Microprocessor Programming

Dr. Tin Ni Ni Kyaw

Ph. D (Kumamoto University, Japan)

Associate Professor

Department of Computer Engineering and Information Technology

Yangon Technological University

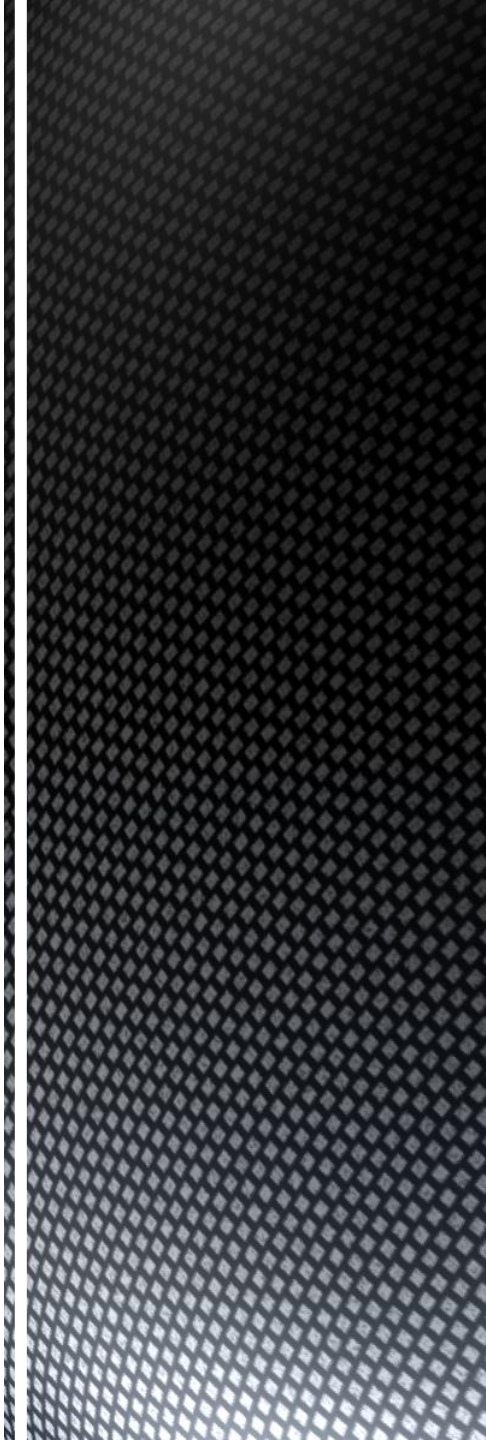
Yangon, Myanmar

Microprocessor Programming

Lecture 10

Macros and Conditional Assembly

Contents



Introduction

- A macro is a named block of assembly language statements.
- Macros are defined directly at the beginning of a source program.
- Once defined, it can be invoked many times in a program whenever it is needed.
- At every point where the macro is called, the assembler inserts a copy of the macro's source code into the program.
- Macros are usually written and located prior to the main program, just after the .code directive.

Structure of Macros

- Macro has a similar structure to a procedure.
- Macro is defined using the MACRO and ENDM directives.
- The first statement of a macro is the MACRO directive together with the name of the macro (e.g., mname) and any parameters associated with it.

```
mname  macro
        .
        .
        .
```

Structure of Macros (Cont.)

- Next, the body of the macro follows.
- Finally, the last statement of a macro is the ENDM directive.
- Unlike an procedure, not only RET instruction but also the name of the macro in front of the ENDM directive do not include in macro as shown below:

```
mname  macro
       .
       ; body of the macro
       .
       endm
```

Structure of Macros (Cont.)

- As an example, a macro for swapping two memory locations num1 and num2 can be written as follows:

```
swap    macro
        mov ebx,num1    ; copy num1 into ebx
        xchg ebx,num2   ; exchange ebx and num2
        mov num1,ebx    ; copy ebx into num1
        endm
```

- Note that the lack of a RET instruction in the macro declaration.

Macros Invocation

- A macro is called (invoked) by inserting its name in the opcode field of the program, possibly followed by macro arguments.
- Call instruction is not necessary to invoke macros.
- Including the CALL instruction when trying to invoke the macro would cause a syntax error.

```
.
swap
.
.
```

Macros Invocation (Cont.)

- The invoking of a macro can be more than one time by just specifying the macro name in the calling program as shown below.

```
.
swap
.
.
swap
.
```

Macros Feature

- A very useful feature when using macros is the ability to use arguments and parameters.
- However, parameters in macros are different from many of the parameters in various high-level languages.
- Macros use the name parameters which are essentially substitution parameters.
- In other words, the names of the arguments are merely substituted in place of the parameter names.

Macros Feature (Cont.)

- For example, in the previous swap macro, we can swap the contents of any two memory locations instead of just num1 and num2.
- The swap macro could be rewritten as follows where p1 and p2 are the two name parameters.

```
swap    macro p1,p2
        mov ebx,p1      ; copy p1 into ebx
        xchg ebx,p2     ; exchange ebx and p2
        mov p1,ebx      ; copy ebx into p1
        endm
```

Macros Feature (Cont.)

- Now, the swap macro with the two name parameters will work with any two memory locations as arguments.
- It can be invoked by following any two arguments as follows.

```
.
.
swap num1,num2
.
.
swap x,y
.
.
```

Macros Feature (Cont.)

- In order to understand how the above macros work, it is best to look at the macro expansions.

```
    .
swap    num1,num2
        mov ebx,num1    ; copy num1 into ebx
        xchg ebx,num2   ; exchange ebx and num2
        mov num1,ebx    ; copy ebx into num1
    .
swap    x,y
        mov ebx,x       ; copy x into ebx
        xchg ebx,y      ; exchange ebx and y
        mov x,ebx       ; copy ebx into x
    .
```

Additional Macro Features

Required Parameters (:REQ)

- If the arguments are required when invoking a macro, :REQ can be used after the parameter name in the macro definition.

```
swap    macro p1:REQ,p2:REQ
        mov ebx,p1          ; copy p1 into ebx
        xchg ebx,p2         ; exchange ebx and p2
        mov p1,ebx          ; copy ebx into p1
        endm
```

Additional Macro Features (Cont.)

Double Semicolons Comment

- Ordinary comment lines appearing in a macro definition appear each time the macro is expanded.
- Not to appear the comment lines in subsequent macro expansions, double semicolons (;;) comments can be used.

```
swap    macro p1:REQ,p2:REQ
        mov ebx,p1          ;; copy p1 into ebx
        xchg ebx,p2         ;; exchange ebx and p2
        mov p1,ebx          ;; copy ebx into p1
        endm
```

Additional Macro Features (Cont.)

Nested Macros

- A macro invoked from another macro is called a nested macro.
- When the assembler's preprocessor encounters a call to a nested macro, it expands the macro in place.
- Parameters passed to an enclosing macro are passed directly to its nested macros.

Conditional Assembly

- Conditional assembly can be a confusing topic to assembly language beginners.
- A number of different conditional-assembly directives can be used in conjunction with macros to make them more flexible.
- The assembler permits the relational operators (LT, GT, EQ, NE, LE, GE) to be used in constant Boolean expressions containing IF and other conditional directives.

Conditional Assembly (Cont.)

- Table 1 lists the conditional assembly directives.
- Probably the best way to illustrate the concept and the directives is through an example.

Table 1. List of Conditional Assembly

Directive	Meaning
`if`	If (can use `EQ, NE, LT, LE, GT, GE, OR, AND`)
`ifb`	If blank
`ifnb`	If not blank
`ifidn`	If identical
`ifidni`	If identical case insensitive
`ifdif`	If different
`ifdifi`	If different case insensitive

Conditional Assembly (Cont.)

Example

- Create a macro called addacc.
- When it is invoked without an argument, the default is to add the number 1 to the eax register.
- When it is invoked with a number or a register or a memory location as an argument, the macro will then add that number or register or memory location to the eax register.

Conditional Assembly (Cont.)

Example

- To implement the micro with the given situation, we need to check the argument first whether it is blank or not.
- If the argument is blank, then the inc instruction would be inserted into the code.
- If the argument is not blank, the add instruction with that argument would be inserted into the code.

Conditional Assembly (Cont.)

Example

- To do this, we can use ifb or ifnb directive.
- The macros using ifb and ifnb directives can be implemented as follows:

```
addacc  macro parm
        ifb <parm>
        inc eax
        else
        add eax,parm
        endif
        endm
```

or

```
addacc  macro parm
        ifnb <parm>
        add eax,parm
        else
        inc eax
        endif
        endm
```

Conditional Assembly (Cont.)

- The following contains four different invocations of the preceding macro definition:

```
.
addacc
.
.
addacc 5
.
.
addacc edx
.
.
addacc num
.
```

Conditional Assembly (Cont.)

- The following shows a copy of each macro invocation followed by only the assembly instruction that would be generated and executed:

```
addacc
    inc eax          ; as the argument is blank
.
addacc 5
    add eax,5        ; as the argument is not blank
.
addacc edx
    add eax,edx      ; as the argument is not blank
.
addacc num
    add eax,num      ; as the argument is not blank
```

Macro Using Conditional Assembly

- Invoking micro with arguments is advantageous.
- But, if the same arguments are used in the swap macro, then redundant code is generated in both cases.
- Here is an example of calling swap micro with the same arguments.

```
swap    macro p1:REQ,p2:REQ
        mov ebx,p1
        xchg ebx,p2
        mov p1,ebx
        endm
```

```
.
swap num1,num1
.
.
swap eax,eax
.
```

Macro Using Conditional Assembly (Cont.)

- As a result, the following redundant codes will be generated.

```
.
swap    num1,num1
mov     ebx,num1
xchg    ebx,num1
mov     num1,ebx
.
swap    eax,eax
mov     ebx,eax
xchg    ebx,eax
mov     eax,ebx
.
```

Macro Using Conditional Assembly (Cont.)

- Generating redundant codes from calling the micro with the same arguments can be solved by using conditional assembly.
- As in Table 1, an ifidn (if identical) checks to see if the two arguments are equal using case sensitivity and an ifidni does the same thing but is case insensitive.
- The directive ifdif (if different) uses case sensitivity to check to see if the arguments are different and ifdifi does the same thing with case insensitivity.

Macro Using Conditional Assembly (Cont.)

Example

- Create a macro called swap with the required two arguments.
- If the arguments are different, then, swap their contents.
- If the arguments are same, then no need to swap the contents and thus no code needs to be generated.

Macro Using Conditional Assembly (Cont.)

Example

- Here, ifidni or ifdifi directives can be used to check the input arguments as shown in the following macro definition:

```
swap    macro p1:REQ, p2:REQ
        ifidni <p1>,<p2>
        else
        mov ebx,p1
        xchg ebx,p2
        mov p1,ebx
        endif
        endm
```

or

```
swap    macro p1:REQ, p2:REQ
        ifdifi <p1>,<p2>
        mov ebx,p1
        xchg ebx,p2
        mov p1,ebx
        endif
        endm
```

Macro Using Conditional Assembly (Cont.)

- The invocation of swap using various different scenarios is shown below:

```
swap    num1,num2
swap    num1,num1
swap    eax,ecx
swap    eax,eax
```

Macro Using Conditional Assembly (Cont.)

- The resulting macro expansions are as follows:
- Note that code is not generated in the second and fourth examples because of the ifidni and ifdifi directives and thus the redundant code need not be generated.

```
swap    num1,num2
        mov ebx,num1
        xchg ebx,num2
        mov num1,ebx
swap    num1,num1
```

```
swap    eax,ecx
        mov ebx,eax
        xchg ebx,ecx
        mov eax,ebx
swap    eax,eax
```

Macro Using Conditional Assembly (Cont.)

Example

- Implement the power function in a macro using the conditional assembly. Use the following modified definition for power function:

x^n = If x < 0 or n < 0, then -1

Else if x = 0 and n = 0, then -1

Else if x = 0 or x = 1, then x

Else if n = 0, then 1

Else if n = 1, then x,

Otherwise 1 * x * x * … * x (n times)

Macro Using Conditional Assembly (Cont.)

Example

- The implementation of the above definition can then be accomplished as below.

```
power   macro x:REQ,n:REQ
        if (x lt 0) or (n lt 0)
        mov eax,-1
        elseif (x eq 0) and (n eq 0)
        mov eax,-1
        elseif (x eq 0) or (x eq 1)
        mov eax,x
        elseif n eq 0
        mov eax,1
        elseif n eq 1
        mov eax,x
        else
        mov eax,x
        mov ebx,eax
        mov ecx,n
        dec ecx
        .repeat
        imul ebx
        .untilcxz
        endif
        endm
```

Macro Using Conditional Assembly (Cont.)

- The following sample invocations test seven different cases:

```
.
power 2,-1
.
power 0,0
.
power 0,2
.
power 1,2
.
power 2,0
.
power 3,1
.
power 2,3
.
```

Macro Using Conditional Assembly (Cont.)

- Given the invocations above, the following macro expansions would be generated in each of the seven cases.

```
.
power   2,-1
        mov eax,-1
.
power   0,0
        mov eax,-1
.
power   0,2
        mov eax,0
.
power   1,2
        mov eax,1
.
```

```
power   2,0
        mov eax,1
.
power   3,1
        mov eax,3
.
power   2,3
        mov eax,2
        mov ebx,eax
        mov ecx,3
        dec ecx
        .repeat
        imul ebx
        .untilcxz
```

Complete Program: Implementing a Macro Calculator

- To implement this macro calculator, assume that there is only one register (eax) in the machine called the accumulator.
- Implement the complete program using the following instructions.

Instruction	Implemented as	Description
INACC	proc	Prompt for and input an integer into the accumulator
OUTACC	proc	Output message and integer in the accumulator
LOADACC	macro	Load the accumulator with the operand
STOREACC	macro	Store accumulator in the operand
ADDACC	macro	Add operand to the accumulator
SUBACC	macro	Subtract operand from the accumulator
MULTACC	macro	Multiply accumulator by the operand (iterative)
DIVACC	macro	Divide accumulator by the operand (iterative)

Complete Program: Implementing a Macro Calculator (Cont.)

- The following is the skeleton of the program:
  1. loads the accumulator with the value 1,
  2. adds the number 2 to the accumulator,
  3. adds the contents of memory location three which contains a 3,
  4. multiplies the accumulator by 4, and then
  5. multiplies the accumulator by a -3.

Complete Program: Implementing a Macro Calculator (Cont.)

- This is the implementation of macro calculator in MASM.

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

                .data
msg1fmt         byte 0Ah,"%s%d",0Ah,0Ah,0
msg1            byte "The contents of the accumulator are: ",0
temp            sdword ?
three           sdword 3
                .code
LOADACC         macro operand
                mov eax,operand     ;; load eax with the operand
                endm
ADDACC          macro operand
                add eax,operand     ;; add to eax the operand
                endm
MULTACC         macro operand
                push ebx            ;; save ebx and ecx
                push ecx
                mov ebx,eax         ;; mov eax to ebx
                mov eax,0           ;; clear accumulator to zero
                mov ecx,operand     ;; load ecx with operand
                if operand LT 0     ;; if operand is negative
                neg ecx             ; make ecx positive for loop
                endif
               .while ecx >0
               add eax,ebx          ;; repetitively add
               dec ecx              ;; decrement ecx
               .endw
               if operand LT 0      ;; if operand is negative
               neg eax              ; negate accumulator, eax
               endif
               pop ecx              ;; restore ecx and ebx
               pop ebx
               endm
```

```
main            proc
                LOADACC 1
                ADDACC 2
                ADDACC three
                MULTACC 4
                MULTACC -3
                CALL OUTACC
                ret
main            endp
OUTACC          proc
                push eax     ; save eax, ecx, and edx
                push ecx
                push edx
                mov temp,eax
                INVOKE printf, ADDR msg1fmt, ADDR msg1, temp
                pop edx    ; restore eax, ecx, and edx
                pop ecx
                pop eax
                ret
OUTACC          endp
                end
```

Complete Program: Implementing a Macro Calculator (Cont.)

- The main program above only contains the following macro invocations and procedure call.

```
LOADACC 1
ADDACC 2
ADDACC three
MULTACC 4
MULTACC -3
CALL OUTACC
```

Summary

- Macro sequences must always be defined before they are used in a program, so they generally appear at the top of the code segment.
- To invoke a macro, put the name of the macro and no need to use CALL instruction.
- RET instruction does not include in a macro.
- The name of the macro does not include in front of the ENDM statement.

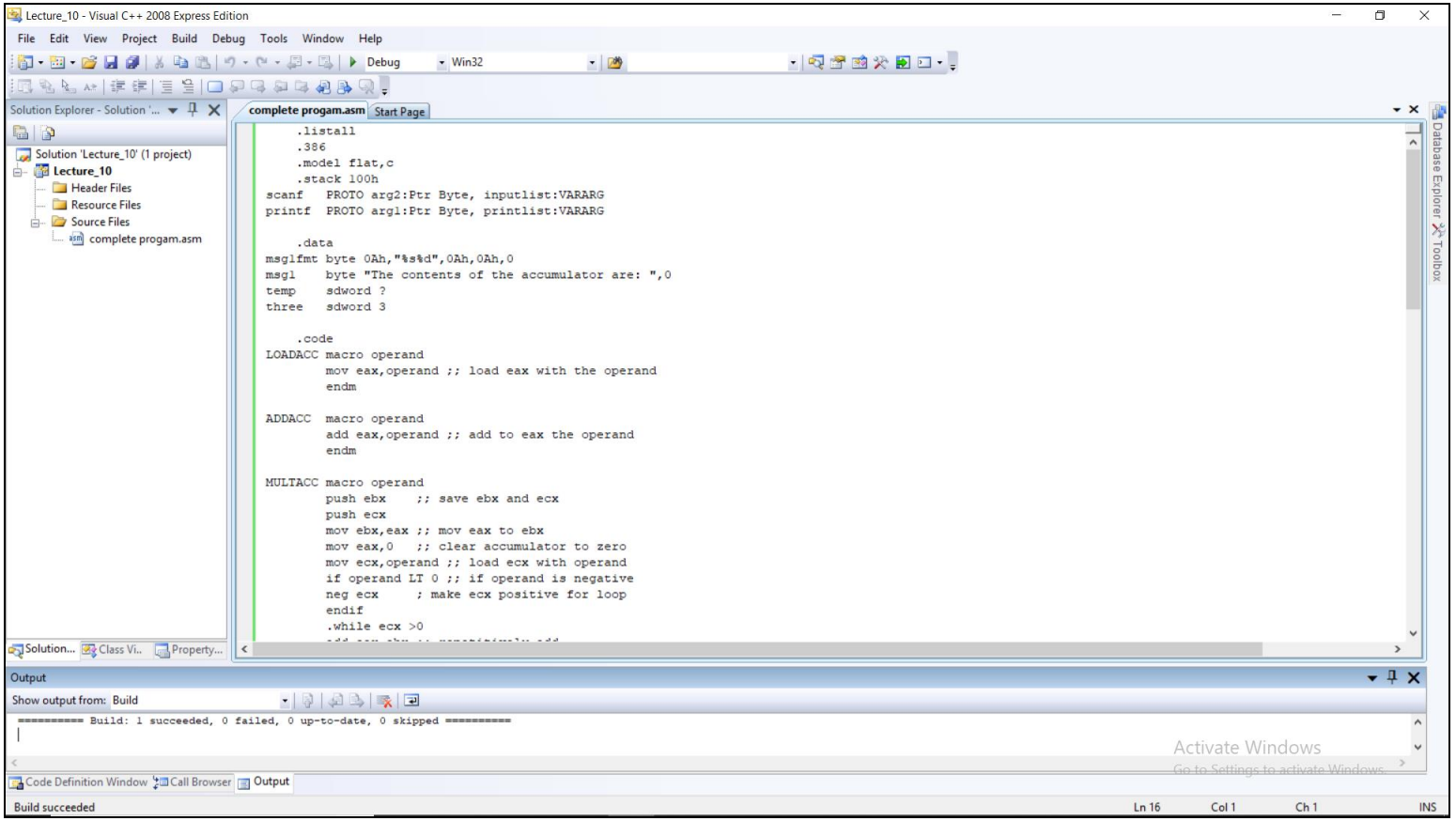
Microprocessor Programming

Practical Works

Implementing a Macro Calculator

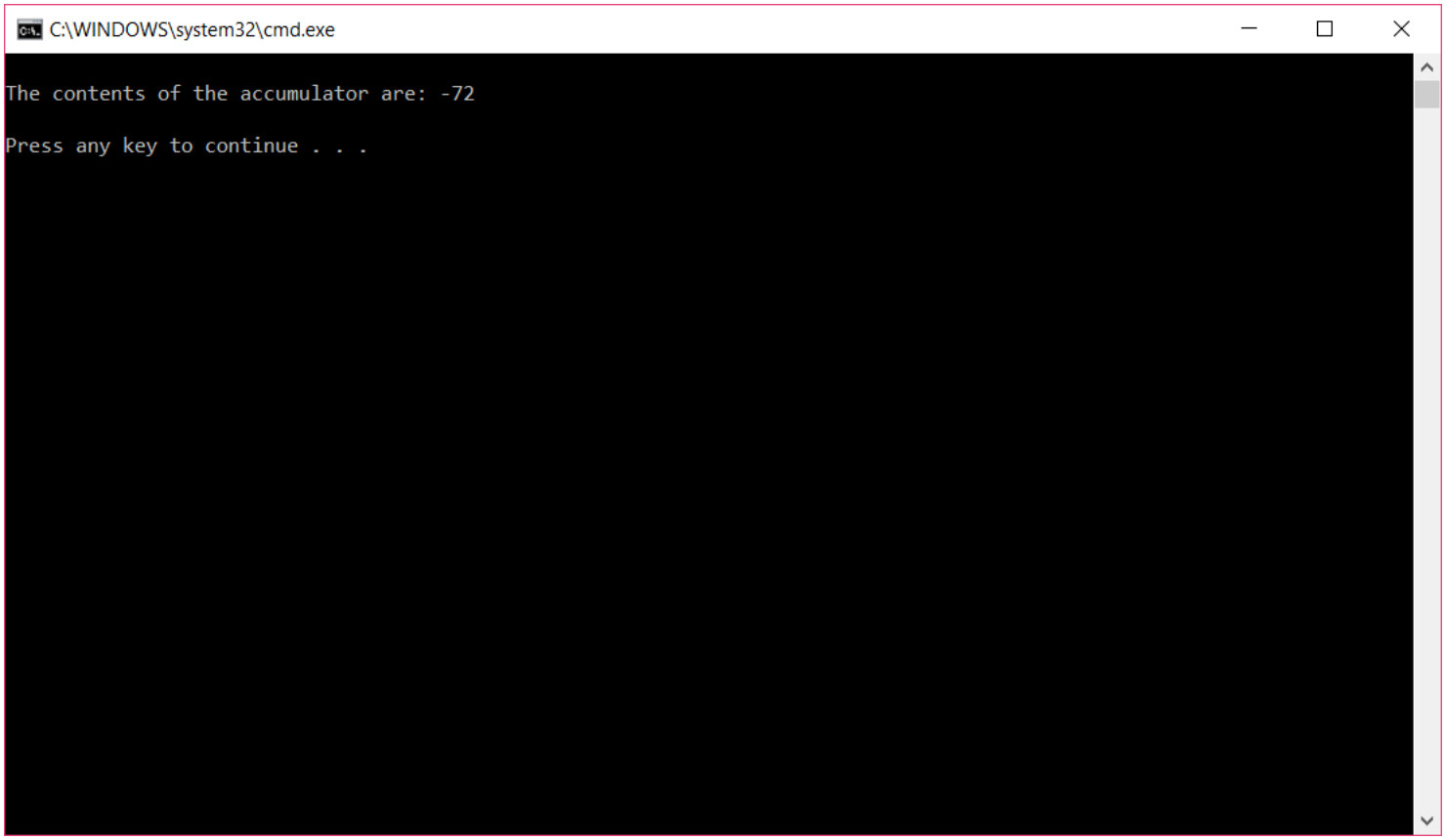
Implementing a Macro Calculator

- The implementation of the given MASM program is described below:



Implementing a Macro Calculator

- The output of the given program is described below:



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

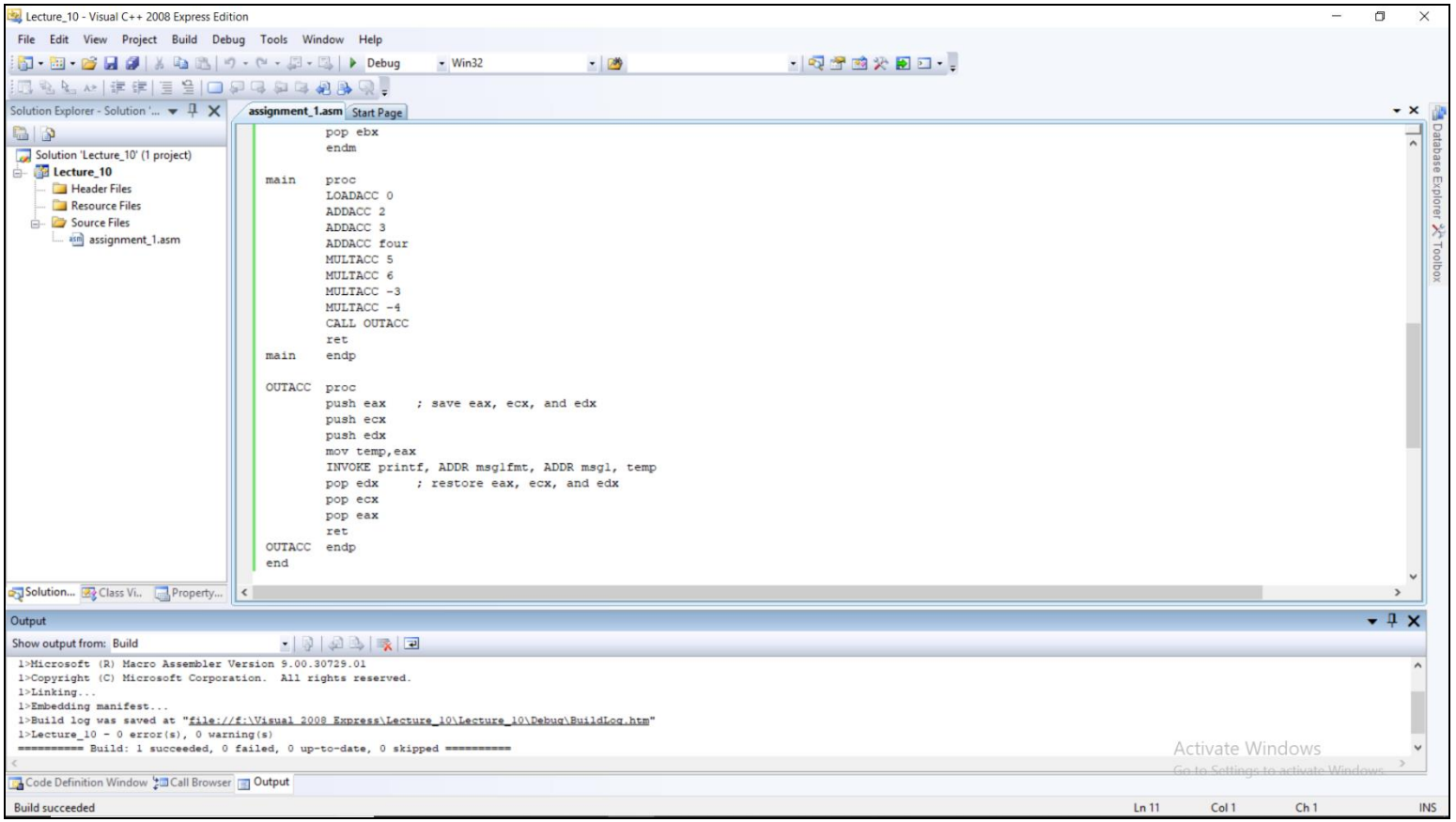
- Rewrite a complete assembly program using macros with conditional assembly for the following skeleton of the program:
  1. loads the accumulator with the value 0,
  2. adds the number 2 and 3 to the accumulator,
  3. adds the contents of memory location four which contains a 4,
  4. multiplies the accumulator by 5 and 6, and then
  5. multiplies the accumulator by a -3 and -4.

Microprocessor Programming

Practical Assignments (Report)

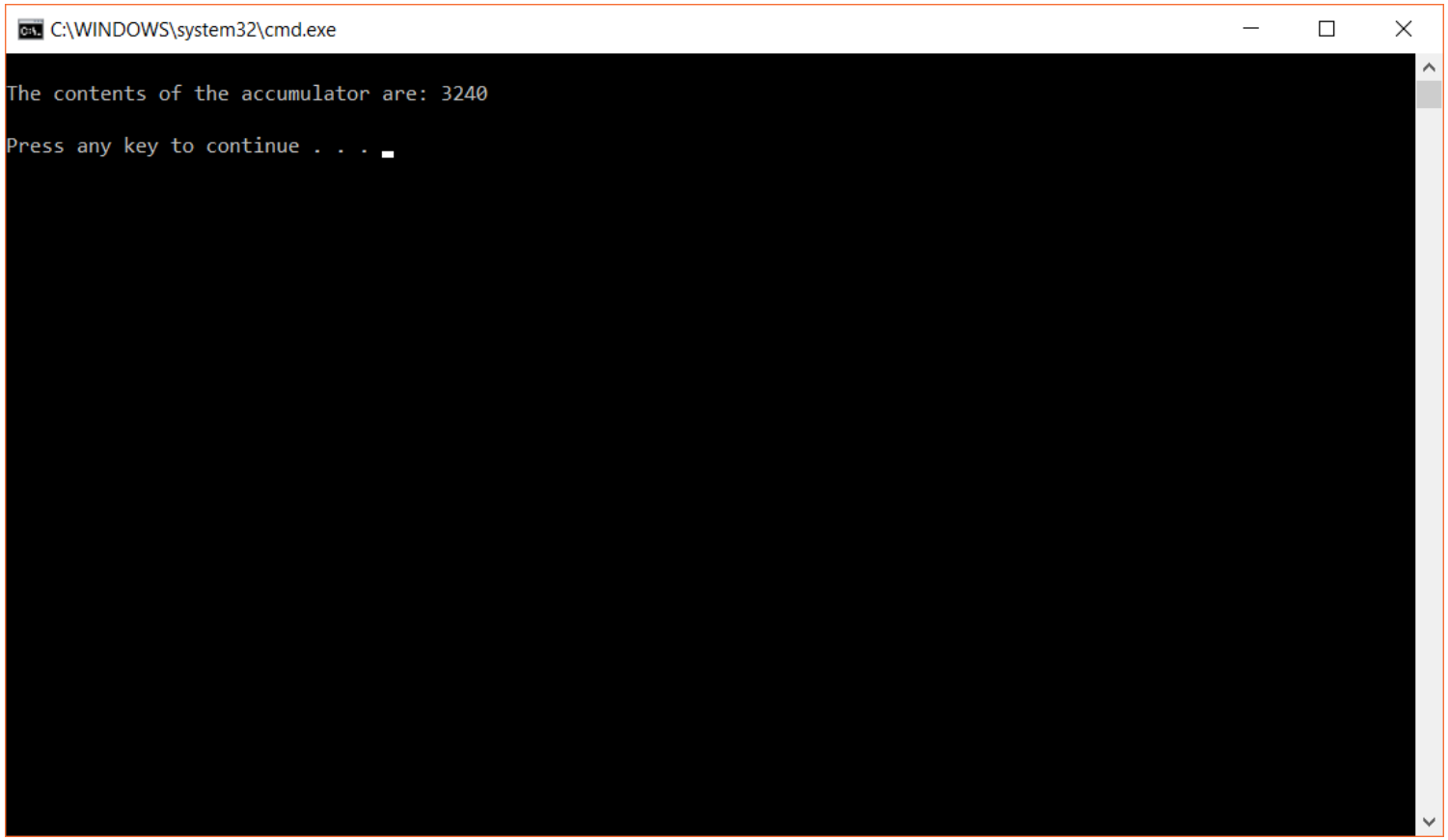
Assignment 1

- A complete assembly program is described below:



Assignment 1

- The output for the complete assembly program is described below:



Next Lecture

- Arrays
- Searching
- Queue
- Selection Sort

Thank You