Microprocessor Programming

Dr. Tin Ni Ni Kyaw

Ph. D (Kumamoto University, Japan)

Associate Professor

Department of Computer Engineering and Information Technology

Yangon Technological University

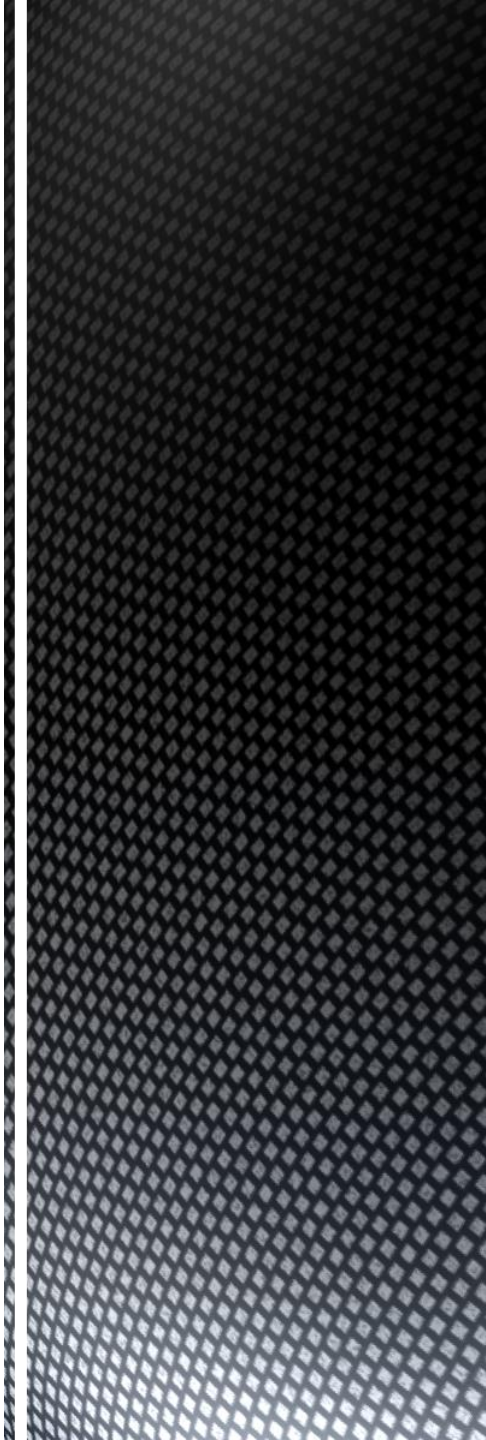
Yangon, Myanmar

Microprocessor Programming

Lecture 6

Iteration Structures

Contents



Introduction

- There are many different types of iteration structures in a high-level programming language.
- Similarly, assembly language also has the corresponding iteration structures, such as the pre-test, post-test, and fixed-iteration loop structures.
- Depending on the circumstances, we should use the best structure for the task.

Pre-Test Loop Structure

- The count-controlled pre-test while loop is the most versatile loop.
- A number of tasks can be performed in the body of the loop.
- The basic structure of this loop can be found in C code segment below:

```
i=1;
while(i<=3){
        i++;
}
```

Pre-Test Loop Structure (Cont.)

- In MASM, .while and .endw directives can simplify the implementation of the while loop structure as shown below:

```
mov i,1
.while i<=3
        inc i
.endw
```

- For any number of statements in the body of the loop, the structure must end with the .endw directive.

Pre-Test Loop Structure (Cont.)

- In another way, the while structure can also be implemented by using compare statement and jump statements.

```
                mov i,1
while01:        cmp i,3
                jg endw01
                inc i
                jmp while01
endw01:         nop
```

- Notice the inclusion of the unconditional jmp while01 at the bottom of the loop, because without it, the loop would execute the body of the loop only once.

Pre-Test Loop Structure (Cont.)

- The following C code is an example of the implementation of while loop structure together with if-then structure.

```
ans=0;
if(x!=0) {
        i=1;
        while(i<=y) {
                ans=ans+x;
                i++;
        }
}
```

Pre-Test Loop Structure (Cont.)

- The equivalent MASM code segment can be written as follow:

```
mov ans,0          ; initialize ans to 0
.if x != 0
mov i,1            ; initialize i to 1
mov eax,y          ; load eax with y for while
.while i<= eax
mov eax,ans        ; load eax with ans
add eax,x          ; add eax to ans
mov ans,eax        ; store eax in ans
mov eax,y          ; reload eax with y for while
inc i              ; increment i by 1
.endw
.endif
```

Pre-Test Loop Structure (Cont.)

- In the program, i is a loop control variable.
- So, alternatively, an ecx register which is a counter register can be used instead of i.
- Then, the value in ecx can simply be moved into the variable i at the end of the segment to get the final value.

Pre-Test Loop Structure (Cont.)

- The equivalent MASM code segment using ecx register is as follow:

```
mov ans,0          ; initialize ans to 0
.if x!= 0
mov ecx,1          ; initialize ecx to 1
.while ecx<=y
mov eax,ans        ; load eax with ans
add eax,x          ; add x to ans
mov ans,eax        ; store eax in ans
inc ecx            ; increment ecx by one
.endw
mov i,ecx          ; store ecx in i
.endif
```

Pre-Test Loop Structure (Cont.)

- To compare the three coding styles: the C code, MASM codes using high-level directives and without using high-level directives for pre-test loop structure, we can see as follows:

```
i=1;
while(i<=3){
        i++;
}
```

```
mov i,1
.while i<=3
        inc i
.endw
```

```
            mov i,1
while01:    cmp i,3
            jg endw01
            inc i
            jmp while01
endw01:     nop
```

Post-Test Loop Structures

- In C programming language, it has a post-test loop structure called the do-while.
- The unique feature of post-test loops is that the body of the loop is executed at least one time.
- In MASM, the post-test loop structure is implemented by using the .repeat and .until directives.

Post-Test Loop Structures (Cont.)

- The C code for do-while loop is given on the left and the corresponding assembly language appears on the right:

```
i=1;
do {
        i++;
} while (i<=3);
```

```
mov i,1
.repeat
inc i
.until i>3
```

- Note that instead of i<=3, the .until has i>3 in MASM.
- The relation of the jump is reversed from the one in the do-while.

Post-Test Loop Structures (Cont.)

- The implementation of this loop without MASM high-level directives using compares, jumps, and labels is shown below.
- The relation of the jump is the same as the one in the do-while loop in C code.

```
mov i,1
repeat01:   nop
            inc i
            cmp i,3
            jle repeat01
endrpt01:   nop
```

Post-Test Loop Structures (Cont.)

- For the next example, the C code and the equivalent MASM code segment are as follows:

```
ans=0;
if (y!=0) {
        i=1;
        do {
                ans=ans+x;
                i++;
        } while (i<=y);
}
```

```
mov ans,0
.if y!=0
mov ecx,1
.repeat
mov eax,ans
add eax,x
mov ans,eax
inc ecx
.until ecx>y
mov i,ecx
.endif
```

Post-Test Loop Structures (Cont.)

- To compare the three coding styles: the C code, MASM codes using high-level directives and without using high-level directives for post-test loop structure, we can see as follows:

```
i=1;
do {
        i++;
} while (i<=3);
```

```
mov i,1
.repeat
inc i
.until i>3
```

```
mov i,1
repeat01:   nop
            inc i
            cmp i,3
            jle repeat01
endrpt01:   nop
```

Fixed-Iteration Loop Structures

- As in many high-level languages, there usually exists a fixed-iteration loop structure often called a for loop structure.
- It can be used when a loop needs to loop only a fixed number of times.
- In C, an example of such a loop is the for loop, where the braces are optional when there is only one statement:

```
for(i=1;i<=3;i++)
{
// body of loop
}
```

Fixed-Iteration Loop Structures (Cont.)

- In MASM, the directives that can be used for this task are the .repeat and .untilcxz directives.
- The .repeat and .untilcxz directives use the ecx register as a counter.
- The .untilcxz directive performs two tasks. It
  1. decrements the ecx register by 1 and then
  2. jumps to the .repeat directive when ecx is not equal to 0.
- In other words, it loops until the ecx register equals 0 (cxz).

Fixed-Iteration Loop Structures (Cont.)

```
mov ecx,3
.repeat
; body of the loop
.untilcxz
```

- In this MASM code segment, first, ecx register is loaded with the number of times the body of the loop should be executed.
- Then, each time the .untilcxz directive is executed, the value of the ecx register is decremented by 1 and then compared to 0.
- If the value is not equal to 0, the loop repeats.
- If the value is 0, the flow of control is passed onto the instruction immediately following the .untilcxz directive.

Fixed-Iteration Loop Structures (Cont.)

- To implement the MASM code segment without using high-level directives, the loop instruction works the same as .repeat and .untilcxz directives.

```
mov ecx,3
for01:      nop
            ; body of the loop
            loop for01
endfor01:   nop
```

- The loop instruction decrements ecx register by 1, branches to the label indicated in the operand field when ecx is not equal to 0, and falls through otherwise.

Fixed-Iteration Loop Structures (Cont.)

- To compare the three coding styles: the C code, MASM codes using high-level directives and without using high-level directives for fixed-iteration loop structures, we can see as follows:

```
for(i=1;i<=3;i++)
{
// body of loop
}
```

```
mov ecx,3
.repeat
; body of loop
.untilcxz
```

```
mov ecx,3
for01:      nop
            ;body of loop
            loop for01
endfor01:   nop
```

Post-Test Loop Structure (Cont.)

- As another example, the C code and the equivalent MASM code segment are as follow:

```
ans =0;
if (y != 0)
for(i=1; i<=y;i++)
        ans = ans + x;
```

```
mov ans,0
.if y != 0
mov ecx,y
.repeat
mov eax,ans
add eax, x
mov ans,eax
.untilcxz
.endif
```

Loops and Input/Output

- This C code segment is for entering exactly 10 integers and calculating the sum of the integers and then displaying the result.

```
sum=0;
for(i=1; i<=10; i++) {
        printf("%s","Enter an integer: ");
        scanf("%d",&num);
        sum=sum+num;
}
printf("\n%s%d\n\n","The sum is ",sum);
return 0;
```

Loops and Input/Output (Cont.)

- The equivalent segment in assembly is shown below.

```
.data
msg1 byte "Enter an integer: ",0
msg2 byte "The sum is ",0
.code
mov sum,0
mov ecx,10
.repeat
mov temp,ecx
INVOKE printf, ADDR msg1fmt, ADDR msg1
INVOKE scanf, ADDR in1fmt, ADDR num
mov eax,sum
add eax,num
mov sum,eax
mov ecx,temp
.untilcxz
INVOKE printf, ADDR msg2fmt, ADDR msg2, sum
```

Loops and Input/Output (Cont.)

- Remember that the INVOKE directive can destroy the eax, ecx, and edx registers.
- Since the .repeat-.untilcxz directive uses the ecx register, it needs to save and restore its value.
- For this reason, the value of ecx is stored in a memory location called temp at the top of the loop and then the value of ecx is restored at the bottom of the loop.

Loops and Input/Output (Cont.)

- However, if the user wants to decide how many integers to be summed, the number of integers to be input may not be predefined in the program.
- In such cases, we need to add some statements such as a prompt for asking the user for the number of integers.
- To do this, we can add those statements before the loop.

Loops and Input/Output (Cont.)

- The following implementation includes the required actions mentioned in the previous section.

```
.data
msg0 byte "Enter the number of integers to input: ",0
msg1 byte "Enter an integer: ",0
msg2 byte "The sum is ",0
.code
mov sum,0
INVOKE printf, ADDR msg1fmt, ADDR msg0
INVOKE scanf, ADDR in1fmt, ADDR count
mov ecx,1
.while ecx<=count
mov temp,ecx
INVOKE printf, ADDR msg1fmt, ADDR msg1
INVOKE scanf, ADDR in1fmt, ADDR num
mov eax,sum
add eax,num
mov sum,eax
mov ecx,temp
inc ecx
.endw
INVOKE printf, ADDR msg2fmt, ADDR msg2, sum
```

Nested Loops

- An example for nested loops by the C code segment is below on the left and by the assembly code using high-level directives is on the right:

```
i = 1;
while(i<=2) {
        j=1;
        while(j<=3) {
                j++;
        }
        i++;
}
```

```
mov i,1
.while i<=2
mov j,1
.while j<=3
inc j
.endw
inc i
.endw
```

Nested Loops (Cont.)

- The equivalent assembly code segment without using high-level directives is as follow.

```
mov i,1
while01:        cmp i,2
                jg endwhile01
                mov j,1
while02:        cmp j,3
                jg endwhile02
                inc j
                jmp while02
endwhile02:     nop
                inc i
                jmp while01
endwhile01:     nop
```

Complete Program: Implementing Power Function

- Consider the implementation of the power function (x^n), where an iterative definition of the power function is as follows:

x^n = If x < 0 or n < 0, then negative message

Else if x = 0 and n = 0, then undefined message

Else if n = 0, then 1

Otherwise 1 * x * x * … * x (n times)

Complete Program: Implementing Power Function

- The following C program implements the above definition:

```
#include <stdio.h>
int main() {
   int x,n,i,ans;
   printf("%s","Enter x: ");
   scanf("%d",&x);
   printf("%s","Enter n: ");
   scanf("%d",&n);
   if(x<0 || n<0)
       printf("\n%s\n\n","Error: Negative x and/or y");
   else
     if(x==0 && n==0)
       printf("\n%s\n\n","Error: Undefined answer");
     else {
         i=1;
         ans=1;
         while(i<=n) {
            ans=ans*x;
            i++;
         }
         printf("\n%s%d\n\n","The answer is: ",ans);
     }
   return 0;
}
```

Complete Program: Implementing Power Function

- This MASM program also implements the power function.
- Note that high-level directives are used in this program.

```
                .listall
                .386
                .model flat,c
                .stack 100h
  scanf         PROTO arg2:Ptr Byte, inputlist:VARARG
  printf        PROTO arg1:Ptr Byte, printlist:VARARG

                .data
  in1fmt        byte "%d",0
  msg1fmt       byte "%s",0
  msg3fmt       byte "%s%d",0Ah,0Ah,0
  errfmt        byte "%s",0Ah,0Ah,0
  errmsg1       byte 0Ah,"Error: Negative x and/or y",0
  errmsg2       byte 0Ah,"Error: Undefined answer",0
  msg1          byte "Enter x: ",0
  msg2          byte "Enter n: ",0
  msg3          byte 0Ah,"The answer is: ",0
  x             sdword ?
  n             sdword ?
  ans           sdword ?
  i             sdword ?

                .code
  main          proc
                INVOKE printf, ADDR msg1fmt, ADDR msg1
                INVOKE scanf, ADDR in1fmt, ADDR x
                INVOKE printf, ADDR msg1fmt, ADDR msg2
                INVOKE scanf, ADDR in1fmt, ADDR n
                .if x<0 || n<0
                INVOKE printf, ADDR errfmt, ADDR errmsg1
                .else
                .if x==0 && n==0
                INVOKE printf, ADDR errfmt, ADDR errmsg2
                .else
                mov ecx,1
                mov ans,1
                .while ecx <= n
                mov eax,ans
                imul x
                mov ans,eax
                inc ecx
                .endw
                mov i,ecx
                INVOKE printf, ADDR msg3fmt, ADDR msg3, ans
                .endif
                .endif
                ret
main            endp
                end
```

Summary

- For pre-test while loop structure, .while and .endw directives can be used to implement.
- For post-test do-while loop structures, the .repeat and .until directives can be used.
- For fixed iteration for loop structures, the .repeat and .untilcxz directives can be used.
- It is important to include the unconditional jump (jmp) or loop at the bottom of the loop to repeat the corresponding loop.

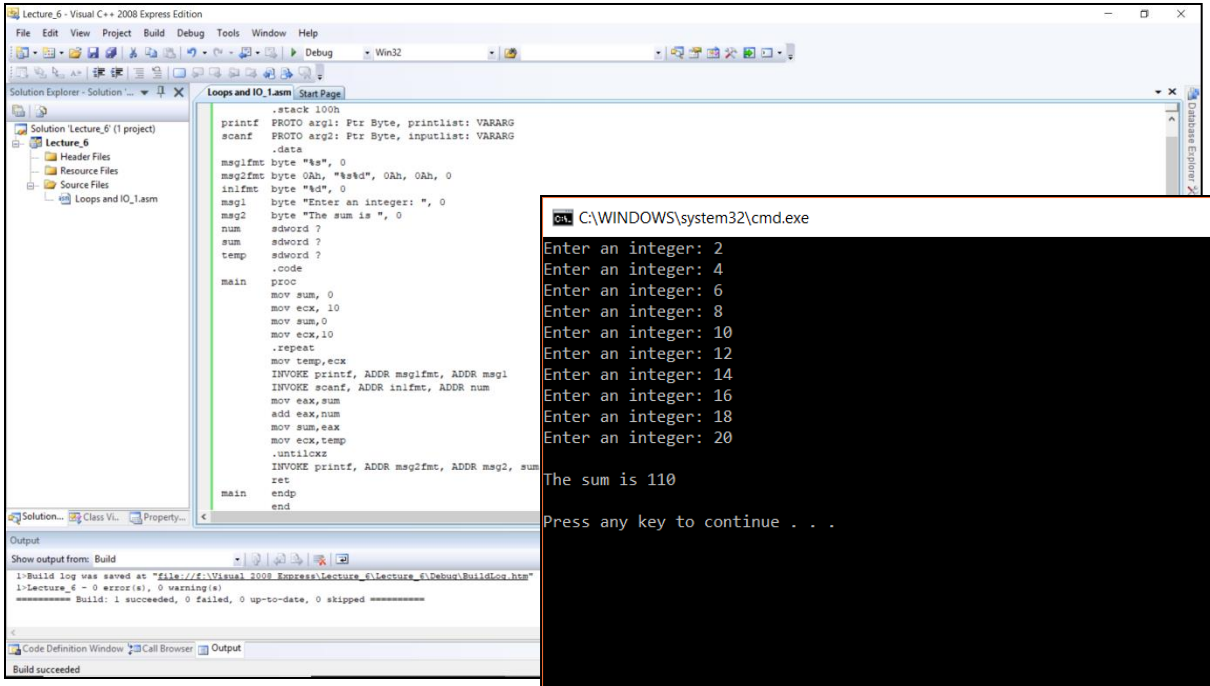
Microprocessor Programming

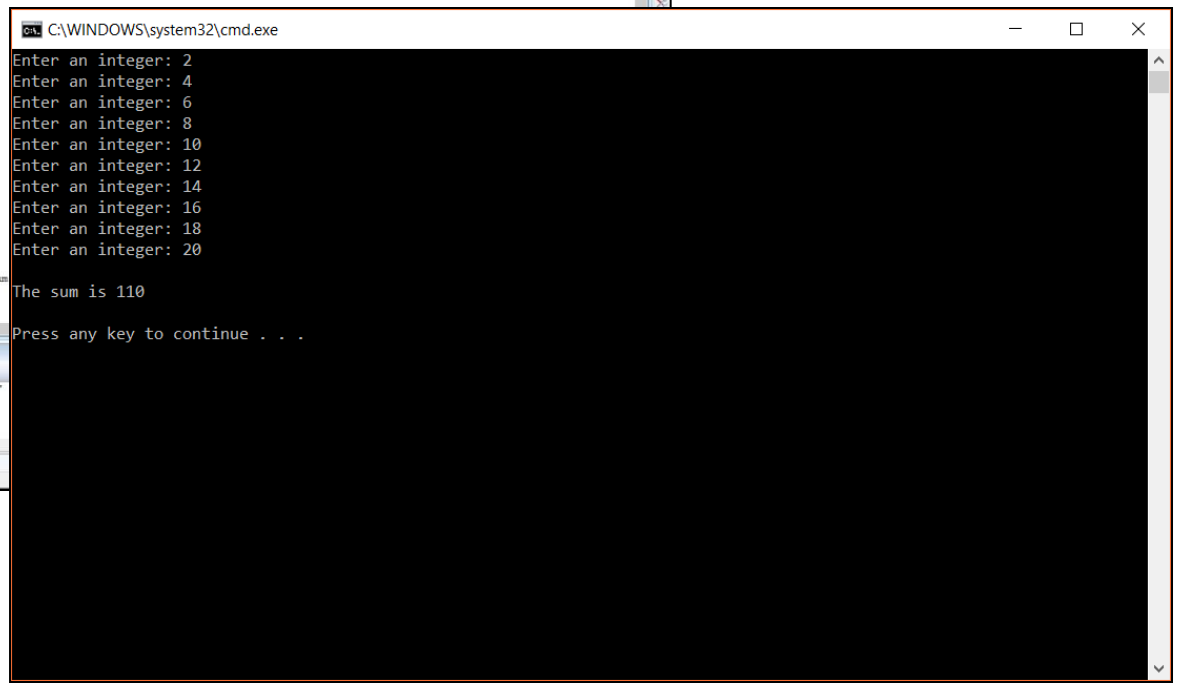
Practical Works

Implementing Loops and I/O and Power Function

Implementing Loops and I/O

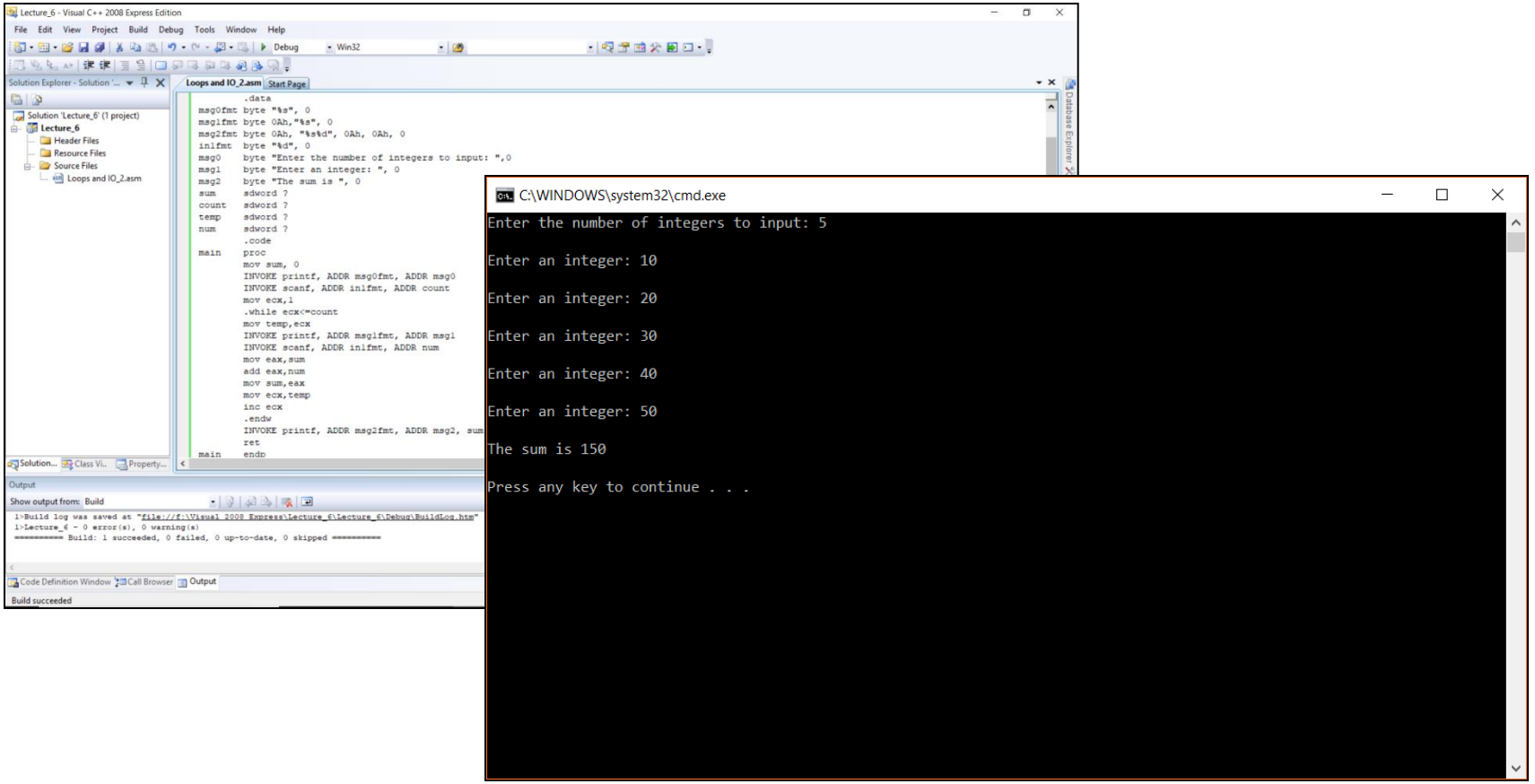
- The given MASM code segment is implemented as a complete program described below:





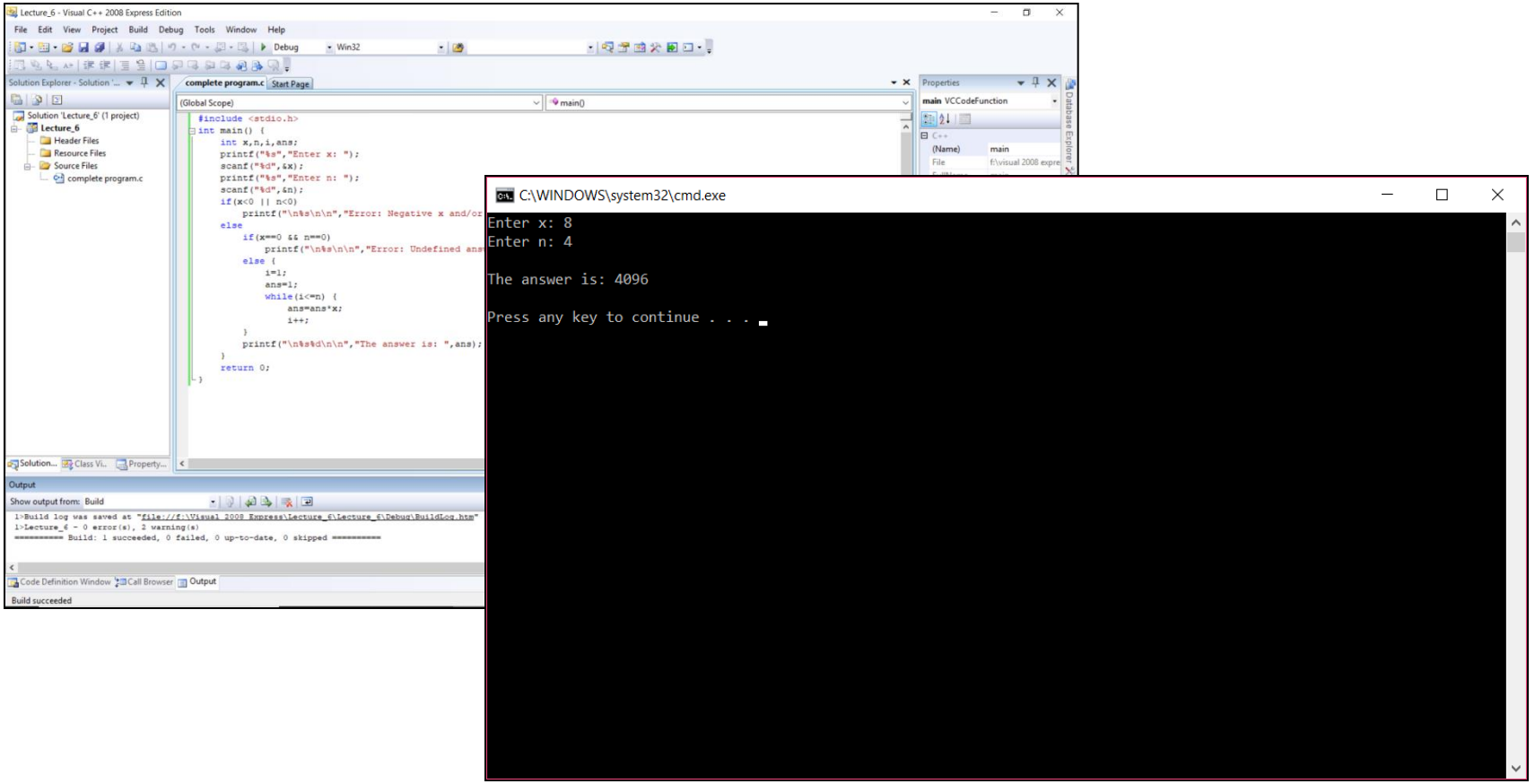
Implementing Loops and I/O (Cont.)

- Given MASM code segment is implemented as a complete program as follow:



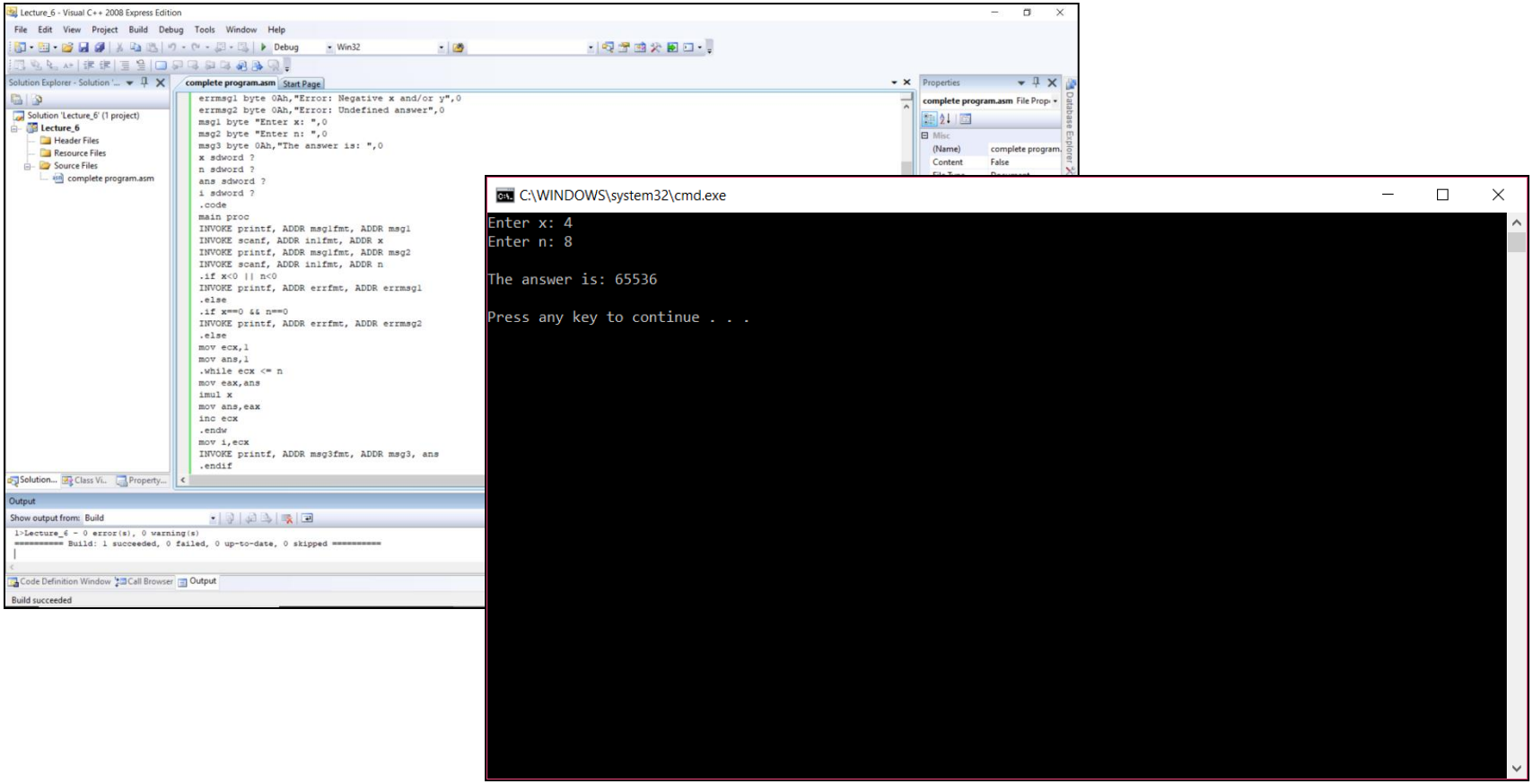
Implementing Power Function

- For a given complete C program, we implement and describe its output below.



Implementing Power Function (Cont.)

- For a given complete MASM program, we implemented as follow:



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

- Implement a complete MASM program for the given segment with compares, jumps, and appropriate labels. Also, display its output.

```
.data
msg1 byte "Enter an integer: ",0
msg2 byte "The sum is ",0
.code
mov sum,0
mov ecx,10
.repeat
mov temp,ecx
INVOKE printf, ADDR msg1fmt, ADDR msg1
INVOKE scanf, ADDR in1fmt, ADDR num
mov eax,sum
add eax,num
mov sum,eax
mov ecx,temp
.untilcxz
INVOKE printf, ADDR msg2fmt, ADDR msg2, sum
```

Assignment 2

- Implement a complete MASM program for the given segment without using high-level directives. Also, display its output.

```
.data
msg0 byte "Enter the number of integers to input: ",0
msg1 byte "Enter an integer: ",0
msg2 byte "The sum is ",0
.code
mov sum,0
INVOKE printf, ADDR msg1fmt, ADDR msg0
INVOKE scanf, ADDR in1fmt, ADDR count
mov ecx,1
.while ecx<=count
mov temp,ecx
INVOKE printf, ADDR msg1fmt, ADDR msg1
INVOKE scanf, ADDR in1fmt, ADDR num
mov eax,sum
add eax,num
mov sum,eax
mov ecx,temp
inc ecx
.endw
INVOKE printf, ADDR msg2fmt, ADDR msg2, sum
```

Assignment 3

- Rewrite a complete MASM code without using high-level directives. Also, display its outputs.

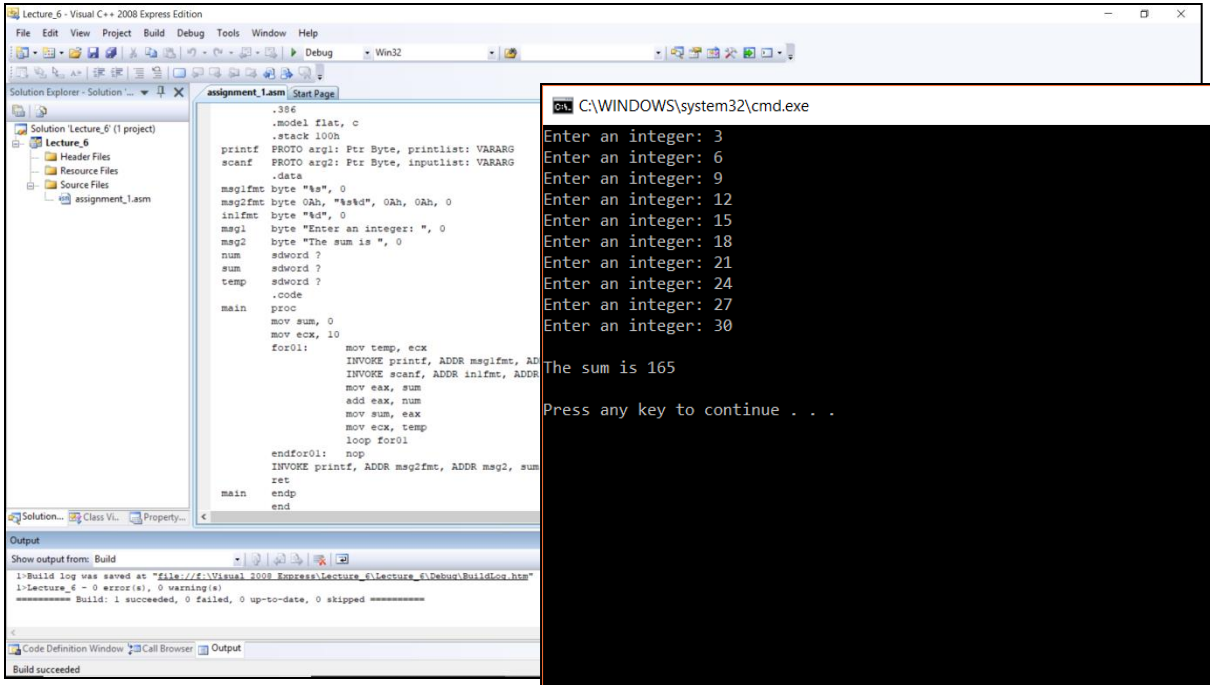
```
                .listall
                .386
                .model flat,c
                .stack 100h
  scanf         PROTO arg2:Ptr Byte, inputlist:VARARG
  printf        PROTO arg1:Ptr Byte, printlist:VARARG

                .data
  in1fmt        byte "%d",0
  msg1fmt       byte "%s",0
  msg3fmt       byte "%s%d",0Ah,0Ah,0
  errfmt        byte "%s",0Ah,0Ah,0
  errmsg1       byte 0Ah,"Error: Negative x and/or y",0
  errmsg2       byte 0Ah,"Error: Undefined answer",0
  msg1          byte "Enter x: ",0
  msg2          byte "Enter n: ",0
  msg3          byte 0Ah,"The answer is: ",0
  x             sdword ?
  n             sdword ?
  ans           sdword ?
  i             sdword ?

                .code
  main          proc
                INVOKE printf, ADDR msg1fmt, ADDR msg1
                INVOKE scanf, ADDR in1fmt, ADDR x
                INVOKE printf, ADDR msg1fmt, ADDR msg2
                INVOKE scanf, ADDR in1fmt, ADDR n
                .if x<0 || n<0
                INVOKE printf, ADDR errfmt, ADDR errmsg1
                .else
                .if x==0 && n==0
                INVOKE printf, ADDR errfmt, ADDR errmsg2
                .else
                mov ecx,1
                mov ans,1
                .while ecx <= n
                mov eax,ans
                imul x
                mov ans,eax
                inc ecx
                .endw
                mov i,ecx
                INVOKE printf, ADDR msg3fmt, ADDR msg3, ans
                .endif
                .endif
                ret
main            endp
                end
```

Microprocessor Programming

Practical Assignments (Report)

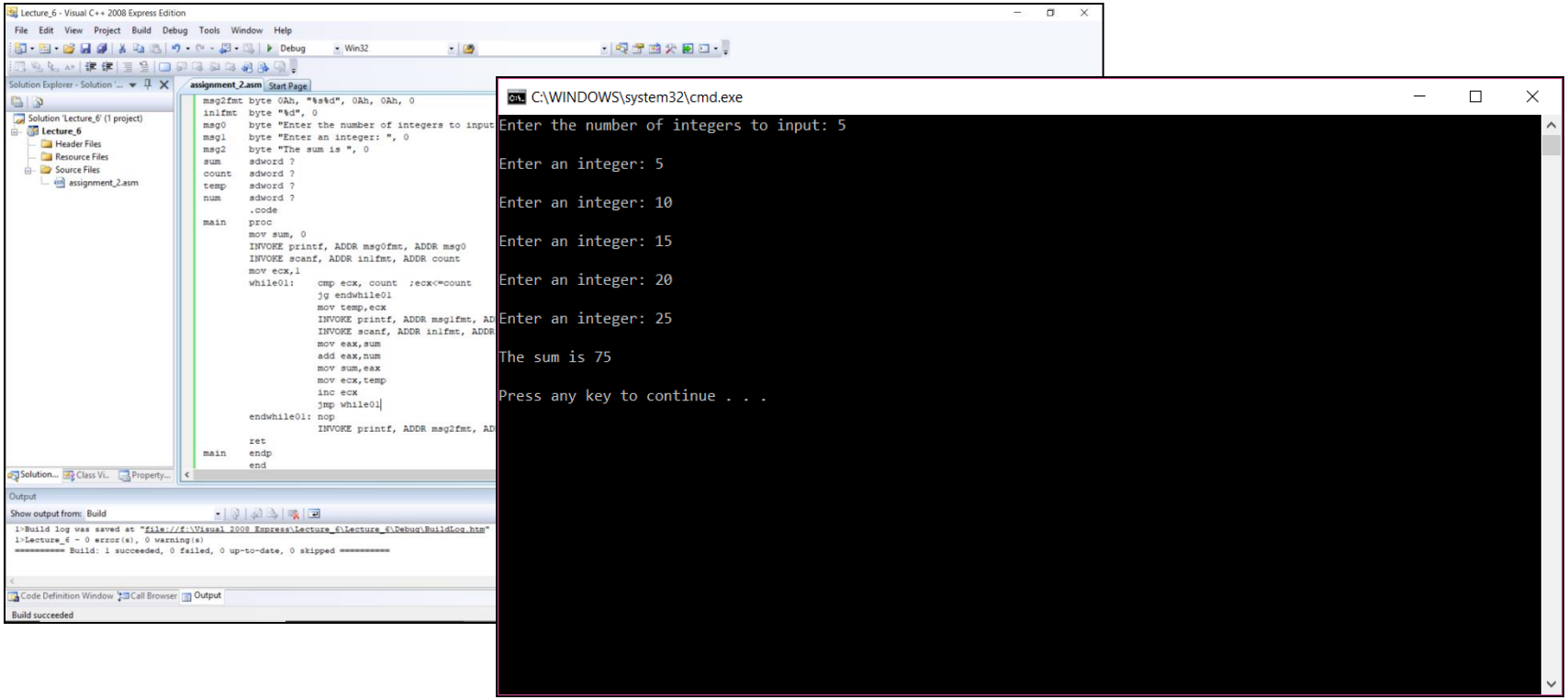
Assignment 1

- A complete assembly program without using high-level directives, with only compares, jumps and labels is implemented and its output is as follow:



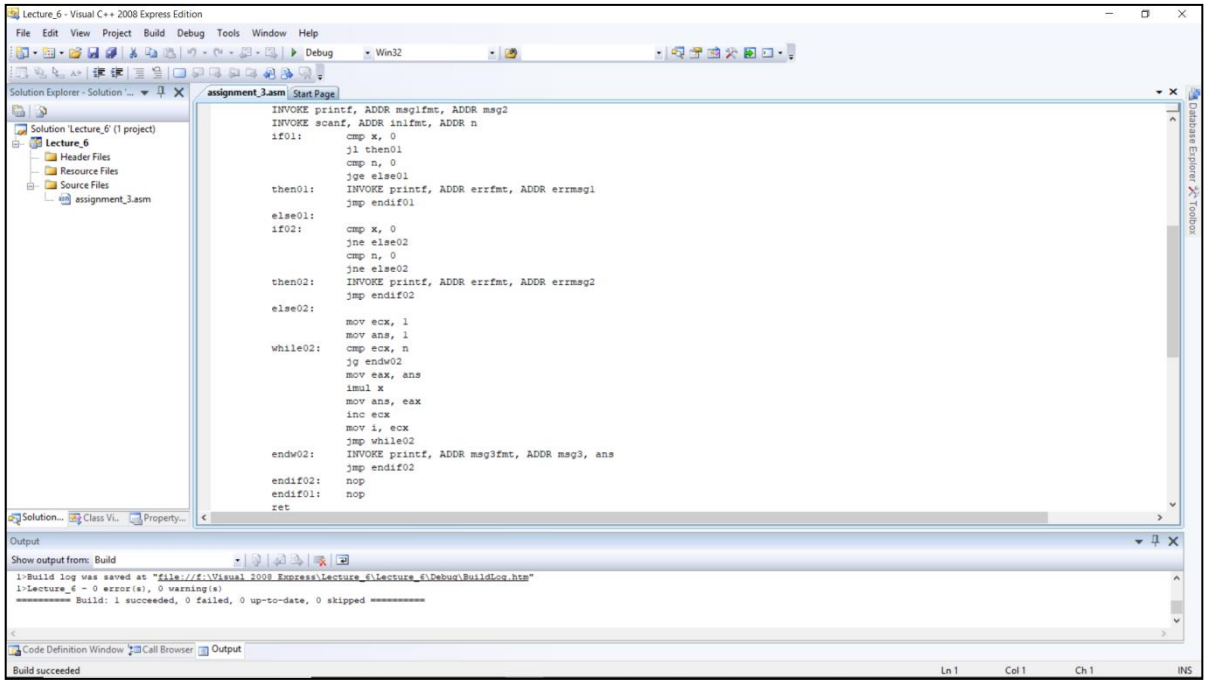
Assignment 2

- A complete assembly program without using high-level directives, with only compares, jumps and labels is implemented as follow:



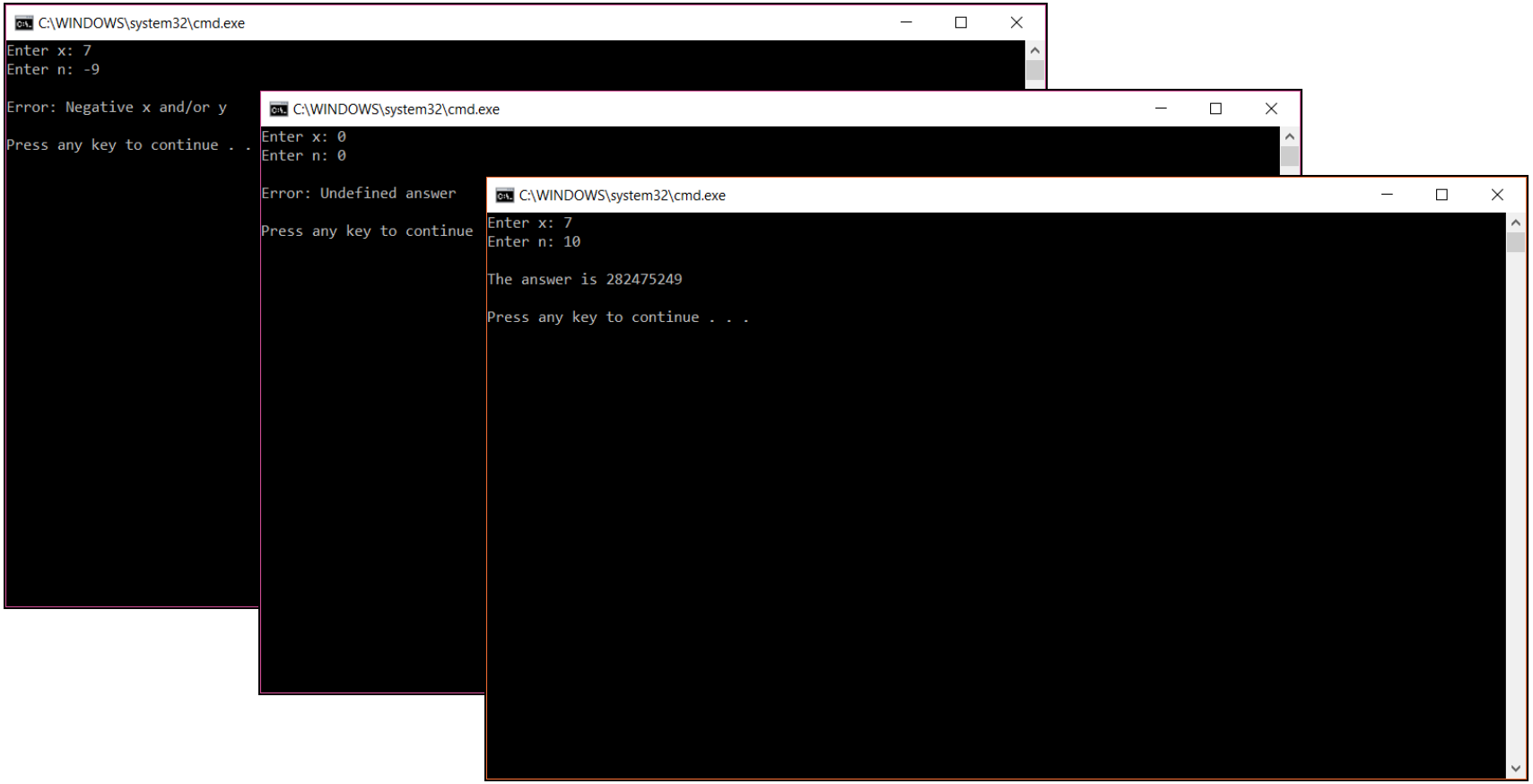
Assignment 3

- A complete assembly program without using high-level directives, with only compares, jumps and labels is implemented as follow:



Assignment 3 (Cont.)

- The followings show some outputs of the implementation of power function:



Next Lecture

- Logic Instructions
- Logical Shift Instructions
- Arithmetic Shift Instructions
- Rotate Instructions

Thank You