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

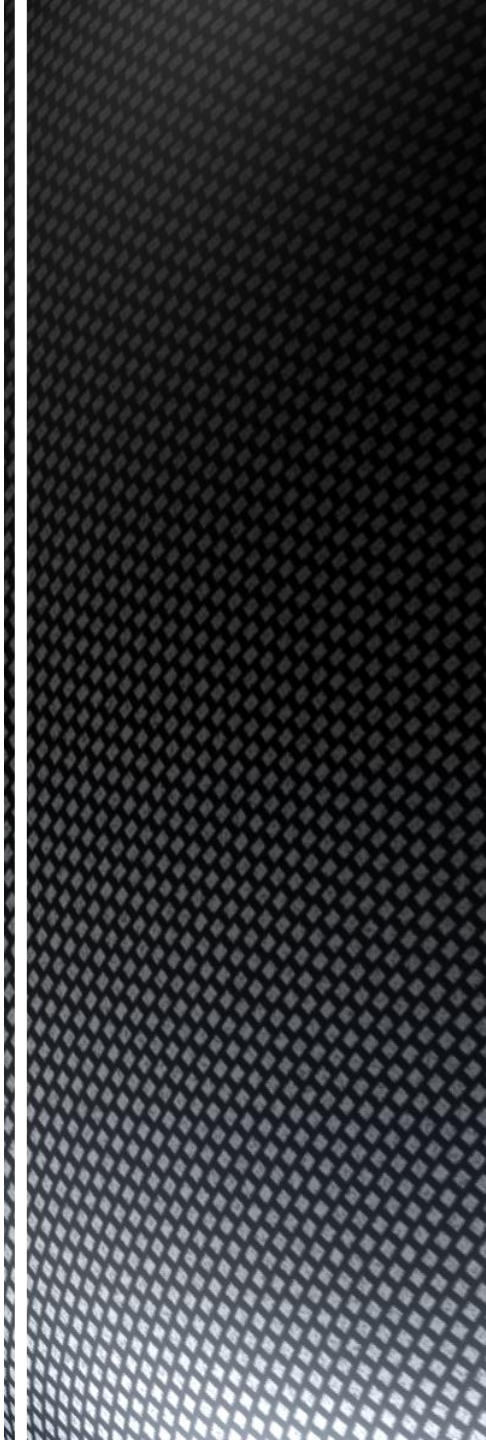
Microprocessor Programming

Lecture 3

Addition, Subtraction, Multiplication and Division

Contents

- Introduction
- Addition
- Subtraction
- Multiplication
- Division
- Complete Program
- Summary
- Practical Works
- Assignments



Introduction

- Similar to the high-level languages, assembly language can execute the arithmetic and logic instructions.
- Addition, subtraction, multiplication, division, comparison, negation, increment, and decrement are included in the arithmetic instructions.
- The logic instructions include AND, OR, Exclusive-OR, NOT, shifts, rotates, and the logical compare (TEST).

Addition Instruction

- In assembly language, the mnemonic used for addition instruction is add.
- The add instruction adds a source operand to a destination operand of the same size.
- The syntax is “add destination, source”.
- Source is unchanged by the operation, and the sum is stored in the destination operand.

Addition Instruction (Cont.)

- The table shows the valid add instructions.
- Note that an add mem,mem instruction does not appear in the list of instructions.

Table 1. Add instructions

Instruction	Meaning
add mem,imm	add the immediate value to memory
add reg,imm	add immediate value to the register
add mem,reg	add the contents of the register to memory
add reg,mem	add the contents of memory to the register
add reg,reg	add the contents of the source (second) register to the destination (first) register

Addition Instruction (Cont.)

- One of the simplest ways to learn how to perform arithmetic in assembly language is to first write the equation as a high-level statement.
- For the addition of the integer variable, namely sum, which is located in memory and the immediate value 7, we can implement the following C statement.

```
sum = sum + 7;
```

Addition Instruction (Cont.)

- Using the instruction format add mem,imm, the following assembly code segment implements the C statement above:

```
add sum,7        ; add the integer value 7 to sum
```

- In this segment, the immediate value 7 is added to the integer variable called sum and the addition result will be stored in sum.

Addition Instruction (Cont.)

- For the addition of the content of the register, namely eax, and the immediate value 5, we can implement the following C statement.

```
eax = 0;
eax = eax + 5;
```

- To implement this C code in assembly, the instruction format add reg, imm can be used.

Addition Instruction (Cont.)

- In MASM, we can implement as follow.

```
mov eax, 0
add eax, 5
mov temp, eax
INVOKE printf, temp
```

- Note that, the INVOKE directive can destroy the contents of eax, ecx and edx registers.
- Therefore, their contents always need to be copied into the other temporary location before the INVOKE directive.

Addition Instruction (Cont.)

- Another example for addition instruction is as follows.
- In C,

```
sum = 0;
sum = sum + num1;
```

- In MASM using the instruction format of add mem, reg,

```
mov sum,0          ; initialize sum to zero
mov eax, num1      ; load eax with the contents of num1
add sum, eax       ; add the contents of eax to sum
```

Addition Instruction (Cont.)

- For the addition of the two memory locations, assume that the integer variables num1 and num2, we can implement the following C statement.

```
sum = num1 + num2;
```

Addition Instruction (Cont.)

- This assembly code segment implements the C statement from above:

```
mov eax,num1      ; load eax with the contents of num1
add eax,num2      ; add the contents of num2 to eax
mov sum,eax       ; store eax in sum
```

- In the above segment, the contents of num1 are copied into the eax register, then the contents of num2 are added to eax, and lastly the contents of eax are copied into the variable sum.

Addition Instruction (Cont.)

- In the above segment, assume that num1 initially contains a 5 and num2 contains a 7.
- After the content of num1 is copied into the eax register, eax will contain 5.
- Then, the content of num2 are added to eax, and this time, eax will contain 12.
- Finally, the content of eax is copied into the variable sum and the sum will contain 12.

Subtraction Instruction

- The mnemonic used for subtraction instruction in assembly language is sub.
- The sub instruction subtracts a source operand from a destination operand of the same size.
- The syntax is “sub destination, source”.
- The result is stored in the destination operand.

Subtraction Instruction (Cont.)

- Similar to the addition instruction, the same formats also apply to the subtraction instruction as in Table 2.
- Note again that a memory to memory instruction does not exist.

Table 2. Sub instructions

Instruction	Meaning
sub mem,imm	subtract the immediate value from memory
sub reg,imm	subtract the immediate value from the register
sub reg,mem	subtract contents of memory from the register
sub mem,reg	subtract contents of the register from memory
sub reg,reg	subtract the contents of the source (second) register from the destination (first) register

Subtraction Instruction (Cont.)

- As an example of subtraction an immediate value from a memory location, we can implement as follow in C.

```
int difference = 10;
difference = difference - 3;
```

- In MASM, the format sub mem, imm can be used as follow.

```
difference sdword 10
sub difference,3          ; subtract 3 from difference
```

Subtraction Instruction (Cont.)

- The next example is subtraction an immediate value from a register and we can implement as follow in C.

```
eax = 12;
eax = eax - 3;
```

- In MASM, the format sub reg, imm can be used as follow.

```
mov eax, 12
sub eax, 3        ; subtract 3 from eax
```

Subtraction Instruction (Cont.)

- The following C code segment is for the difference between the two memory locations.

```
difference = num2 - num1;
```

- It can be implemented in assembly language as follows:

```
mov eax,num2          ; load num2 into eax
sub eax,num1          ; subtract num1 from eax
mov difference,eax    ; store answer in variable difference
```

Multiplication Instruction

- The mnemonic used for multiplication instruction in assembly language is mul.
- There exist two instructions for multiplication of unsigned and signed numbers.
- The mul instructions perform unsigned integer multiplication.
- The imul instructions perform signed integer multiplication.

Multiplication Instruction (Cont.)

- The multiplication instruction comes in three versions.
- The first version multiplies an 8-bit operand by the al register.
- The second one multiplies a 16-bit operand by the ax register.
- The third one multiplies a 32-bit operand by the eax register.
- These three formats accept register and memory operands, but not immediate operands.
- In this lecture, only the third version which used 32-bit operand will be emphasized.

Multiplication Instruction (Cont.)

- Unlike the mul instruction, imul preserves the sign of the product.
- This is done by sign extending the highest bit of the lower half of the product into the upper bits of the product.
- The x86 instruction set supports three formats for the imul instruction: one operand, two operands, and three operands.
- Here, only the one-operand versions of the instruction will be examined.

Multiplication Instruction (Cont.)

- In the one-operand versions of imul instruction, the multiplier and multiplicand must be the same size (32 bits).
- Firstly, the multiplicand must be loaded into the eax register.
- Then, the multiplier is placed into a register or can be located in a memory location.
- During execution of the imul instruction, the number in eax is multiplied by the number either in the specified register or in the memory location.

Multiplication Instruction (Cont.)

- While addition and subtraction are likely to be easy, multiplication can be just a little more complicated.
- For example, when multiplying the numbers 999 and 999 in base 10, the answer is 998,001, where there is potentially twice of the digits in the product.
- For the case of binary numbers, multiplying the numbers 111 and 111 results in the answer 110001, where again there are twice digits in the product.

Multiplication Instruction (Cont.)

- Therefore, if the two 32-bit registers are multiplied together, the result could take up 64 bits.
- As a result, when multiplication occurs in the computer, there needs to be room for the extra digits.
- For this purpose, the one-operand formats store the product in edx:eax register pair in which the size is 64 bits in total.

Multiplication Instruction (Cont.)

- The valid formats of the imul instruction are shown in Table 3.

Table 3. imul instructions

Instruction	Meaning
imul mem	multiply eax by an integer in a register
imul reg	multiply eax by an integer in a memory location

Multiplication Instruction (Cont.)

- Here is a simple example for multiplication of the two variables in C instruction.

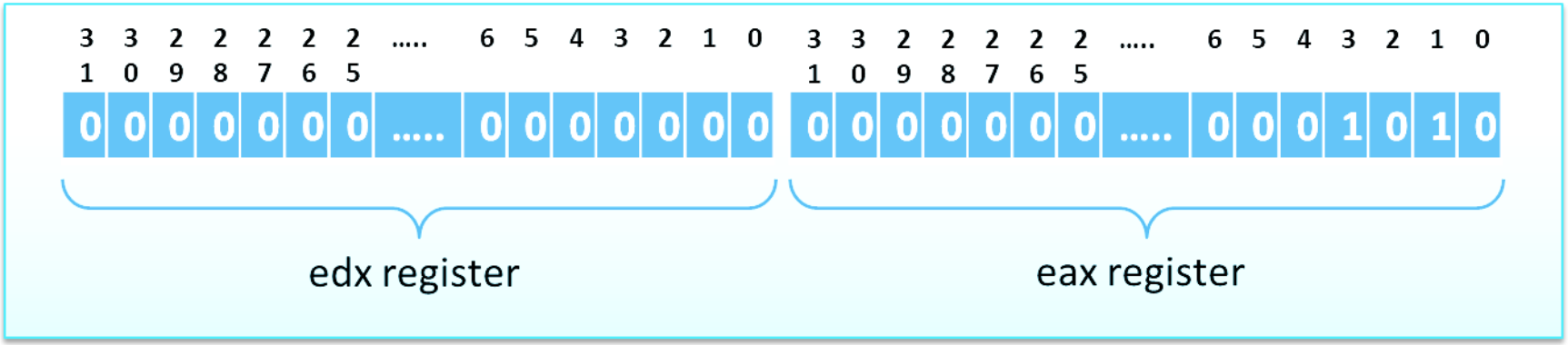
```
product = num1 * num2;
```

- The above code segment in assembly language is:

```
mov eax,num1        ; load eax with the contents of num1
imul num2           ; multiply eax by num2
mov product,eax     ; store eax in product
```

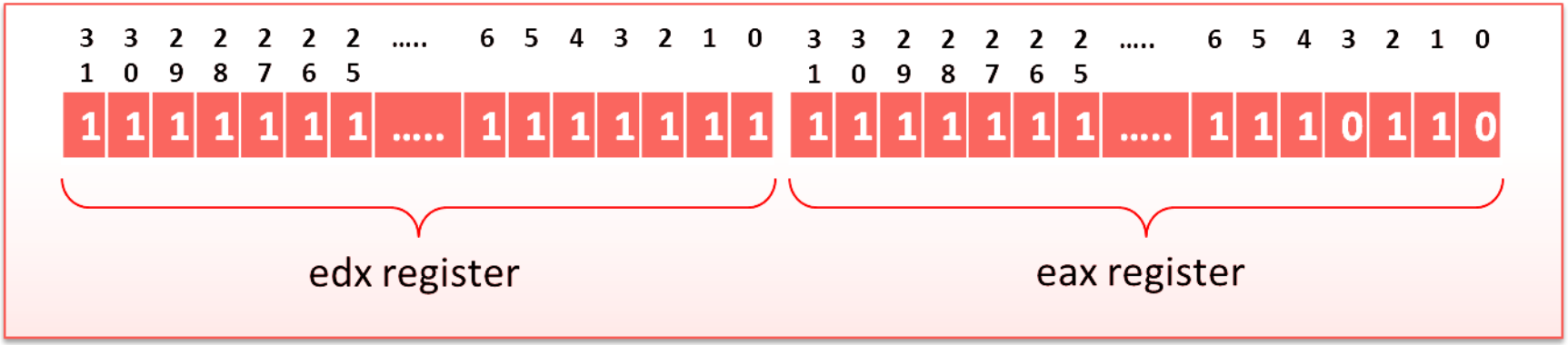
Multiplication Instruction (Cont.)

- Assume that num1 contains a positive 2 and num2 contains a positive 5.
- The results in the edx:eax register pair would be that the 0 located in the 31st bit (leftmost bit) of the eax register would be copied or propagated throughout all 32 bits of the edx register as shown in figure.



Multiplication Instruction (Cont.)

- Furthermore, assume that num1 contains a negative 2 and num2 contains a positive 5.
- Then the results in the edx:eax register pair would have been that the 1 located in the 31st bit of the eax register would have been copied or propagated throughout all 32 bits of the edx register as shown in figure.



Multiplication Instruction (Cont.)

- For multiplication of the variable and immediate value, we can implement in C as:

```
product = num1 * 2;
```

- But, there is no provision for an immediate operand and that the use of the eax register for the multiplicand is implied.
- The above C code segment in assembly language is:

```
mov eax,num1        ; load eax with the contents of num1
mov ebx,2           ; load ebx with the value 2
imul ebx            ; multiply eax by ebx
mov product,eax     ; store eax in product
```

Division Instruction

- The mnemonic used for division instruction in assembly language is div.
- The div instructions perform division for unsigned numbers.
- The idiv instructions perform division for signed numbers.
- the idiv instruction follows the same format as for the previously introduced one-operand imul instruction.

Division Instruction (Cont.)

- The formats of the idiv instruction are shown in Table 4.

Table 4. idiv instructions

Instruction	Meaning
idiv mem	divide the edx:eax register pair by memory
idiv reg	divide the edx:eax register pair by a register

Division Instruction (Cont.)

- In division, the answer (quotient) and remainder can be smaller than the original number to be divided (the dividend).
- The dividend must be initially placed in the edx:eax pair prior to using the idiv instruction.
- After execution of the idiv instruction, the quotient is in the eax register and the remainder is in the edx register.

Division Instruction (Cont.)

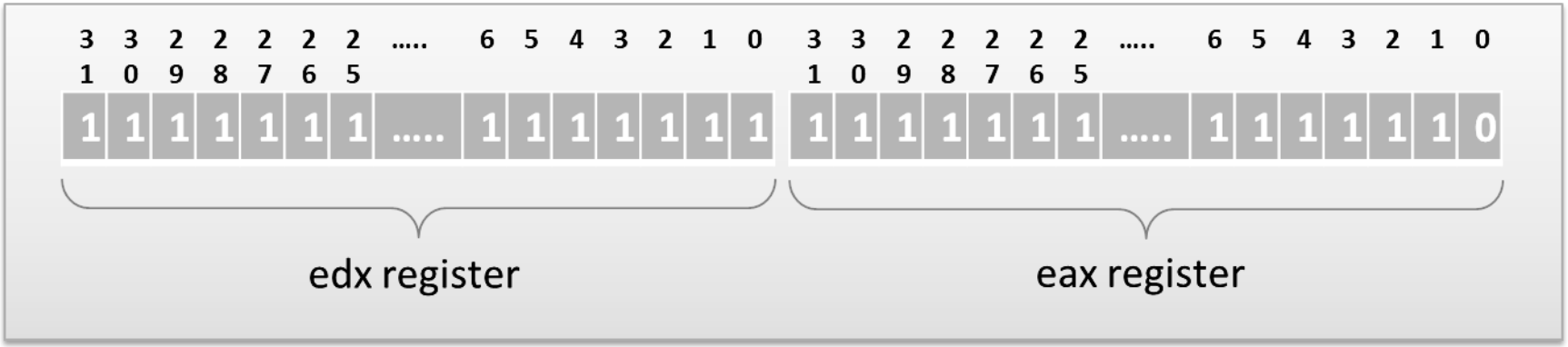
- There exists a special instruction to propagate or extend the sign bit from a smaller register to a larger register.
- The instruction is shown in Table 5.

Table 5. Convert Instructions

Opcode	Meaning	Description
cdq	Convert double to quad word	Extends sign from eax to edx:eax pair

Division Instruction (Cont.)

- The cdq instruction allows the sign bit, whether a 0 or a 1, to be propagated throughout the edx register.
- Assume that the eax register originally contains a -2.
- Then, the sign bit (a 1 in bit position 31) of eax is copied into each bit position of the edx register as illustrated in Figure.



Division Instruction (Cont.)

- As an example for division, the following C code segment

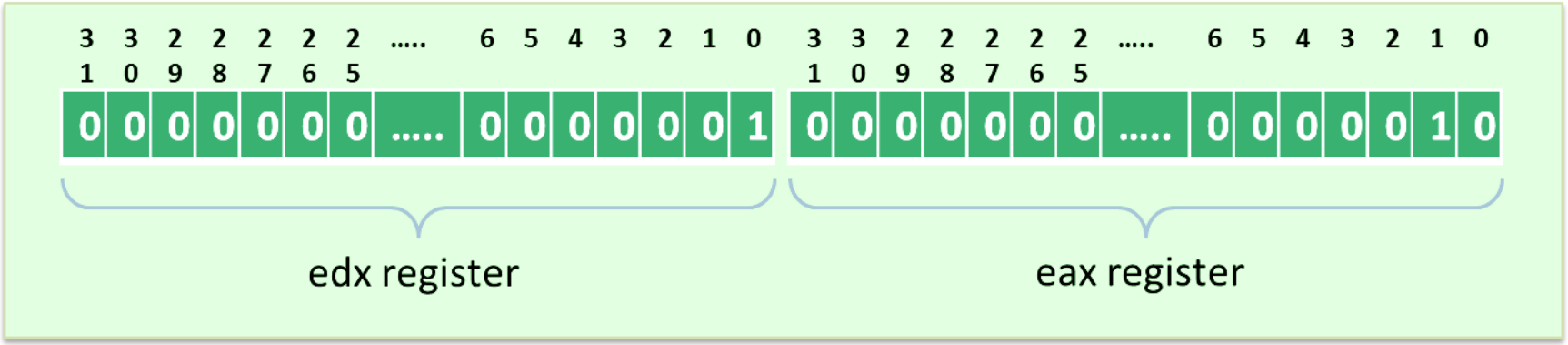
```
answer = number / amount;
```

- can be rewritten in assembly as:

```
mov eax,number      ; load eax with number
cdq                 ; propagate sign bit into the edx register
idiv amount         ; divide edx:eax by amount
mov answer,eax      ; store eax in answer
```

Division Instruction (Cont.)

- Assume that number contains a positive 5 and amount contains a positive 2.
- The contents of the edx:eax pair would be as follows after the execution of the above code segment.
- The quotient (2) is in the eax register and the remainder (1) is in the edx register in binary format as shown in Figure.



Complete Program: Implementing I/O and Arithmetic

- Write a program to calculate the number of amperes given the number of volts and ohms?

```
#include <stdio.h>
int main(){
    int volts, ohms, amperes;
    printf("\n%s", "Enter the number of volts: ");
    scanf("%d", &volts);
    printf("%s", "Enter the number of ohms: ");
    scanf("%d", &ohms);
    amperes = volts / ohms;
    printf("\n%s%d\n\n","The number of amperes is:",amperes);
    return 0;
}
```

Complete Program: Implementing I/O and Arithmetic (Cont.)

- This program is implemented in assembly language as follows:

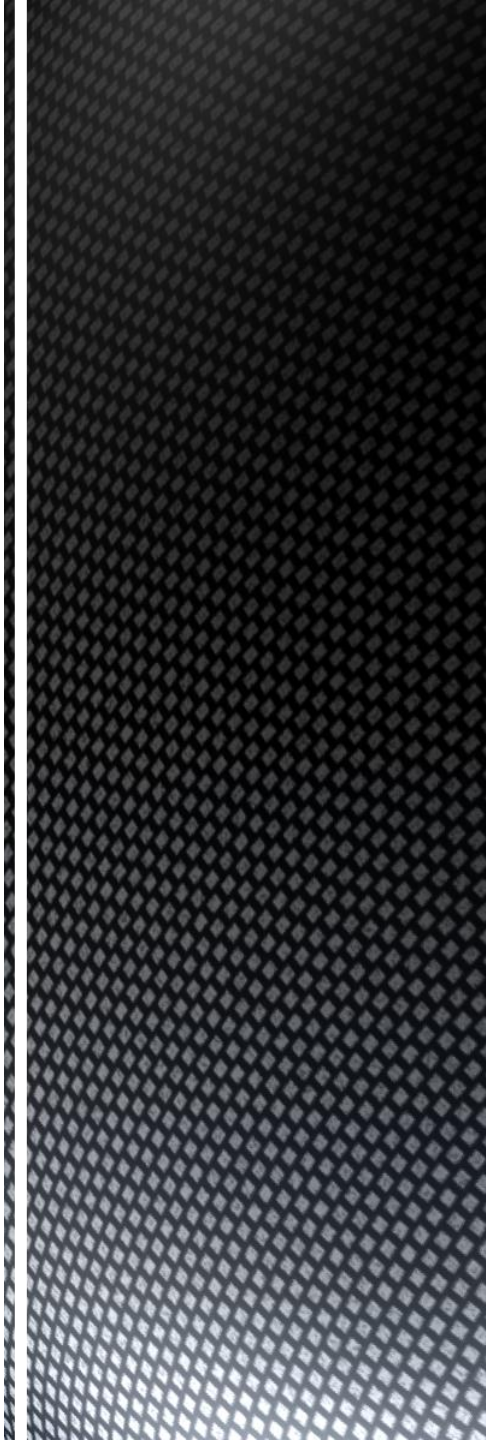
```
            .386
            .model flat, c
            .stack 100h
printf      PROTO arg1:Ptr Byte, printlist:VARARG
scanf       PROTO arg2:Ptr Sdword, inputlist:VARARG
            .data
in1fmt      byte "%d",0
msg1fmt     byte 0Ah,"%s",0
msg2fmt     byte "%s",0
msg3fmt     byte 0Ah,"%s%d",0Ah,0Ah,0
msg1        byte "Enter the number of volts: ",0
msg2        byte "Enter the number of ohms: ",0
msg3        byte "The number of amperes is: ",0
volts       sdword ?        ; number of volts
ohms        sdword ?        ; number of ohms
amperes     sdword ?        ; number of amperes
            .code
main        proc
            INVOKE printf, ADDR msg1fmt, ADDR msg1
            INVOKE scanf, ADDR in1fmt, ADDR volts
            INVOKE printf, ADDR msg2fmt, ADDR msg2
            INVOKE scanf, ADDR in1fmt, ADDR ohms
            ; amperes = volts/ohms
            mov eax,volts     ; load volts into eax
            cdq               ; extend the sign bit
            idiv ohms         ; divide eax by ohms
            mov amperes,eax   ; store eax in amperes
            INVOKE printf, ADDR msg3fmt, ADDR msg3, amperes
            ret
main        endp
            end
```

Summary

- The mnemonics used for arithmetic instructions are add for addition, sub for subtraction, imul for signed integer multiplication and idiv for signed integer division.
- The size of the product is twice of the size of multiplier and multiplicand.
- Before idiv instruction, the cdq instruction is needed to be executed for sign bit extension.

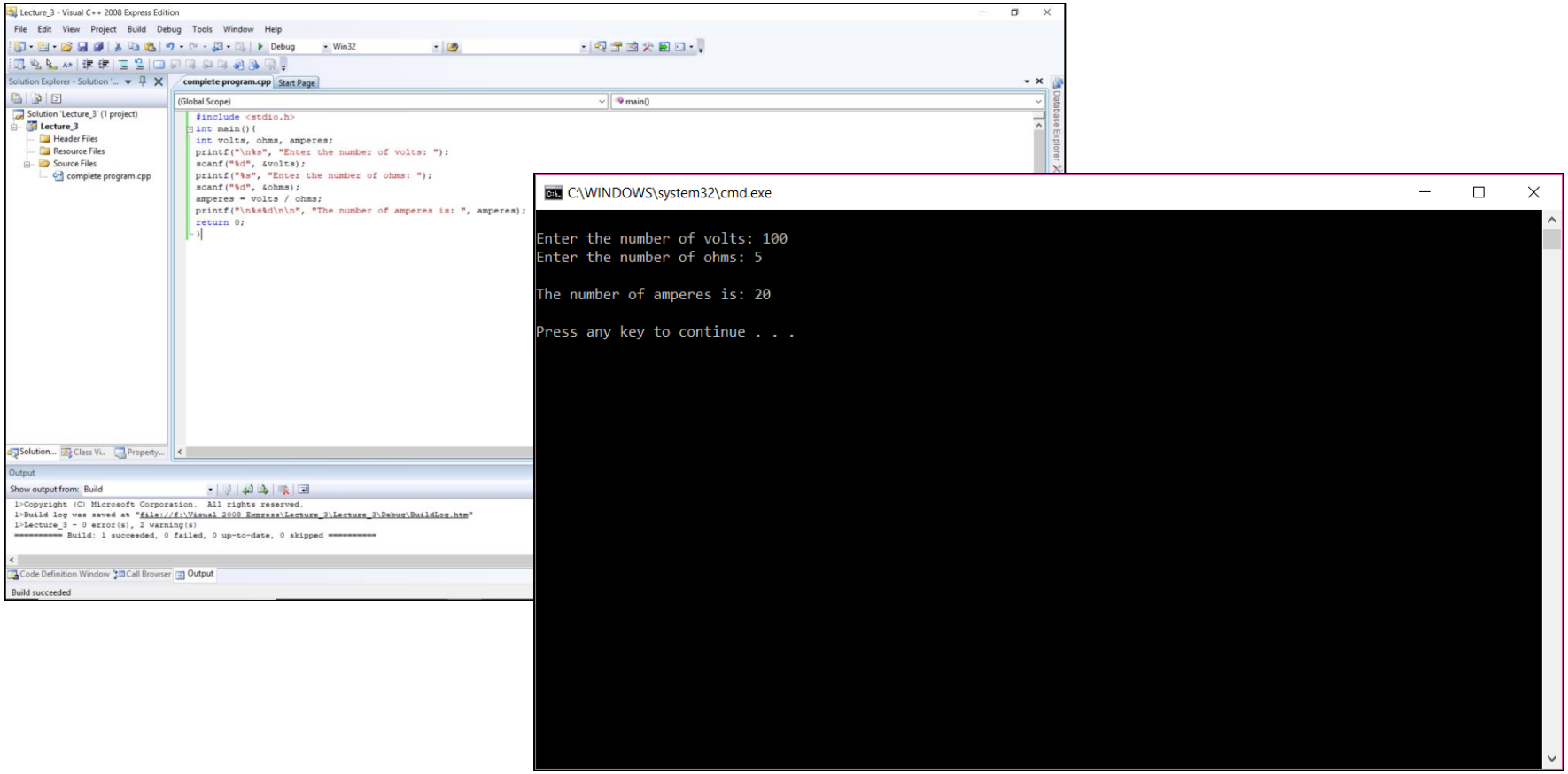
Microprocessor Programming

Practical Work:
Implementing I/O and Arithmetic



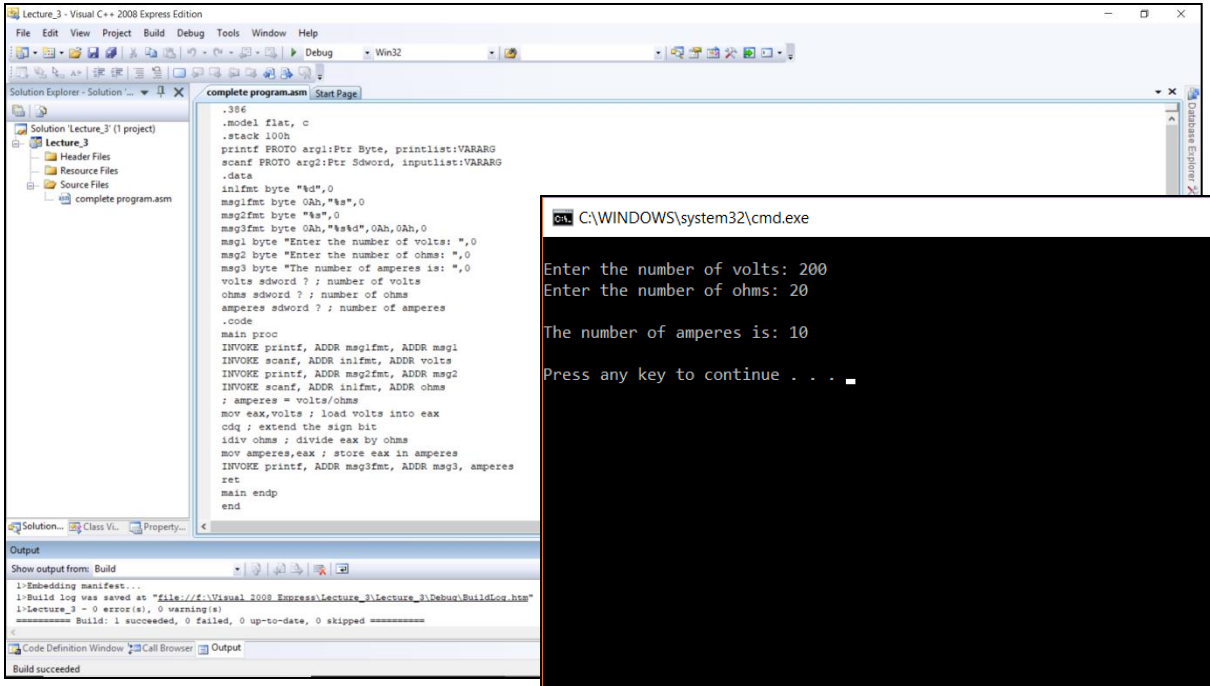
Implementing I/O and Arithmetic

- This implementation of C program is to calculate the number of amperes for the given number of volts and ohms.



Implementing I/O and Arithmetic (Cont.)

- This implementation is to calculate the number of amperes for the given the number of volts and ohms in MASM.



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

- Write a complete assembly language program to implement the following C program:

```
#include <stdio.h>
int main(){
    int number;
    printf("\n%s","Enter an integer: ");
    scanf("%d",&number);
    number=7-number*3;
    printf("\n%s%d\n\n","The integer is: ",number);
    return 0;
}
```

Assignment 2

- Given Ohm's law from the complete program at the end of this chapter and Watt's law as W= IE, where W stands for the number of watts, write a complete assembly language program to prompt for and input the number amperes and ohms, and then calculate both the number of volts and number of watts. The form of the input and output can be found below, and as always be careful with the vertical and horizontal spacings:

  Input and Output

  Enter the number of amperes: 5

  Enter the number of ohms: 4

  The number of volts is: 20

  The number of watts is: 100

Assignment 3

- Write a complete assembly language program to prompt for and input the temperature in degrees Fahrenheit, calculate the degrees in Celsius, and then output the degrees in Celsius. The equation to be used is C = (F–32)/9∗5, where C stands for Celsius and F stands for Fahrenheit. Note that the answer will be off slightly due to using integers and be very careful to use the proper order of operations. The form of the input and output can be found below. Be sure to use proper vertical and horizontal spacings:

  Input and Output

  Enter the degrees in Fahrenheit: 100

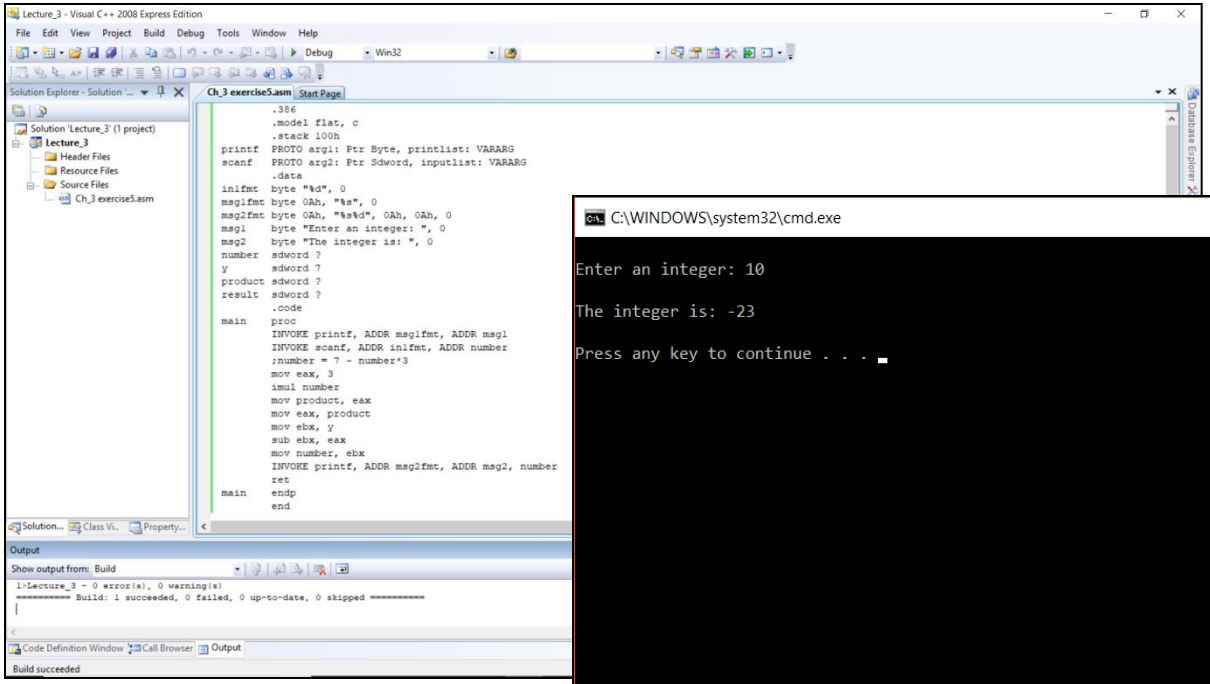
  The degrees in Celsius is: 35

Microprocessor Programming

Practical Assignments (Report)

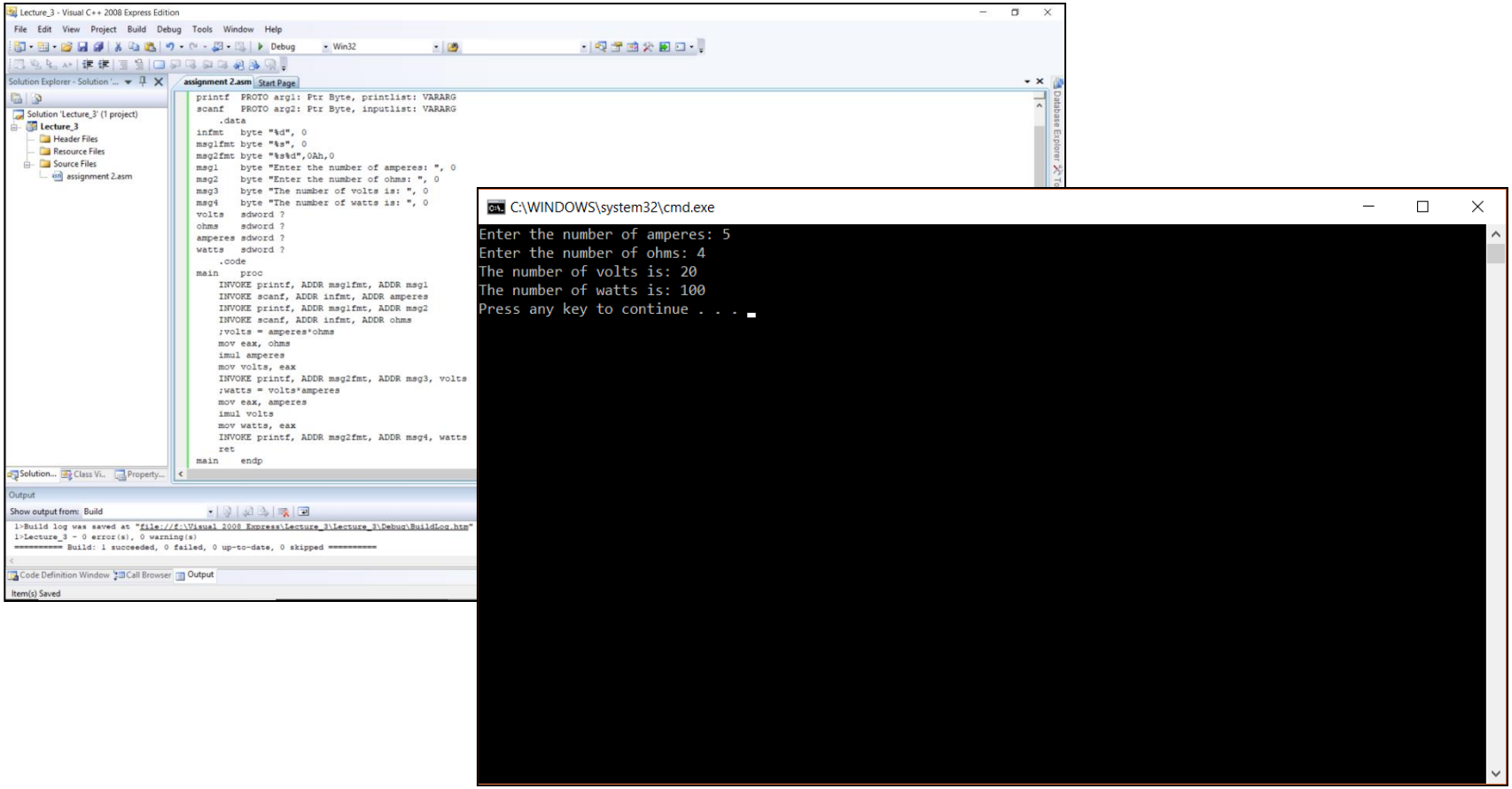
Assignment 1

- A complete assembly program to implement for the given C program and its output are as follows:



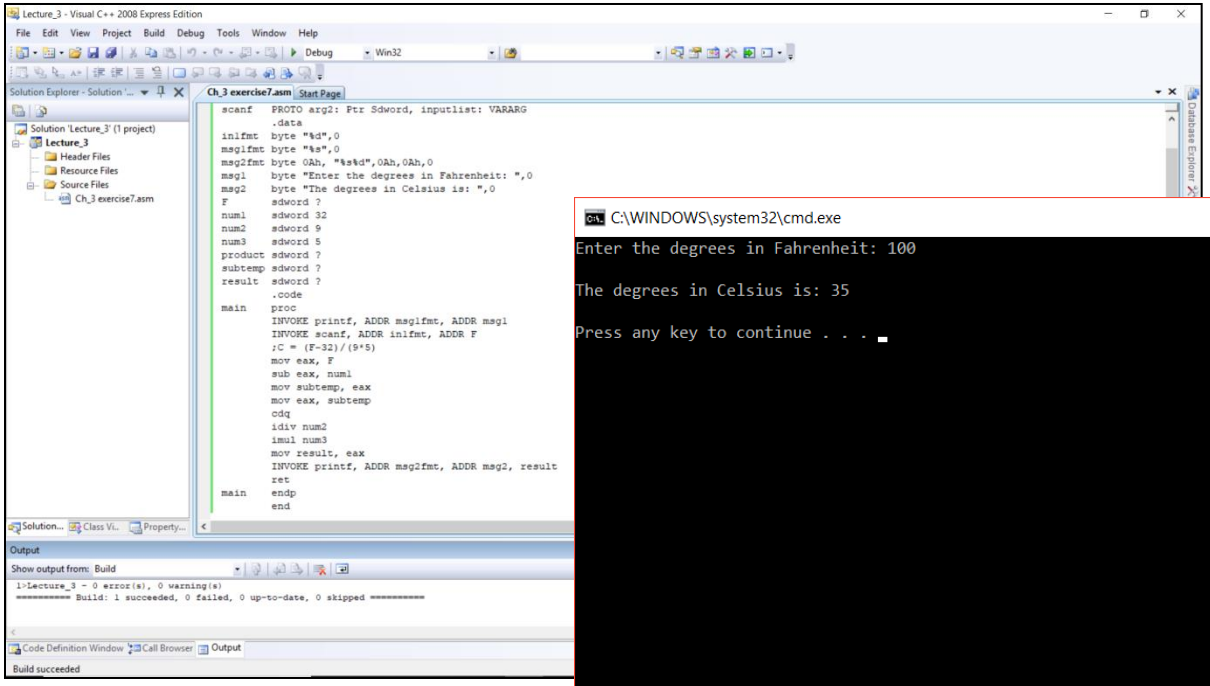
Assignment 2

- A complete assembly program for the given problem and its output are as follows:



Assignment 3

- A complete assembly program for the given problem and its output are as follows:



Next Lecture

- Introduction to Operators
- Unary Operators
- Binary Operators
- Order of Operators

Thank You