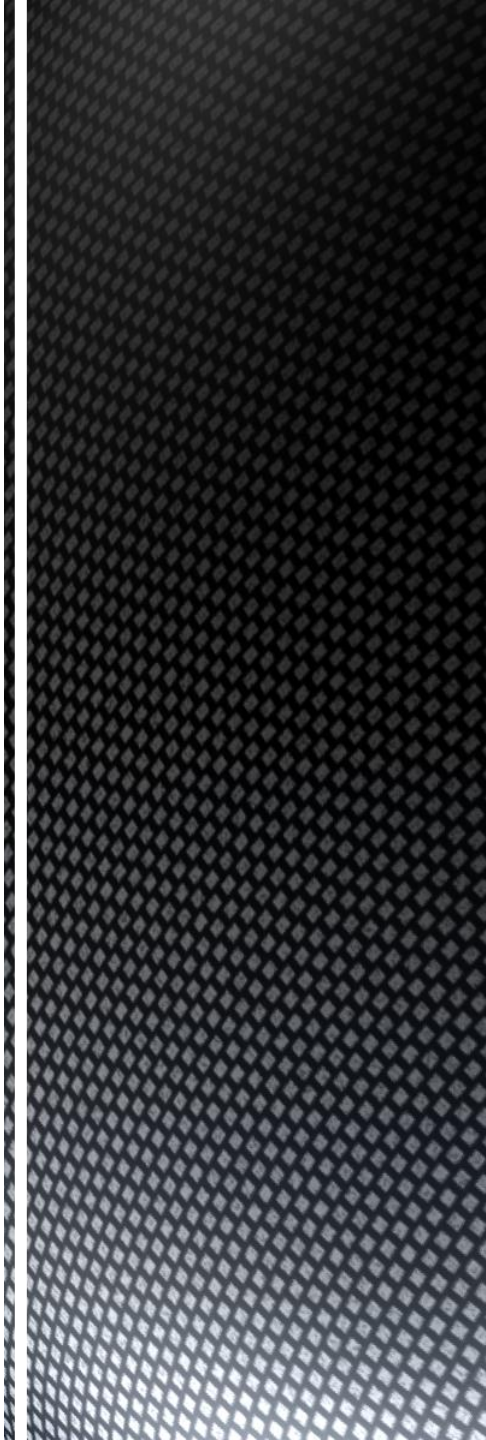
Microprocessor Programming

Lecture 1

Introduction to Assembly Language

Contents

- Introduction
- Basic Layout
- Instruction Parts
- Variable Declaration and Initialization
- Registers
- Data Movement
- Errors
- Summary
- Practical Works
- Assignments



Introduction

- There are two kinds of programming language: high-level language and low-level language.
- C, C++, and Java are some examples of high-level languages and the source codes in the programs are easy to understand.
- Low-level languages are also known as machine languages or binary languages as they are coded by using zeros and ones.
- In addition to high-level and low-level languages, there also exists an assembly language that uses mnemonics for the instructions.

Microsoft Assembler (MASM)

- Compilers, interpreters and assemblers are the language translators used to translate the source codes from one level to another level.
- To convert from a high-level language to a low-level language, it is done by the compilers and interpreters.
- Whereas, an assembler which is a utility program converts source codes from assembly language into machine language.
- For translation to Intel machine language, there exists many assemblers and here, MASM (Microsoft Assembler) will be used.

Linker and Debugger

- Besides the assembler, linker and debugger are also important for assembly language programming.
- The assembler creates the individual object files containing machine codes and those files need to be combined as a single executable program. This is done by the linker.
- The function of the debugger is to debug the executable programs at the assembly level.

Basic Layout of An Assembly Program

```
.386
.model flat, c
.stack 100h
.data
.....
.code
main proc
.....
ret
main endp
end
```

Basic Layout of An Assembly Program (Cont.)

- The .386 directive means that the type of CPU required for the program is Intel 386, the first x86 processor.
- The next directive, .model flat, identifies that the program uses 32-bit addresses in protected mode and is possible to address 4 GB of memory.
- The c in the model directive specifies that the program can be run in the Microsoft Visual C++.

Basic Layout of An Assembly Program (Cont.)

- The .stack 100h directive means that the size of the stack is 100 hexadecimal bytes large or 256 bytes.
- The .data directive indicates the beginning of data segment and it contains variable declarations and initializations.
- The .code directive indicates the beginning of code segment and it contains instructions.
- The main is the label for the name of the program.

Basic Layout of An Assembly Program (Cont.)

- The proc directive means procedure and it identifies the beginning of the program for execution.
- The ret instruction is similar to the return 0 statement in C or C++.
- The main endp label and directive indicate the end of the procedure called main.
- Lastly, the end directive identifies the end of the program for the assembler.

Instruction Parts of Assembly Program

- An instruction is an executable statement in a program.
- The assembler translates the instructions and converts into machine language which are loaded and executed by the CPU at runtime.
- An instruction contains the following four basic parts:

1. Label
2. Opcode
3. Operand
4. Comment

Instruction Parts of Assembly Program (Cont.)

1. Label

- The first field of the instruction is label and it is typically reserved for the variable names.
- It is also used for branching to various instructions.
- A label also acts as a place marker for instructions and data.
- To indicate the addresses of instruction and variable, a label can be placed just before the instruction and the variable respectively.

Instruction Parts of Assembly Program (Cont.)

2. Opcode

- The second field is opcode or operation codes .
- Operation codes can be executable instructions and also assembler directives.
- Some examples of operation codes are mov, add, sub, mul, jmp, call.

Instruction Parts of Assembly Program (Cont.)

3. Operand

- The third field is operand.
- It is used for operands.
- The number of operands can be zero to three.
- The operand can be a register, memory location, immediate value and so on.

Instruction Parts of Assembly Program (Cont.)

4. Comment

- The optional last field is comment.
- It is typically used for writing comments.
- Comments can be written in two ways: single-line comments, beginning with a semicolon character and block comments, beginning with the COMMENT directive and a user-specified symbol.

Variable Declaration

- Similar to high-level languages, a variable name can begin with a letter and then be followed by letters or digits.
- Languages such as C, C++, and Java are case-sensitive for variable names.
- However, in assembly, the names are not case-sensitive.
- For example, the variables num and NUM refer to the same memory location in assembly program.
- The length of a variable name is 247 characters in maximum.

Variable Declaration (Cont.)

Table 1. Valid and Invalid Variable Names

Valid	Invalid
letter	1letter
num1	23num
numary456	123numary

Variable Declaration (Cont.)

Table 2. Types, Number of Bits, and Range of Values

Type	Number of bits	Range (inclusive)
sdword	32	–2,147,483,648 to +2,147,483,647
dword	32	0 to +4,294,967,295
sword	16	–32,768 to +32,767
word	16	0 to +65,535
sbyte	8	–128 to +127
byte	8	0 to +255

Variable Initialization

- Variable initialization is always in the data segment.
- The codes for variable initialization can be written in the next line after .data directive.
- First, take a look some examples of initializing the variables in C:

```
int num3 = 5;

char grade1;

char grade2='A';

int num1,num2;
```

Variable Initialization (Cont.)

- The equivalents of the above C codes in assembly language are as follows:

Label	Opcode	Operand	Comment
num3	sdword	5	; num3 initialized to 5
grade1	byte	?	
grade2	byte	'A'	
num1	sdword	?	;first number
num2	sdword	?	;second number

Registers

- There are many types of registers in all processors.
- Among them, general purpose registers can be accessible by the programmer.
- The original 16-bit Intel processors have the general purpose registers called ax, bx, cx, and dx.
- Later, the 386 microprocessor used 32-bit registers.
- These four general purpose registers in a modern Intel processor are called eax, ebx, ecx, and edx.

Registers (Cont.)

Table 3. Summary of Registers

Registers			Name	Usage
32 bits	16 bits	8 bits		
eax	ax	ah, al	Accumulator	Arithmetic and Logic
ebx	bx	bh, bl	Base	Arrays
ecx	cx	ch, cl	Counter	Loop
edx	dx	dh,dl	Data	Arithmetic
esi	si		Source Index	Strings and Arrays
edi	di		Destination Index	Strings and Arrays
esp	sp		Stack Pointer	Top of Stack
ebp	bp		Base Pointer	Stack Base
eip	ip		Instruction Pointer	Points to next instruction
eflags	flags		Flags	Status and Control Flags

Data Movement

- For data movement, mov instruction is used.
- A mov instruction always moves information from the right called the source operand to the left called the destination operand.
- The mov instruction is similar to the assignment symbol (the equals sign) in C, C++ and Java where the instruction does not necessarily move data.

Data Movement (Cont.)

Table 4. mov Instructions

Instruction	Meaning
mov mem, imm	move the immediate data to memory
mov reg, mem	move the contents of memory to a register
mov mem, reg	move the contents of a register to memory
mov reg, imm	move immediate data to a register
mov reg, reg	move the contents of the source (second) register to the destination (first) register

Data Movement (Cont.)

- Note that, in the table, there is no format for mov mem,mem.
- It means that it is not possible to move from one memory location to another memory location directly.
- For example, implementing the instruction num2=num1; as mov num2,num1 will be invalid.
- To do so, a register can be used as intermediary.
- Firstly, the contents of one memory location must be moved into a register and then the contents of the register must be restored into the other memory location.

Data Movement (Cont.)

- Example of moving an immediate value in C is as follows:

```
num3 = 5;
```

- In assembly language, the format mov mem, imm is used to copy the immediate data into one memory location as follows:

```
mov num3, 5
```

Data Movement (Cont.)

- Here is an example of integer data movement between two memory locations in C:

```
num2 = num1;
```

- In assembly language, this movement can be written by using mov reg, mem and mov mem, reg formats as follows:

```
mov   eax, num1    ; load eax with the contents of num1

mov   num2, eax    ; store the contents of eax in num2
```

- Since an integer is 32 bits long, a 32-bit register (eax) needs to be used.

Data Movement (Cont.)

- This is another example of character data movement between two memory locations in C:

```
letter2 = letter1;
```

- The equivalent of the above C codes in assembly language is as follows:

```
mov   al, letter1     ; load al with letter1

mov   letter2, al     ; store al in letter2
```

- Since a character is only 8 bits long, only an 8-bit register (al) can be used.

Errors

- In assembly language programs, the common types of error are as follows:

1. Syntax Error: easy to solve by using the error messages given by the program
2. Execution or Runtime Error: such as a division by zero error
3. Logic Error: the most difficult error to solve and take time to debug

Summary

- Assembly language uses mnemonics.
- The assembler knows what to do by the directives.
- The processor knows what to do by the instructions.
- One instruction includes four parts.
- Variable names are valid if they start with a letter and followed by letters or digits.

Summary

- The four general purpose registers are eax, ebx, ecx, and edx.
- The contents cannot be moved directly from one memory location to another by using mov instruction.
- sdword is used to declare integers and for characters, byte is used.
- Like high-level languages, error messages are given for syntax and execution (or run-time) errors, but not for logic errors.

Microprocessor Programming

Practical Works:

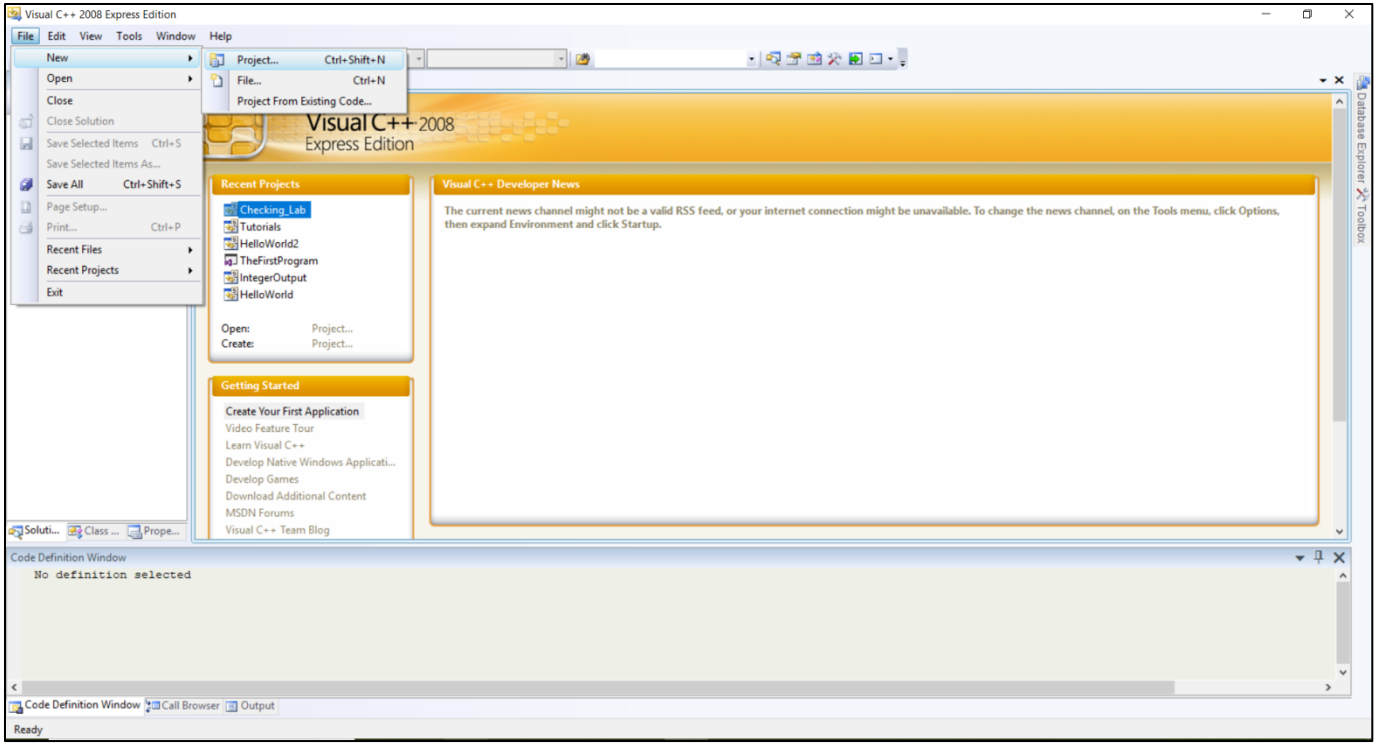
Creating New Projects and Implementing A Simple C Program

Requirements for Practical Works

- The software used in this lecture is Microsoft Visual C++ 2008 Express Edition with SP1 and we can get the free download.
- To invoke the MASM assembler instead of the C++ compiler, some careful changes to the properties section of the project and assembly program (*.asm)* file are required.

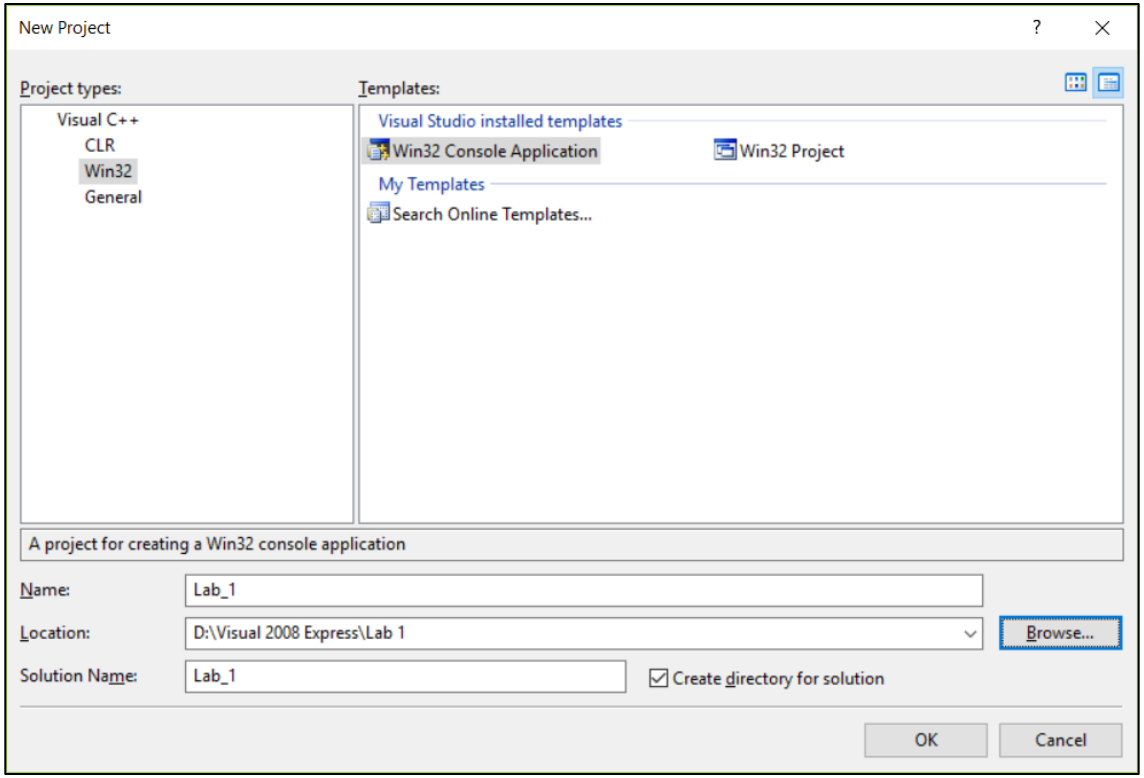
Creating New Project

- Open the Microsoft Visual C++ 2008.
- Create new project as shown in figure.



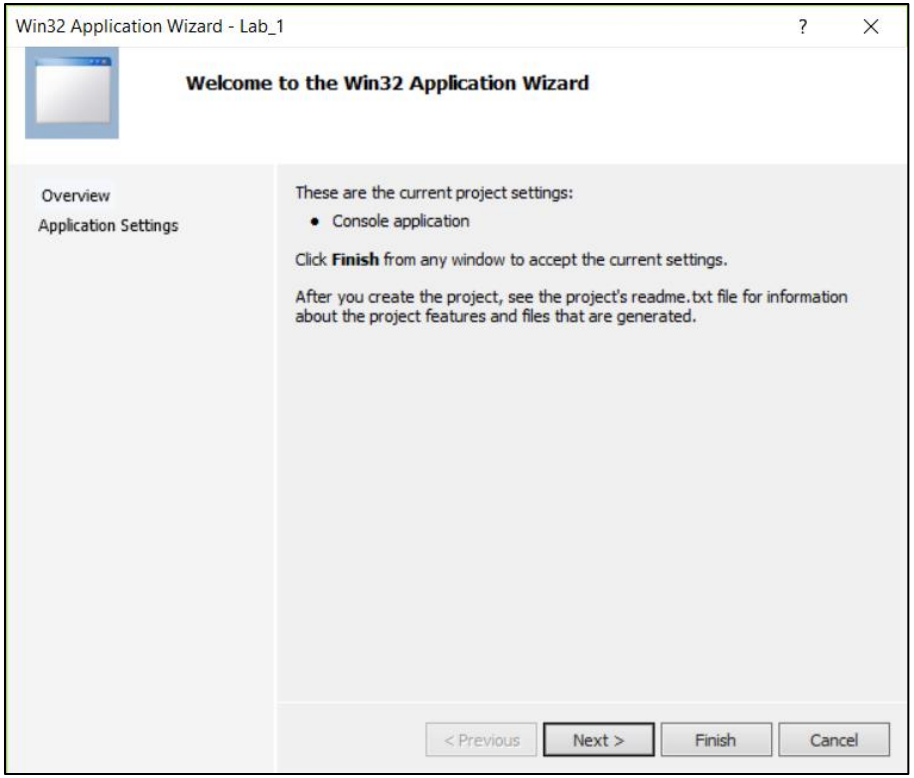
Creating New Project

- Enter new project name and choose the desired location of the project as shown in figure.



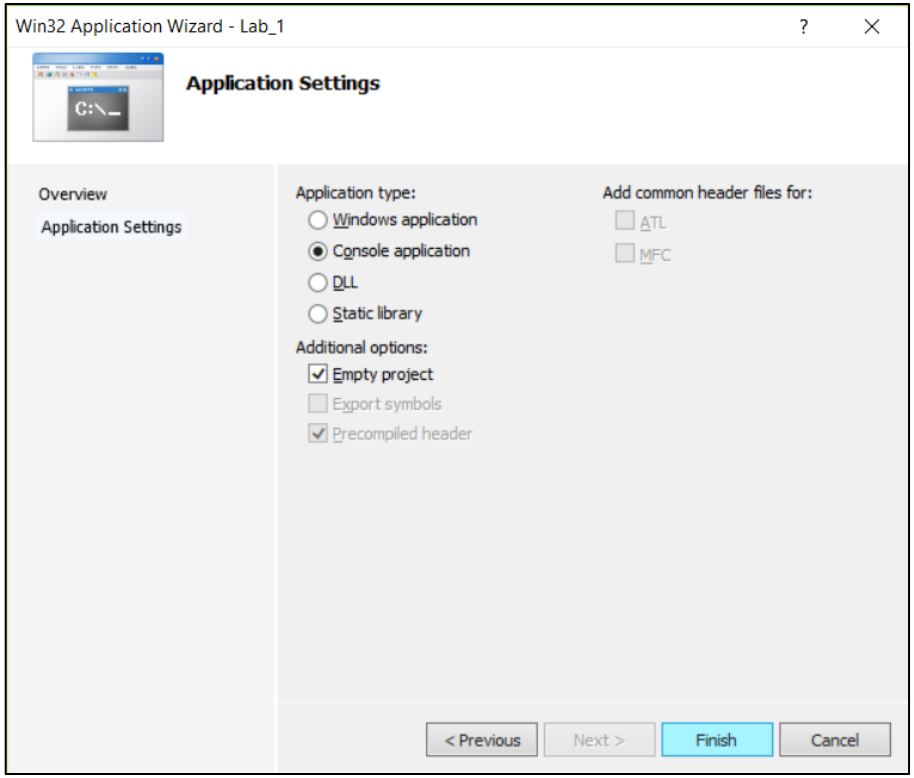
Creating New Project

- View the application settings as shown in figure and then click Next button.



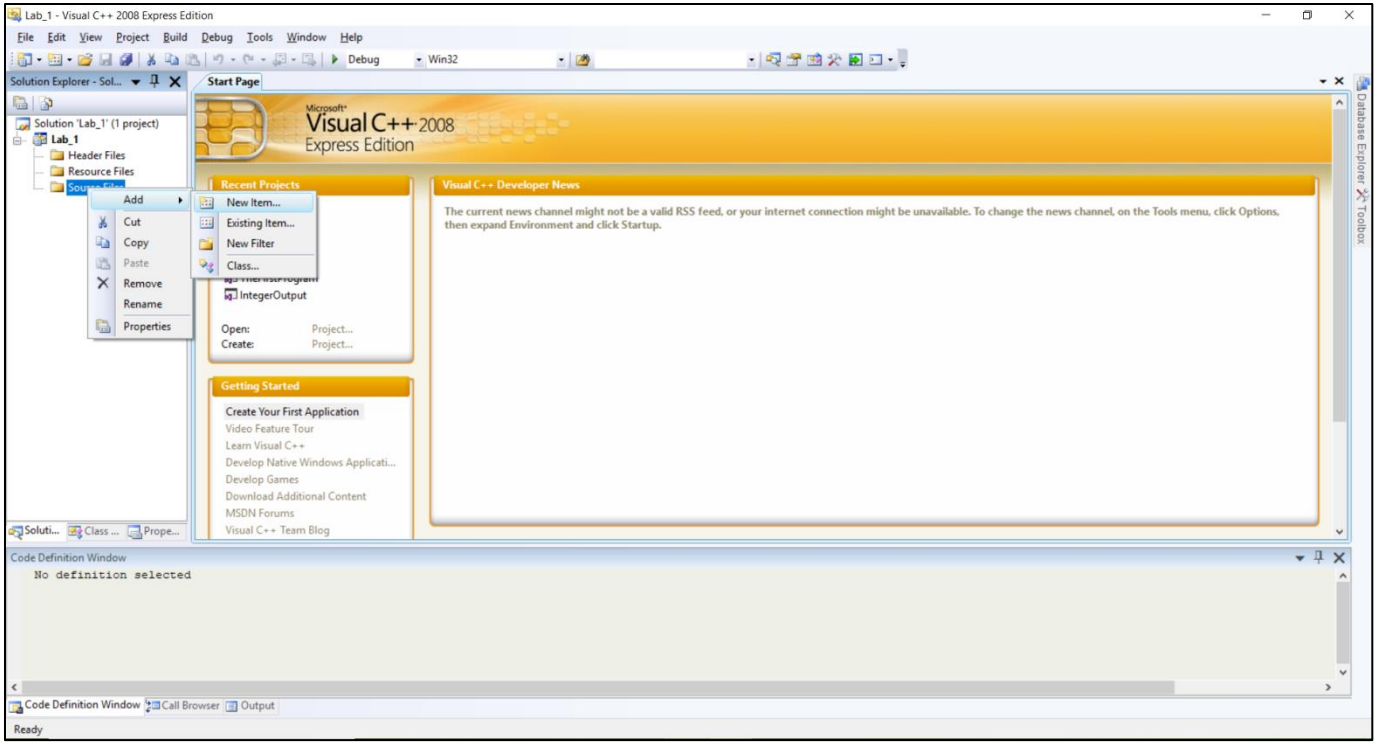
Creating New Project

- Check Empty project in Additional options as shown in figure and then click Finish button.



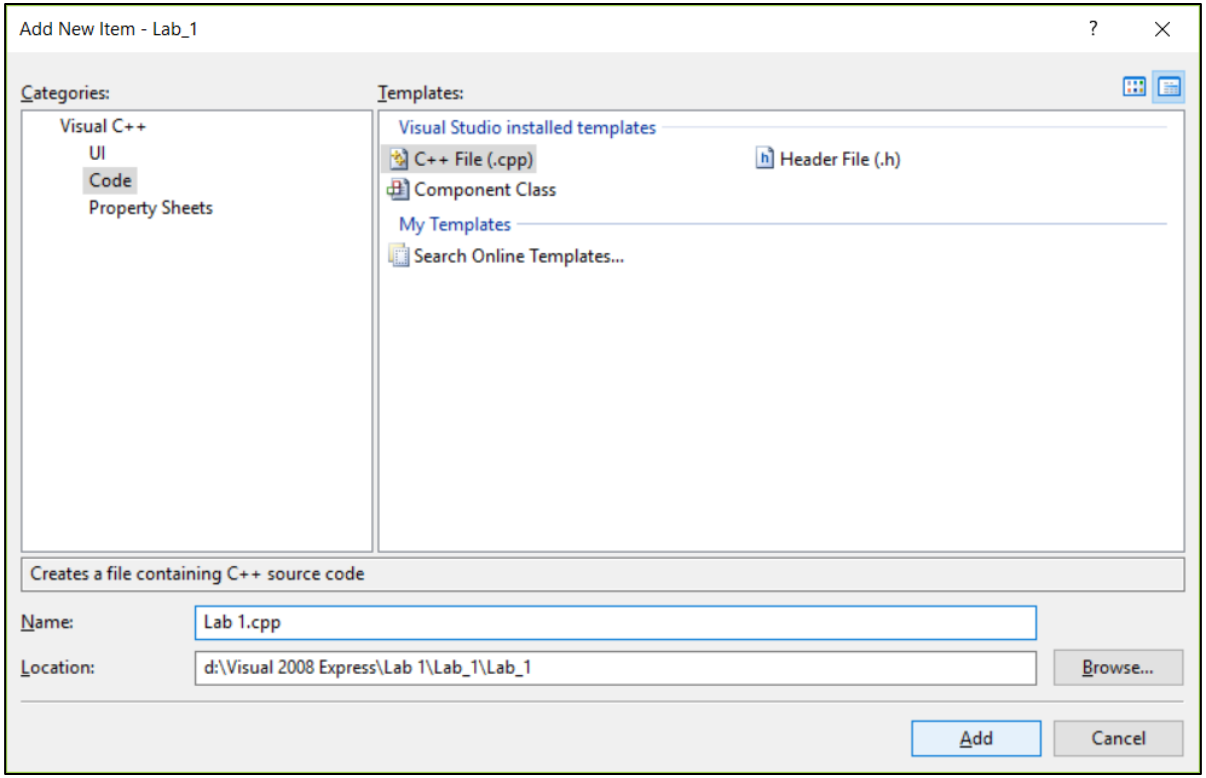
Implementing A Simple C Program

- Now, the new project, named Lab_1, is created.
- Then, add New Item in Source Files to write down the program as shown in figure.



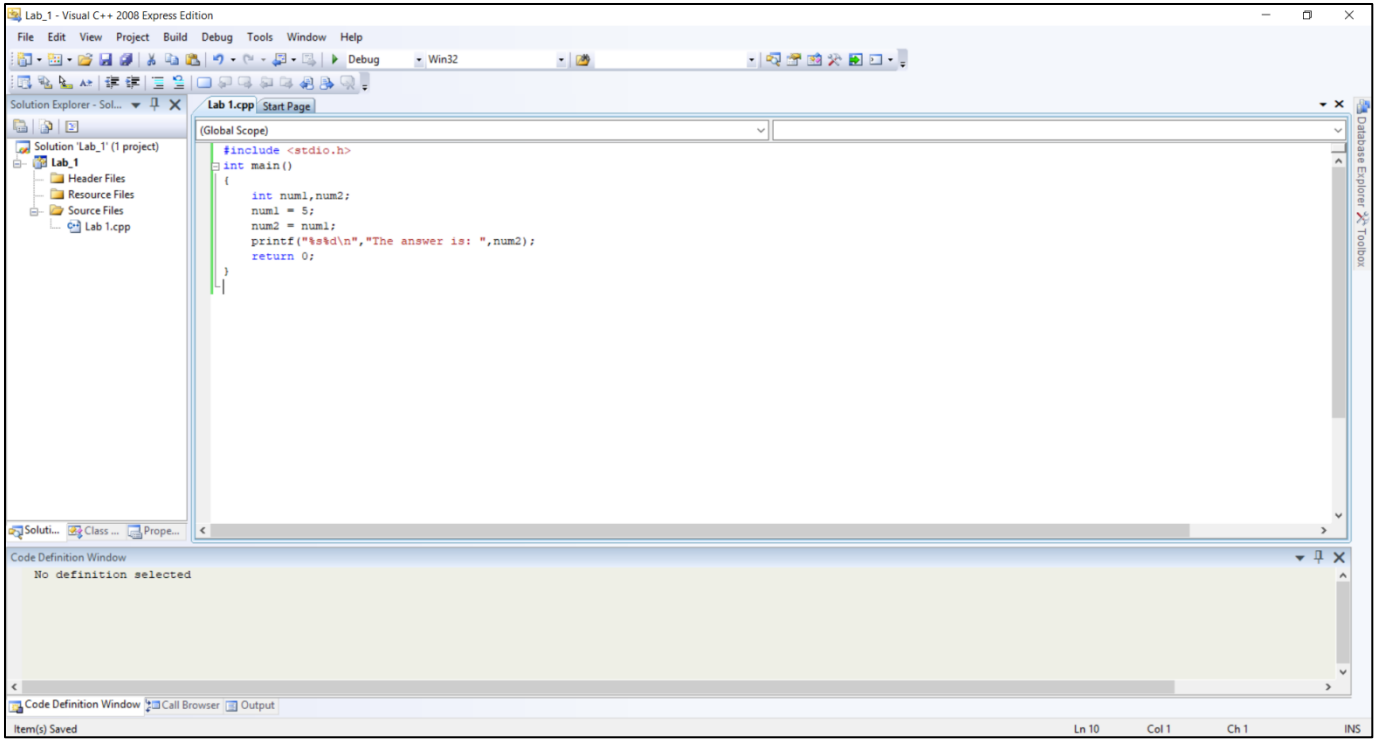
Implementing A Simple C Program

- Enter program name with extension (.cpp) for simple C program as shown in figure.



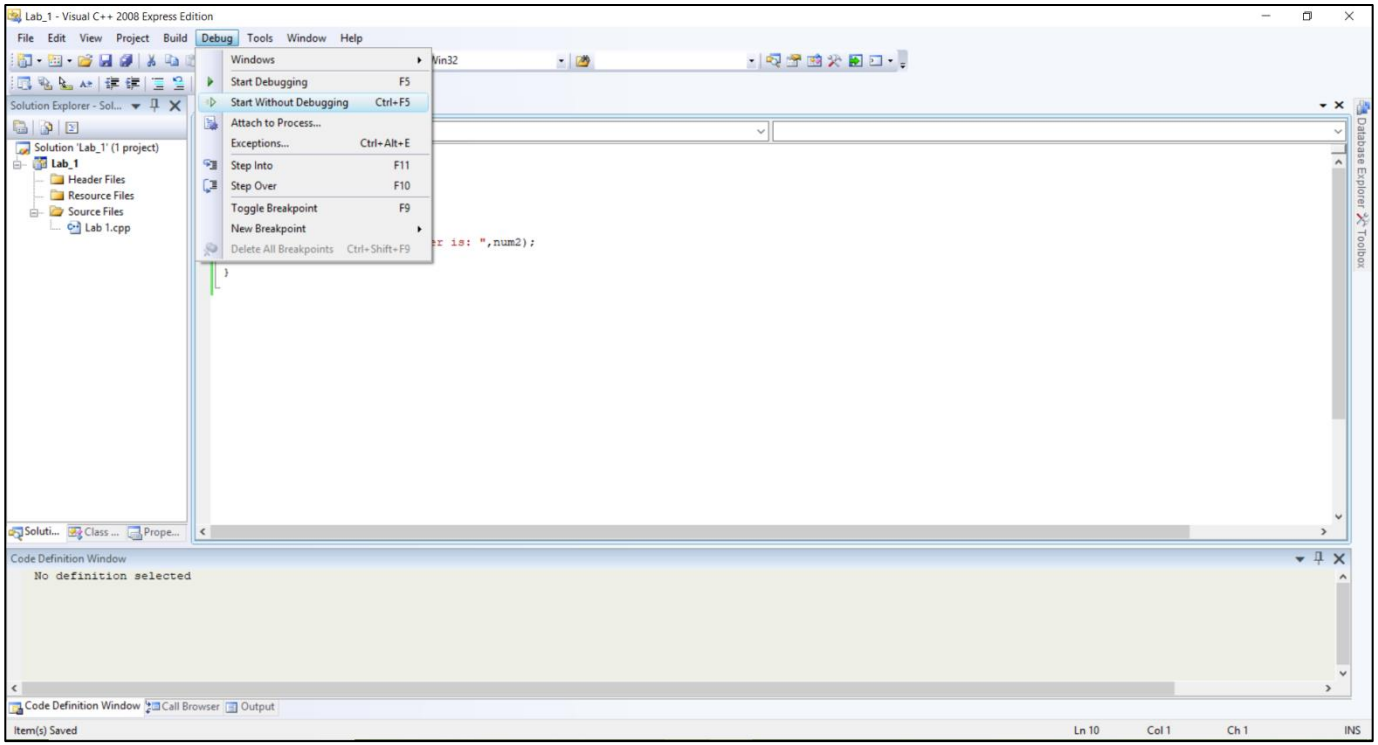
Implementing A Simple C Program

- Write down a simple C program as shown in figure.



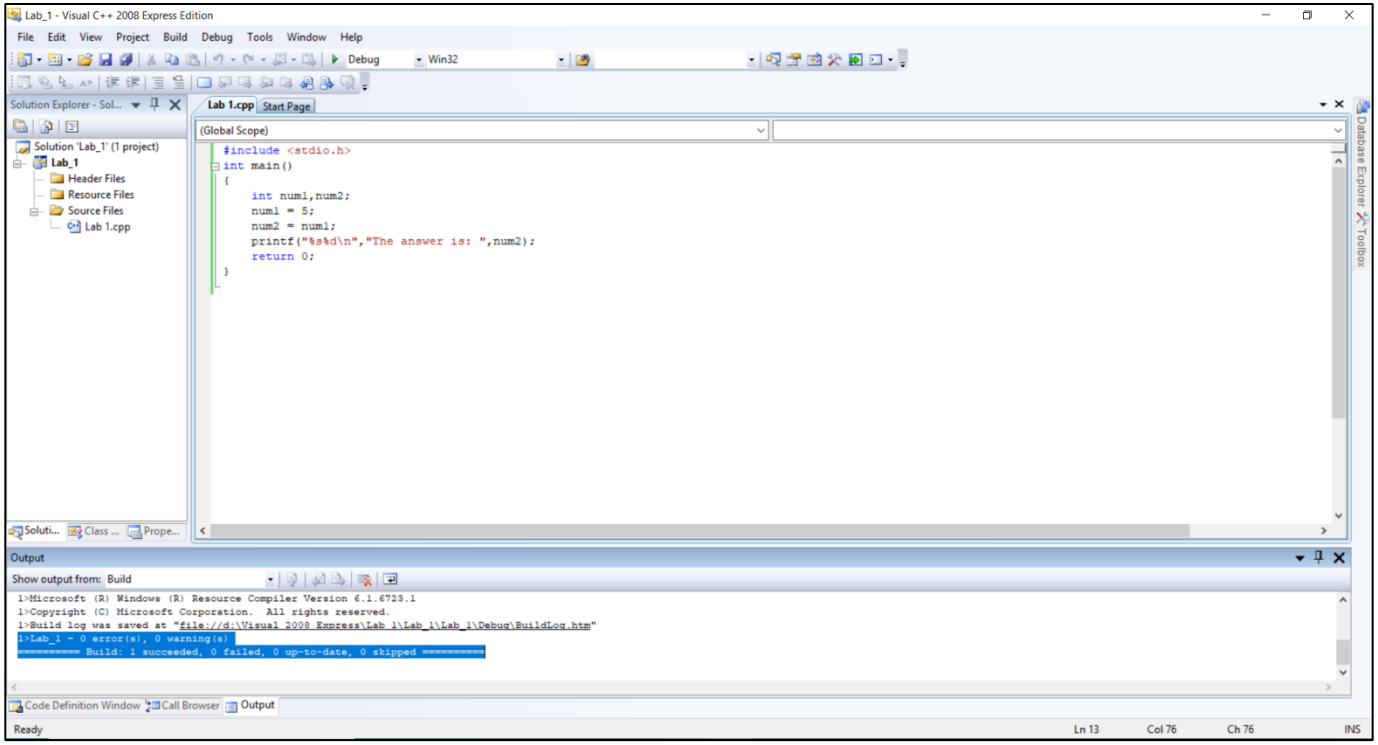
Implementing A Simple C Program

- After that, run the program as shown in figure.



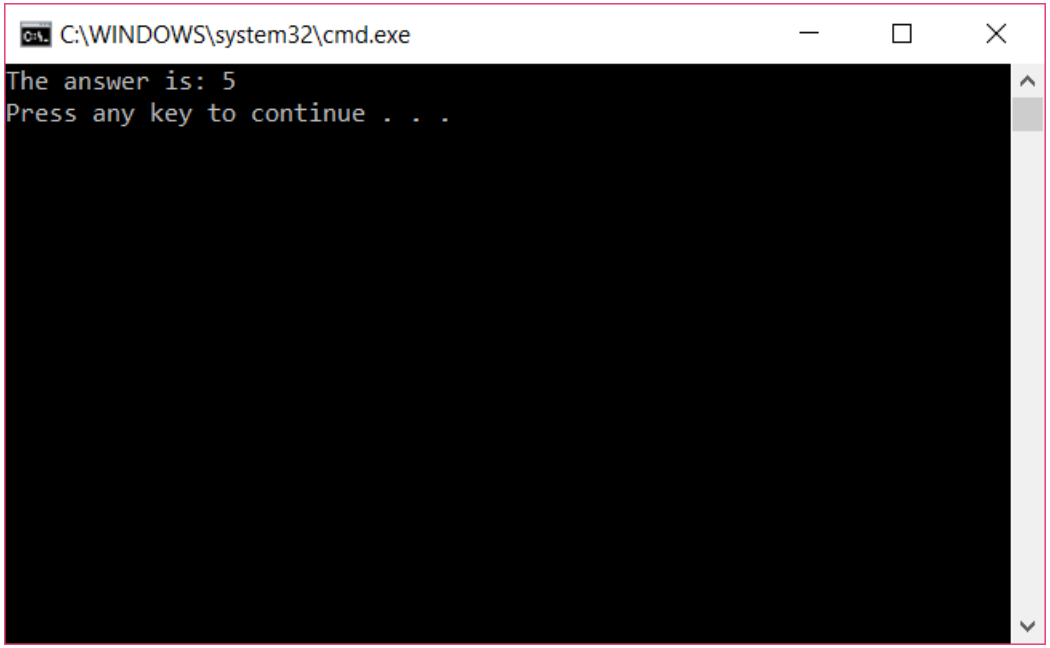
Implementing A Simple C Program

- View the build result of the program in Output window as shown in figure.



Implementing A Simple C Program

- Finally, the output of the program will be displayed as shown in figure.



Microprocessor Programming

Practical Assignment (Instructions)

Implementing Inline and Standalone Assembly Programs

Inline Assembly Program

Implement the following inline assembly program and show the output screen.

```
#include <stdio.h>
int main(){
    int num1,num2;
    num1 = 5;
    __asm {
        mov eax,num1
        mov num2,eax
    }
    printf("%s%d\n","The answer is: ",num2);
    return 0;
}
```

Standalone Assembly Program

Implement the following standalone assembly program and show the output screen.

```
            .386
            .model flat, c
            .stack 100h
printf      PROTO arg1:Ptr Byte, printlist:VARARG
            .data
msg1fmt     byte "%s%d",0Ah,0
msg1        byte "The answer is: ",0
num1        sdword ?
num2        sdword ?
            .code
main        proc
            mov num1,5
            mov eax,num1
            mov num2,eax
            INVOKE printf, ADDR msg1fmt, ADDR msg1, num2
            ret
main        endp
            end
```

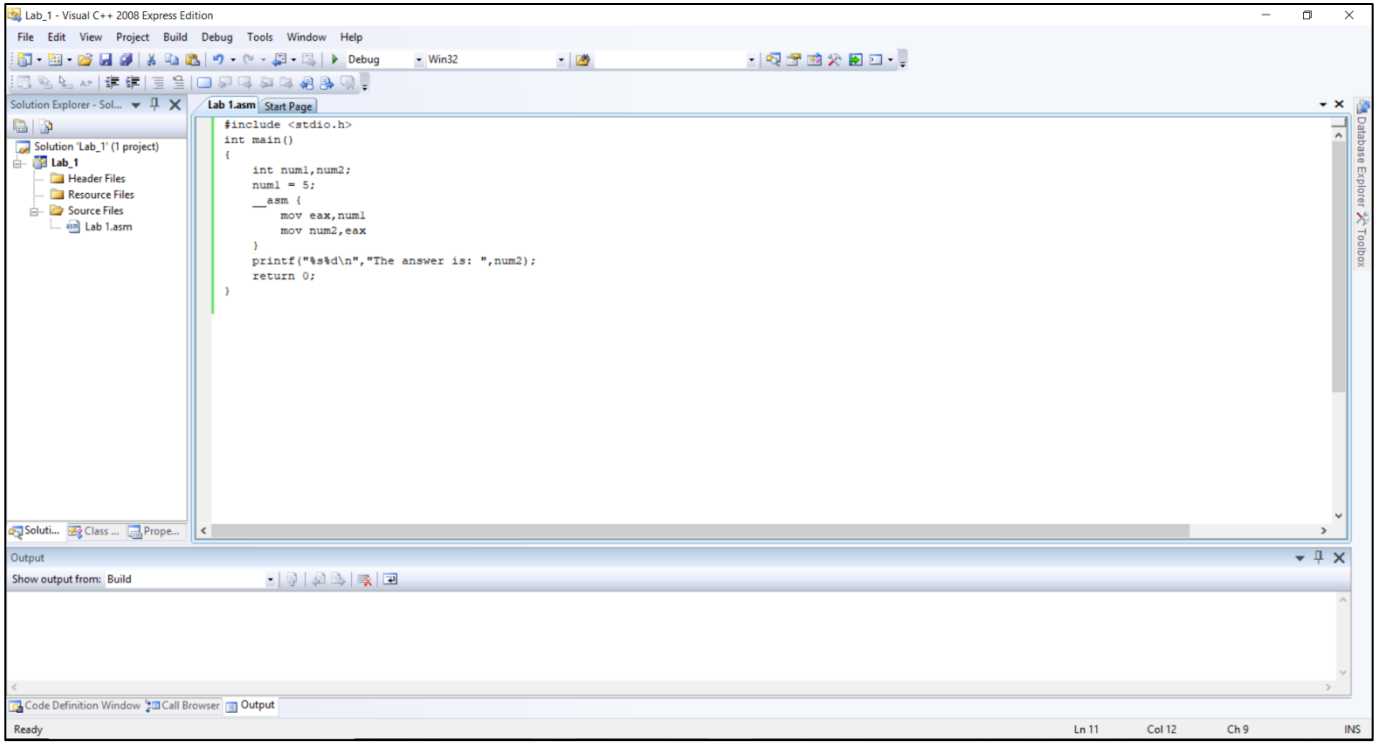
Microprocessor Programming

Practical Assignment (Report)

Implementing Inline and Standalone Assembly Programs

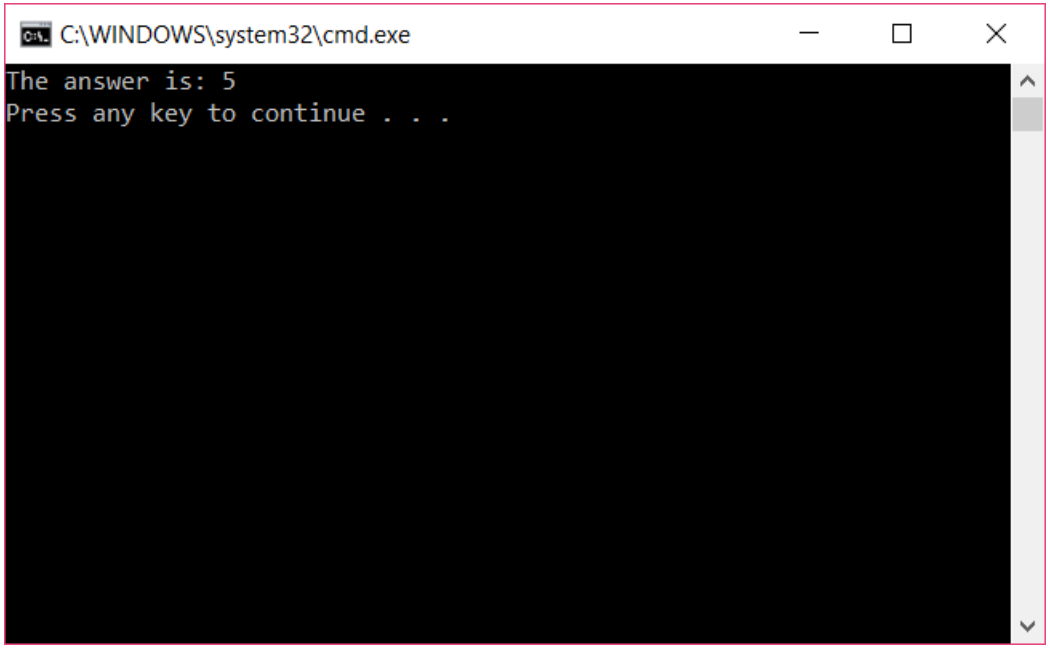
Implementing Inline Assembly Program

- The given inline assembly program is implemented as shown in figure.



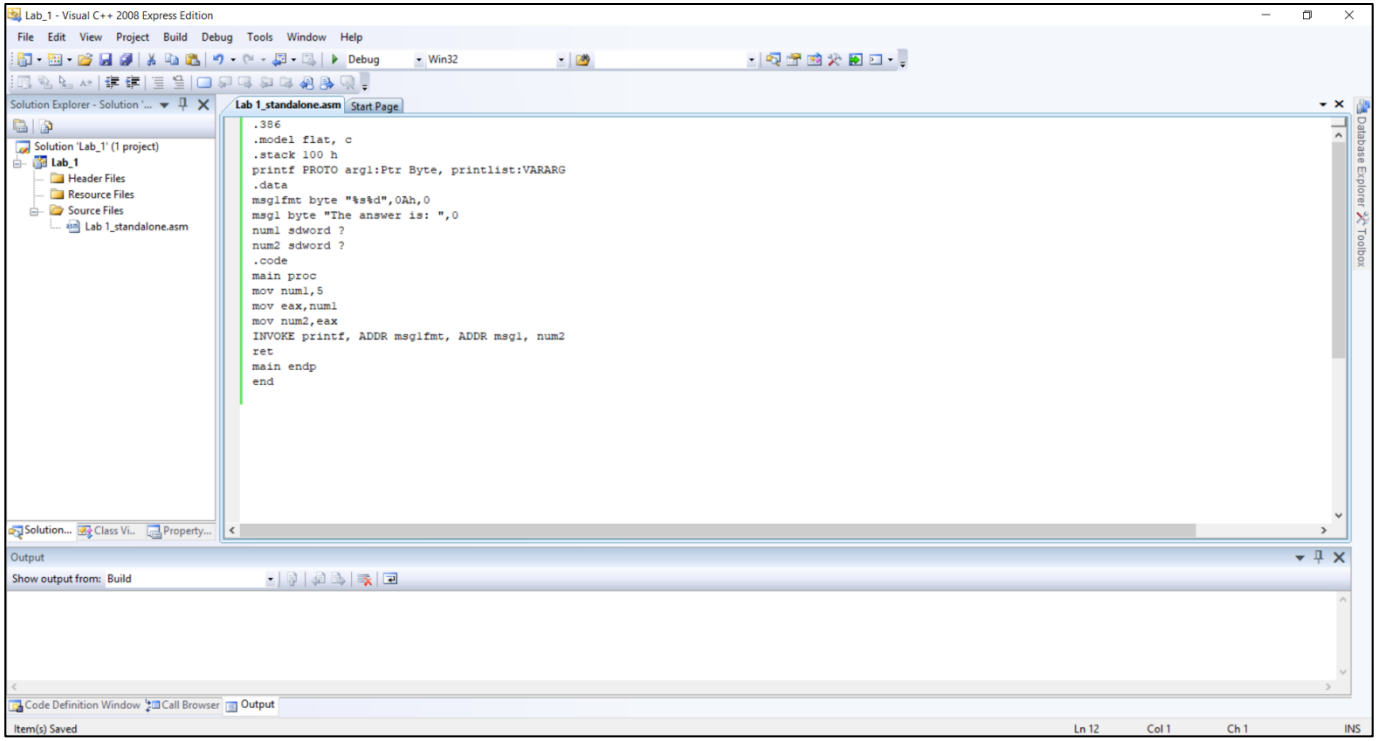
Implementing Inline Assembly Program

- After debugging the program, the output will be displayed as shown in figure.



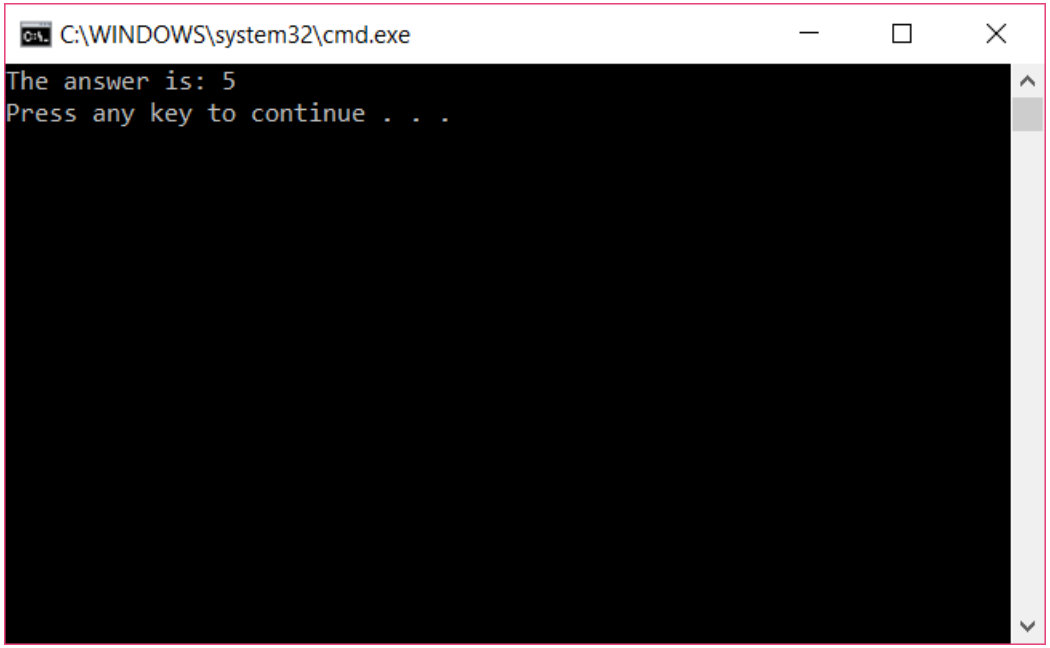
Implementing Standalone Assembly Program

- The given standalone assembly program is implement as shown in figure.



Implementing Standalone Assembly Program

- The output of the program will be displayed as shown in figure after the program is debugged.



Next Lecture

- How to get the inputs from the user and,
- How to display the messages or outputs to the user in assembly language program

Thank You