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

Microprocessor Programming

Lecture 4

Operators in Assembly Language

Contents

- Introduction
- Basic Operators in Assembly Language
- Types of Operator
- Order of Operators
- Summary
- Exercises
- Assignments

Introduction

- The assembler supports the following operators for use in expressions.

1. Arithmetic Operators
2. Logical Operators
3. Relational Operators

Basic Operators in Assembly Language

1. Arithmetic Operators

- The arithmetic operations can be performed by using the arithmetic operators.
- The arithmetic operators used in assembly language are shown in Table 1.

Table 1. Arithmetic Operators

Name	Mnemonics	Symbol
Addition	add	+
Subtraction	sub	-
Multiplication	mul	*
Division	div	/
Modulus	mod	%

Basic Operators in Assembly Language (Cont.)

2. Logical Operators

- The logical operations can be performed by using the logical operators.
- The logical operators used in assembly language are shown in Table 2.

Table 2. Logical Operators

Name	Mnemonics	Symbol
logical AND	and	&&
logical OR	or	||
logical NOT	not	!
logical XOR	xor	^

Basic Operators in Assembly Language (Cont.)

3. Relational Operators

- The relational operators used in assembly language are shown in Table 3.
- The result is only either true or false.

Table 3. Relational Operators

Name	Mnemonics	Symbol
equal to	e	=
not equal to	ne	!=
greater than	g	>
greater than or equal to	ge	>=
less than	l	<
less than or equal to	le	<=

Types of Operators

- As in the high-level language, there are two types of operators in assembly language.
- They are:
  1. Binary Operators and
  2. Unary Operators.

Binary Operators

- The operators which not only perform arithmetic on binary numbers but also have two operands (e.g., x + y), are called binary operators.
- The arithmetic operations such as addition, subtraction, multiplication and division, presented in the previous lecture, are performed by using the binary operators.

Binary Operators (Cont.)

- For example, to increment a variable x by 1 and decrement a variable y by 1, we can write in C as follows.

```
x = x + 1;

y = y - 1;
```

- The above code segment can be implemented in MASM by simply using the add and sub instructions:

```
add x,1

sub y,1
```

Unary Operators

- However, unary operators have only one operand such as the negative sign –y.
- In C, to increment a variable x by 1 and decrement a variable y by 1, the increment and decrement (unary) operators can be used as follows.

  x++; or ++x;

  y--; or --y;

- In a stand-alone statement, it does not matter whether the ++ or -- goes before or after the variable.

Unary Operators (Cont.)

- In assembly, the designers of Intel processor designed the two instructions that specifically increase and decrese by only 1.
- The mnemonic used for increment is inc and dec is for decrement.
- The format of each instruction is shown in Table 4.

Table 4. inc and dec instructions

Instruction
inc reg
inc mem
dec reg
dec mem

Unary Operators (Cont.)

- The implementations of increment x by 1 and decrement y by 1 using unary operators in MASM are as follow:

```
inc x

dec y
```

- These instructions take up less memory than do add or sub instructions.

Unary Operators (Cont.)

- In addition to increment and decrement unary operators, there exists another unary operator called negation.
- If we need to negate a number (to find the two's complement of a number) and store it in another memory location, the negation operator can be used.
- For example, x = -y; in C.

Unary Operators (Cont.)

- In assembly language, the instructions that can be used for negation are shown in Table 5.
- Here, neg is the mnemonic used for negation.

Table 5. neg instructions

Instruction	Meaning
neg reg	two's complement of the contents of the register
neg mem	two's complement of the contents of a memory location

Unary Operators (Cont.)

- The equivalent MASM code segment for x=-y; is as follow.

```
mov eax,y
neg eax
mov x,eax
```

- The neg instruction takes up less memory than does the corresponding sub instruction.

Order of Operators

- To understand the order of operation and to improve the skill of using assembly language arithmetic instructions, slightly more complicated arithmetic statements will be examined in this section.
- Here is an example of a high-level instruction:

```
answer = num1 + 3 - num2;
```

Order of Operators (Cont.)

- Since addition and subtraction have the same level of precedence in C, C++, and Java, the order of operation is from left to right.
- In this case, the addition should be done first and the subtraction done second.

Order of Operators (Cont.)

- For the case of MASM code, first, the contents of the variable num1 should be loaded into the eax register.
- Then, the number 3 needs to be added to the eax register.
- Note that the number 3 is not in a memory location but rather is implemented as an immediate value.
- Next, the value in num2 needs to be subtracted from the eax register.
- Lastly, the contents of the eax register need to be copied into the variable answer.

Order of Operators (Cont.)

- The following MASM code segment is the implementation of answer = num1 + 3 – num2.

```
mov eax,num1        ; load eax with the content of num1
add eax,3           ; add 3 to eax
sub eax,num2        ; subtract num2 from eax
mov answer,eax      ; store the result in answer
```

Order of Operators (Cont.)

- Again, the following C statement will be considered to illustrate the rules concerning order of operation:

```
answer = num1 + 3 * num2;
```

Order of Operators (Cont.)

- Multiplication has a higher precedence over addition.
- So, the value 3 must be loaded into the eax register first.
- Next, it is multiplied by the value in num2.
- Then, the value in num1 must be added to eax.
- Lastly, the value in eax should be stored in answer as follows:

```
mov eax,3          ; load eax with the number 3
imul num2          ; multiply eax by num2
add eax,num1       ; add the content of num1 to eax
mov answer,eax     ; store the content of eax in answer
```

Order of Operators (Cont.)

- As another example, consider the following C statement:

```
result = num3 / (num4 - 2);
```

- In this case, the expression in parentheses should be evaluated first although division has a higher precedence over subtraction.

Order of Operators (Cont.)

- So, the subtraction must be performed prior to the division.
- Then, the value in num3 is divided by subtraction result.
- Lastly, the value in eax should be stored in result.

```
mov ebx,num4          ; load ebx with num4
sub ebx,2             ; subtract 2 from ebx
mov eax,num3          ; load eax with the content of num3
cdq                   ; propagate sign bit into the edx register
idiv ebx              ; divide edx:eax by ebx
mov result,eax        ; store the contents of eax in result
```

Order of Operators (Cont.)

- Combining the unary instructions and the binary instructions, consider the following C statement:

```
v = -w + x * y - z++;
```

Order of Operators (Cont.)

- The order of operation is as follow.
- Firstly, the value in w should be negated.
- Secondly, perform the multiplication of x and y.
- Thirdly, perform the addition of the negated value of w and multiplication result.
- Next, perform the subtraction of z.
- Then, assign the result to v.
- Finally, the value of z should be incremented since it is a postfix ++.

Order of Operators (Cont.)

- The MASM code segment implementation is as:

```
mov ebx,w
neg ebx
mov eax,x
imul y
add eax,ebx
sub eax,z
mov v,eax
inc z
```

Summary

- Follow the order of operations when implementing arithmetic statements:
  - Parentheses first with the most nested first.
  - Unary minus sign has precedence over multiplication and division.
  - Multiplication and division prior to addition and subtraction.
  - In a tie, go left to right.

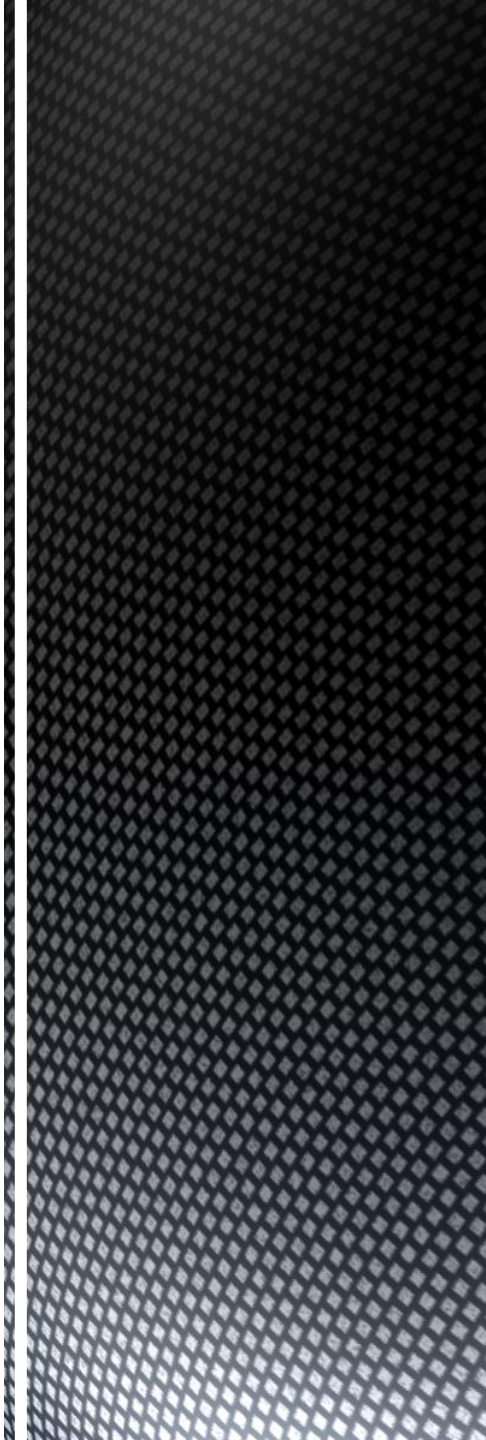
Summary

- Be careful with increment and decrement (++ and --) operators:
  - When stand-alone, there is no difference between prefix and postfix.
  - In an assignment statement, prefix is performed first prior to an assignment and postfix is performed after an assignment.

Microprocessor Programming

Exercises

Using Operators



Exercise 1

- Using order of operations from C, convert the following arithmetic statements into the equivalent assembly language statements.
- Be sure not to destroy the contents of any of the variables that appear only to the right of the assignment symbol:

A. x = x ∗ y + z ∗ 2;

B. a = b - c / 3;

C. total = num1 / num2 - (num3 ∗ num4);

D. r = -s + t++;

E. m = n ∗ ((i - j) ∗ k);

Exercise 1(A)

- The C code and the equivalent assembly language code segment is as follows:

```
x = x * y + z * 2;
```

```
mov eax,x
imul y
mov ecx,eax
mov eax,z
mov ebx,2
imul ebx
add ecx,eax
mov x,ecx
```

Exercise 1(B)

- The C code and the equivalent assembly language code segment is as follows:

```
a = b - c / 3;
```

```
mov eax,c
mov ebx,3
cdq
idiv ebx
mov edx,b
sub edx,eax
mov a,edx
```

Exercise 1(C)

- The C code and the equivalent assembly language code segment is as follows:

```
total = num1 / num2 - (num3 * num4);
```

```
mov eax,num3
imul num4
mov ebx,eax
mov eax,num1
cdq
idiv num2
sub eax,ebx
mov total,eax
```

Exercise 1(D)

- The C code and the equivalent assembly language code segment is as follows:

```
r = -s + t++;
```

```
mov eax,s
neg eax
add eax,t
mov r,eax
inc t
```

Exercise 1(E)

- The C code and the equivalent assembly language code segment is as follows:

```
m = n * ((i - j) * k);
```

```
mov eax,i
sub eax,j
imul k
imul n
mov m,eax
```

Exercise 2

- Using the order of operations from C, convert the following arithmetic statements into the equivalent assembly language code segment.
- Be careful to implement the unary minus sign, increment, and decrement operators carefully:

A. --i;

B. j = ++k - m;

C. z = -(x + y);

D. a = ++b - c++;

E. x = -y + z--;

Exercise 2(A)

- The C code and the equivalent assembly language code segment is as follows:

Exercise 2(B)

- The C code and the equivalent assembly language code segment is as follows:

```
j = ++k - m;
```

```
mov eax,k
inc eax
sub eax,m
mov j,eax
```

Exercise 2(C)

- The C code and the equivalent assembly language code segment is as follows:

```
z = -(x + y);
```

```
mov eax,x
add eax,y
neg eax
mov z,eax
```

Exercise 2(D)

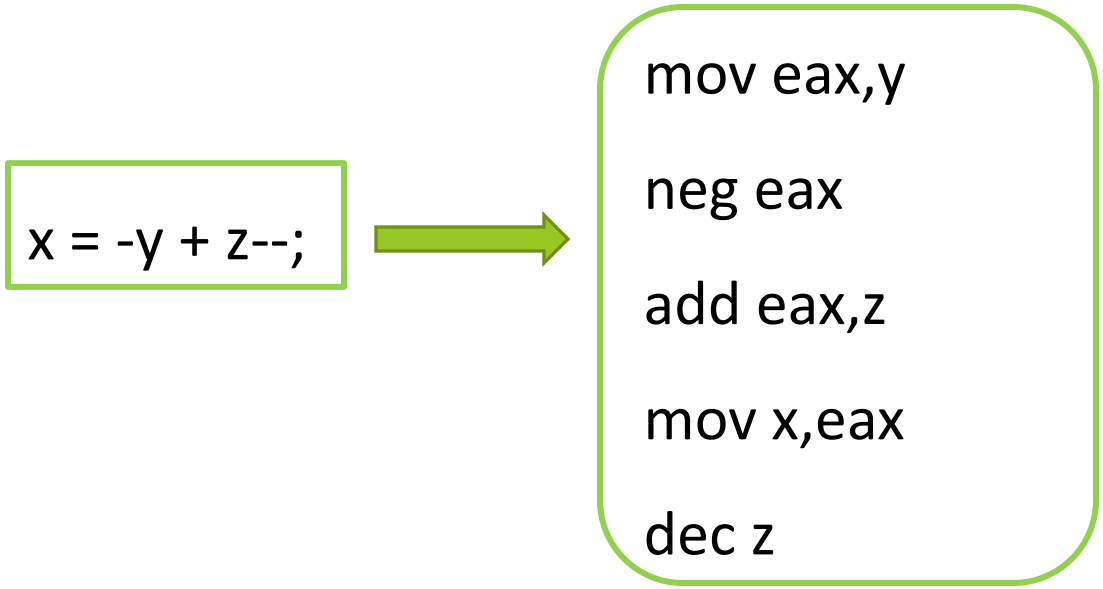
- The C code and the equivalent assembly language code segment is as follows:

```
a = ++b - c++;
```

```
mov eax,b
inc eax
sub eax,c
mov a,eax
inc c
```

Exercise 2(E)

- The C code and the equivalent assembly language code segment is as follows:



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

- Write the complete assembly programs for each assembly code segment from Exercise 1 and display the outputs.

Assignment 2

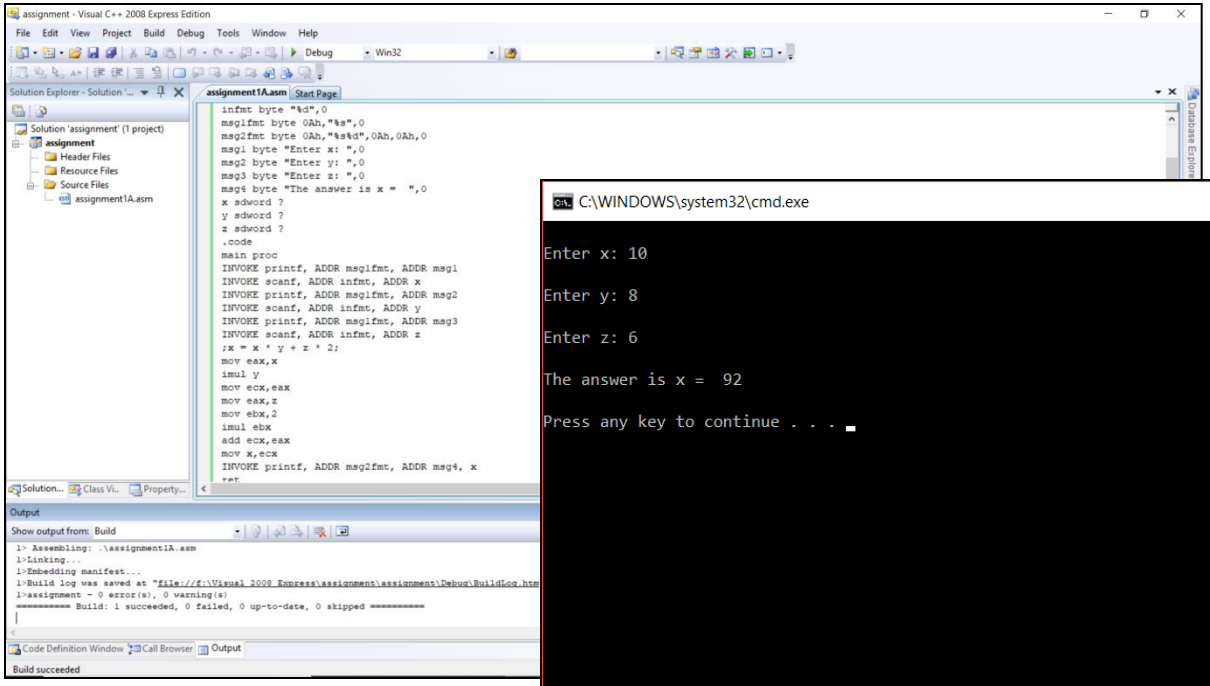
- Write the complete assembly programs for each assembly code segment from Exercise 2 and display the outputs.

Microprocessor Programming

Practical Assignments (Report)

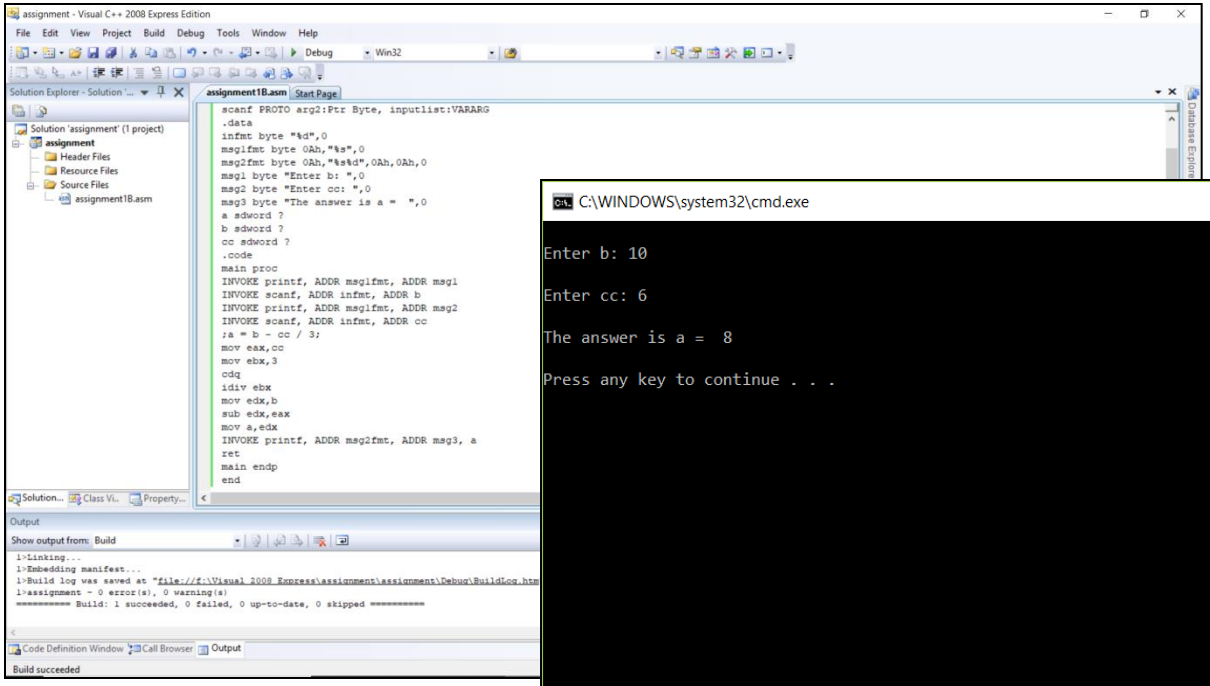
Assignment 1(A)

- A complete assembly program for the given MASM code segment and its output are as follows:



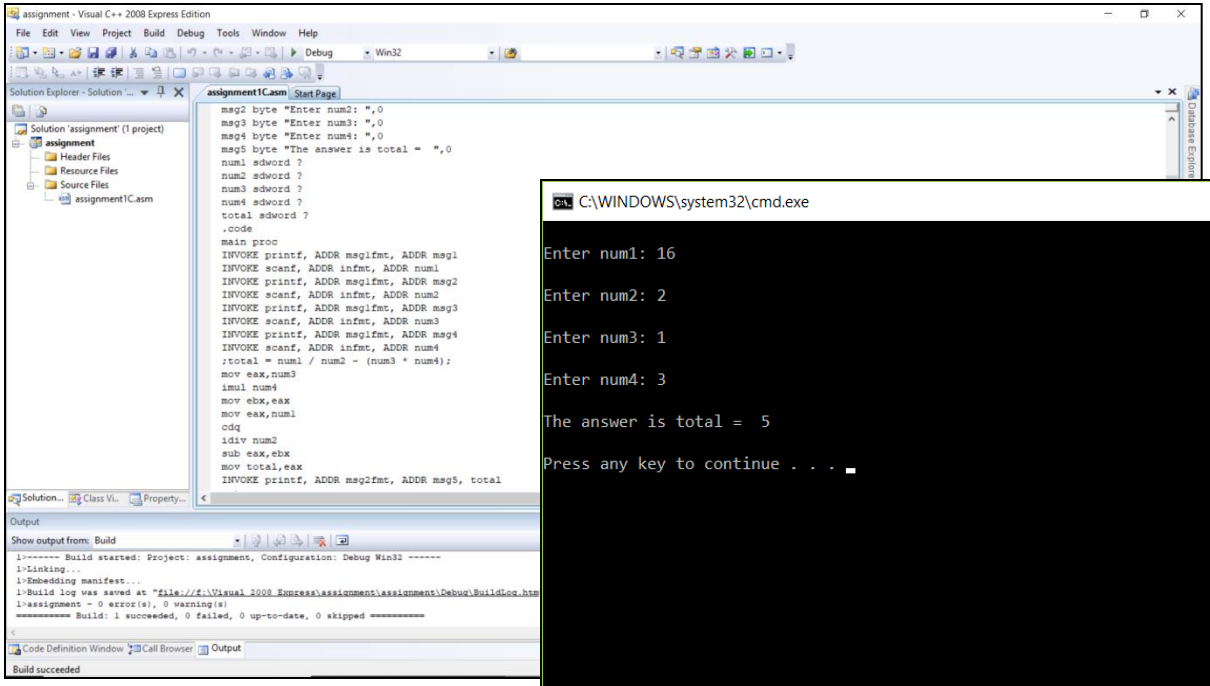
Assignment 1(B)

- A complete assembly program for the given MASM code segment and its output are as follows:



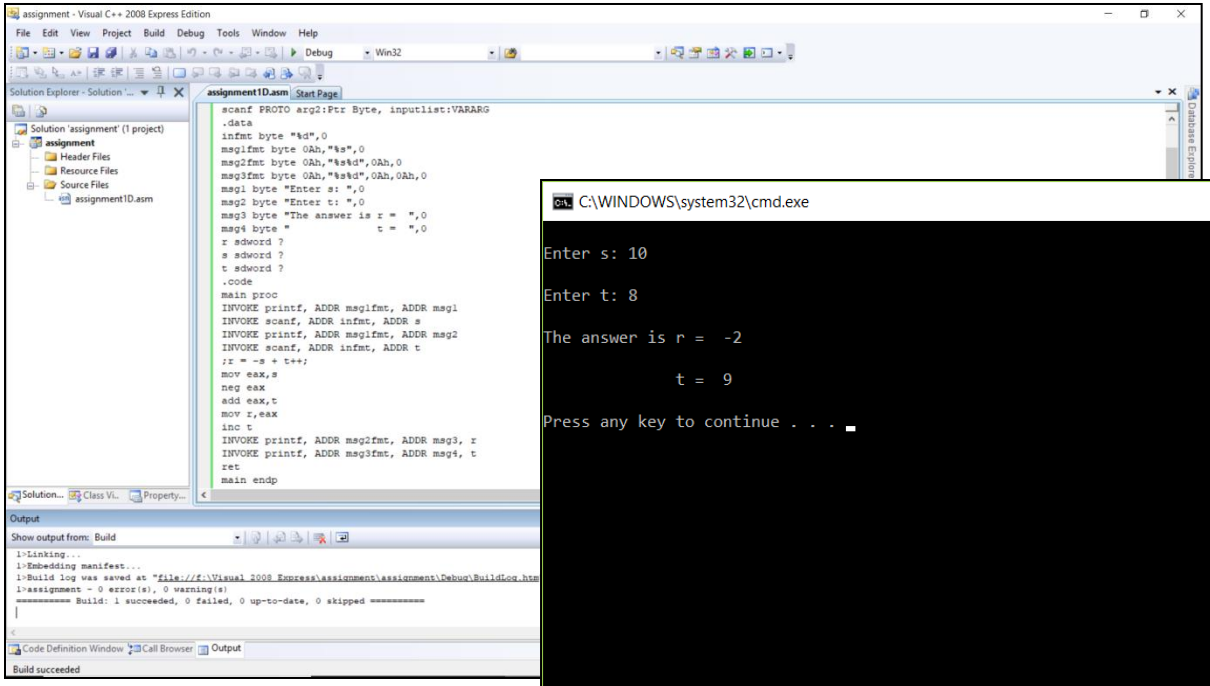
Assignment 1(C)

- A complete assembly program for the given MASM code segment and its output are as follows:



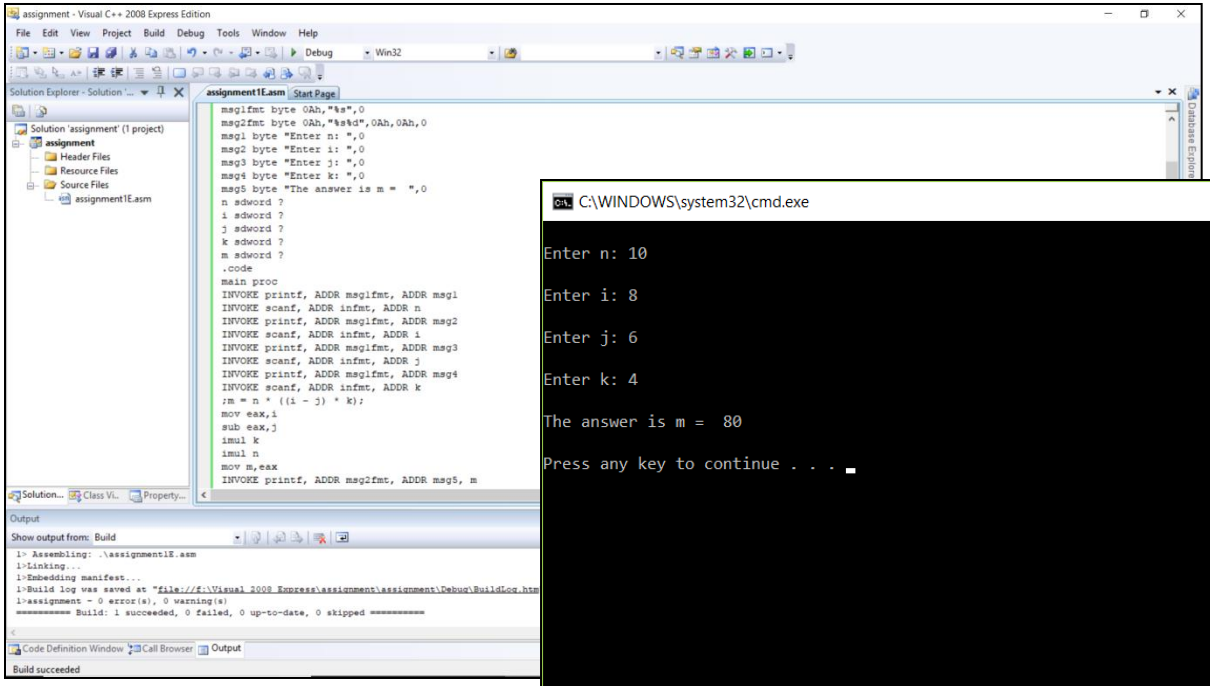
Assignment 1(D)

- A complete assembly program for the given MASM code segment and its output are as follows:



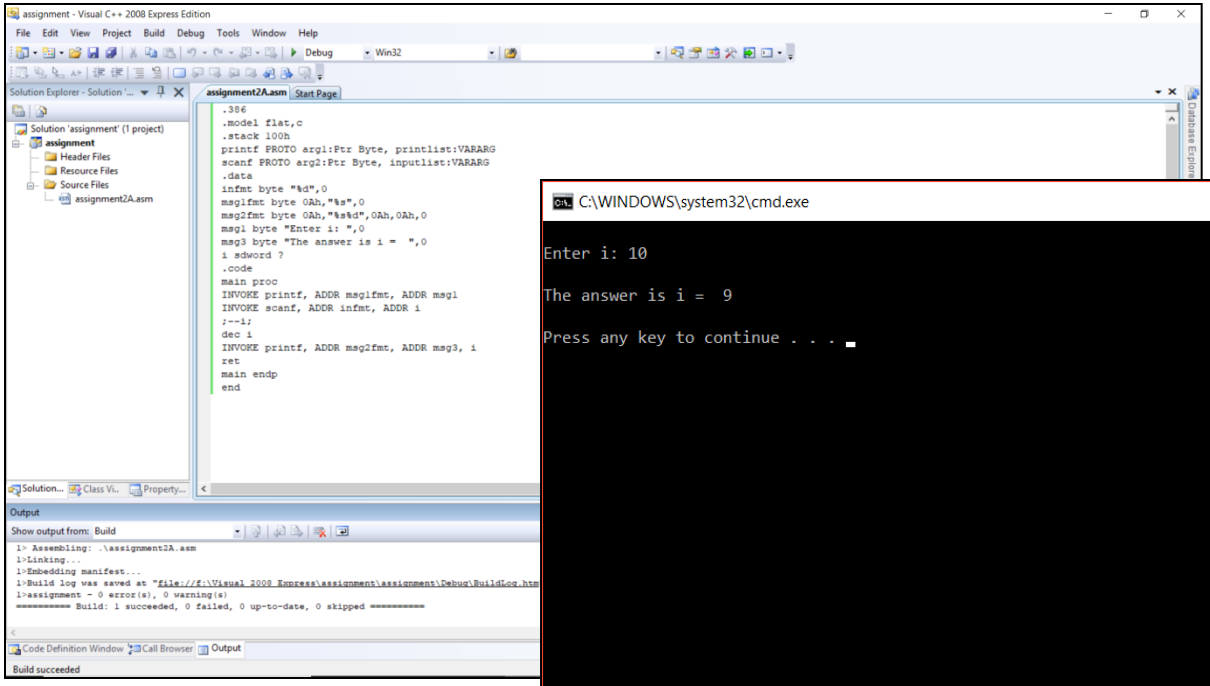
Assignment 1(E)

- A complete assembly program for the given MASM code segment and its output are as follows:



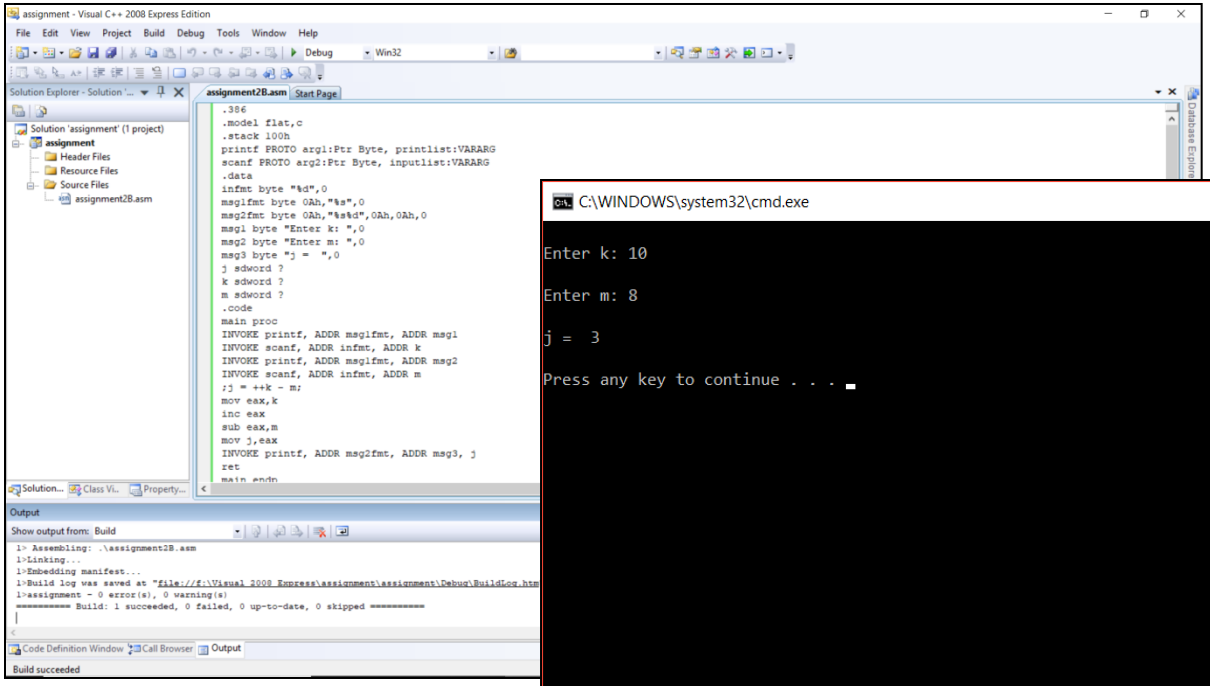
Assignment 2(A)

- A complete assembly program for the given MASM code segment and its output are as follows:



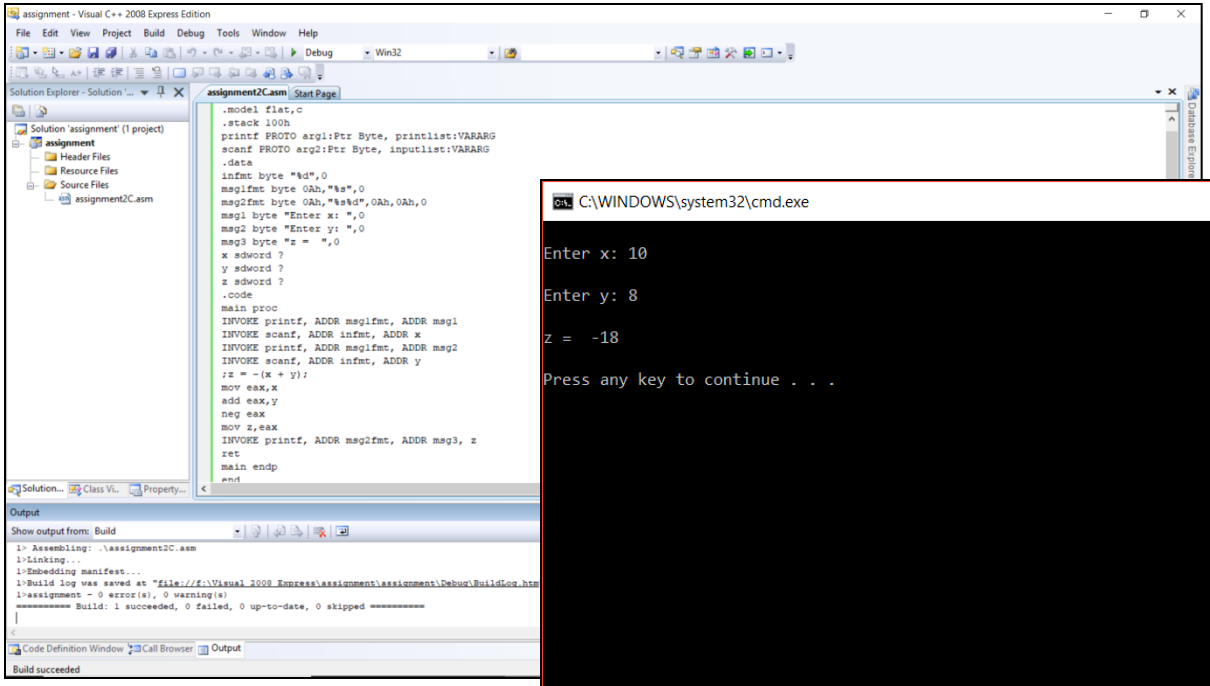
Assignment 2(B)

- A complete assembly program for the given MASM code segment and its output are as follows:



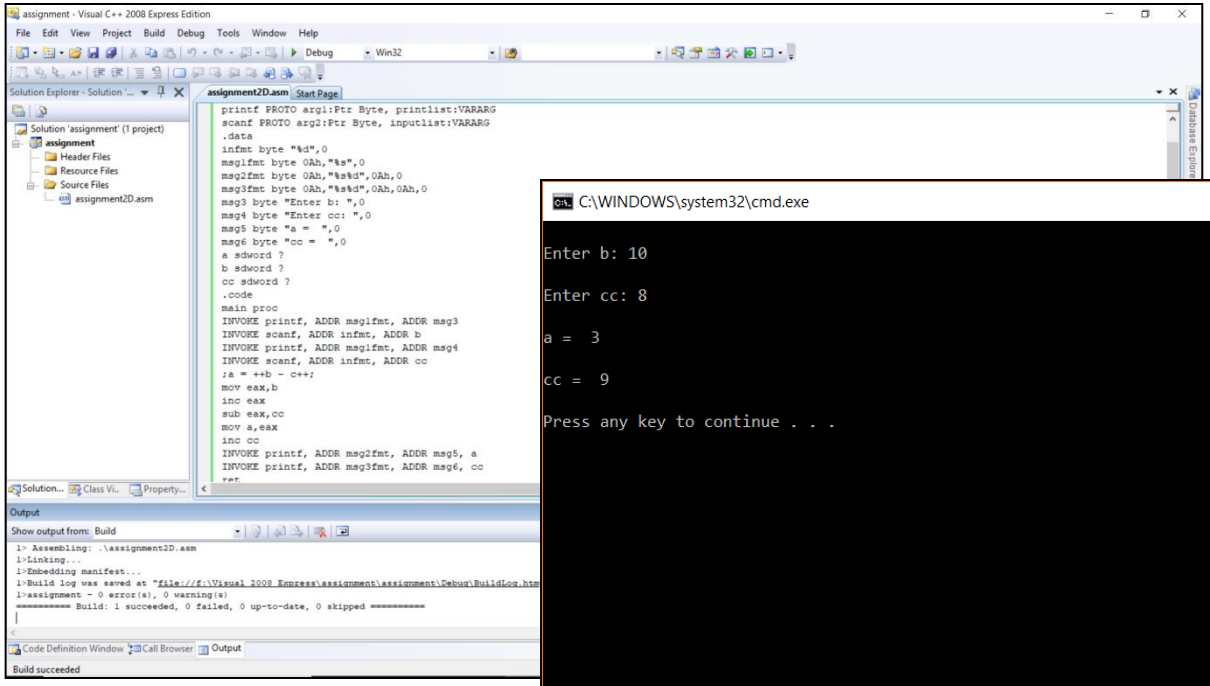
Assignment 2(C)

- A complete assembly program for the given MASM code segment and its output are as follows:



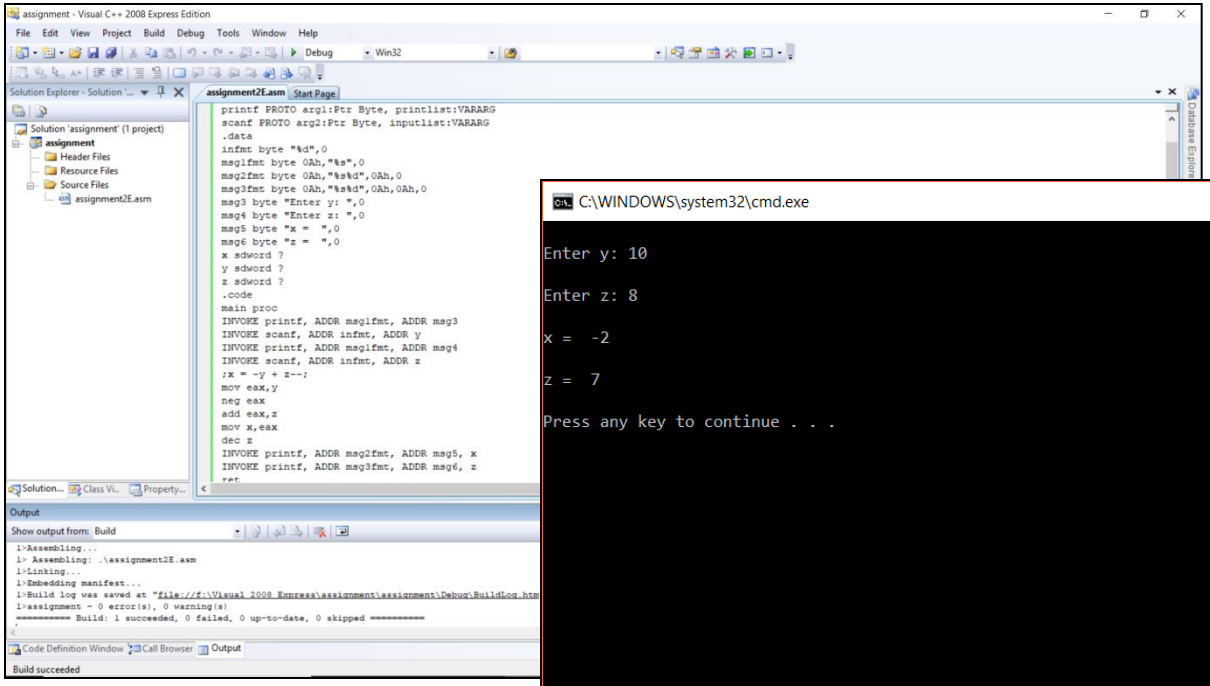
Assignment 2(D)

- A complete assembly program for the given MASM code segment and its output are as follows:



Assignment 2(E)

- A complete assembly program for the given MASM code segment and its output are as follows:



Next Lecture

- Introduction to Structures
- Selection Structures

Thank You