Microprocessor Programming

Dr. Tin Ni Ni Kyaw

Ph. D (Kumamoto University, Japan)

Associate Professor

Department of Computer Engineering and Information Technology

Yangon Technological University

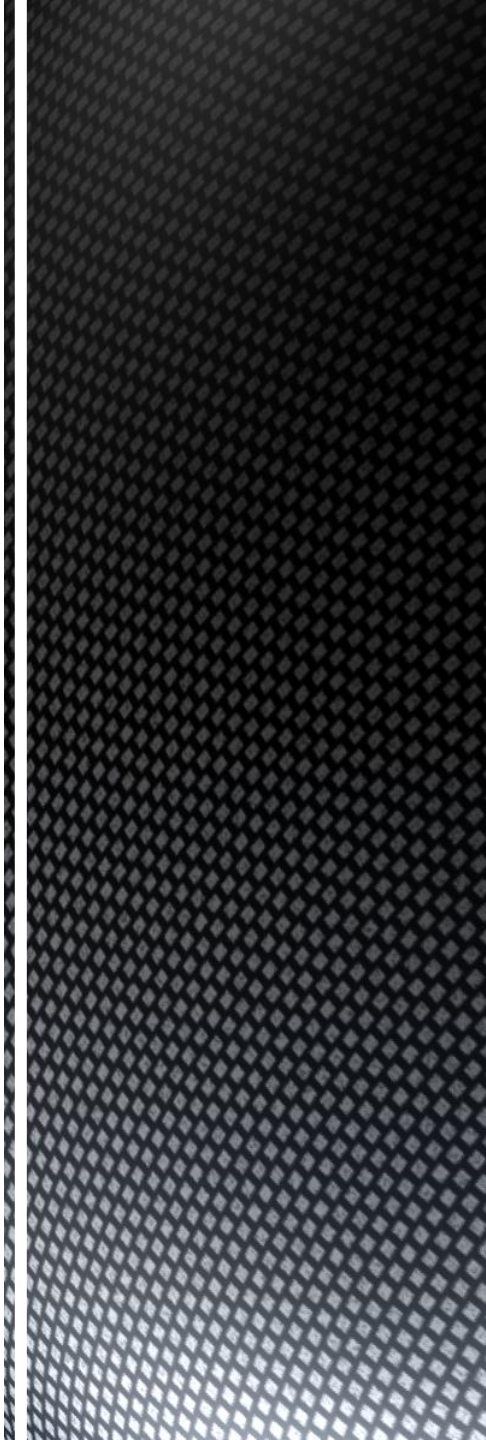
Yangon, Myanmar

Microprocessor Programming

Lecture 5

Selection Structures

Contents



Introduction

- Selection structure is one of the basic control structures in any programming language.
- The most common selection structures are:
  1. if-then structures and
  2. if-then-else structures.
- In addition, there exists the case structure, known as the switch statement in C, C++, and Java.

Introduction

- At a lower level, all control structures can be created using if and conditional and unconditional branch statements.
- The conditional branches work only under certain conditions (such as if equal to 0).
- The unconditional one branches unconditionally whatever the condition is.
- Most assembly languages often use the equivalent of a goto statement in their programs.

If-Then Structure

- In C, the common if-then structure has the following form.

```
if (number == 0)
        number--;
```

- If there is only one statement in the then section, the use of the opening and closing braces { } is optional.

If-Then Structure (Cont.)

- The corresponding MASM code using the high-level directives for the above is shown below:

```
.if number == 0
        dec number
.endif
```

- .if and .endif are high-level directives that tell the assembler to insert the necessary code to implement the directives.
- Also, the directive .endif is required even there exists only one statement in then section.

If-Then Structure (Cont.)

- If there are more than one statement in the then section of the if structure, the use of the braces is required in C as shown below.

```
if (amount != 1)
{
    count++;
    amount = amount + 2;
}
```

If-Then Structure (Cont.)

- The corresponding MASM code using high-level directives for the above C statements is shown below.

```
.if amount != 1
        inc count
        add amount,2
.endif
```

- Note that there is no parenthesis around the relational as in C.

If-Then Structure (Cont.)

- If we need to compare the two memory locations, we can write the C statements as follows.

```
if (count > number)
        flag = -1;
```

If-Then Structure (Cont.)

- In MASM, the memory to memory comparison cannot be performed and so, the contents of one of the two variables need to be copied into a register.
- Then, a comparison between the register and the other variable can be performed as illustrated below.

```
mov eax,count
.if eax > number
        mov flag,-1
.endif
```

If-Then Structure (Cont.)

- In order to implement using only if-then-structure instead of the .if directives, we can use the compare instructions along with one of the conditional jump instructions.
- The compare instruction performs the comparison between the two operands and sets the flags according to the comparison result.
- To do this, first, we need to know compare instructions, flags and also jump instructions.

If-Then Structure (Cont.)

Compare Instructions

- The mnemonics used for comparison is cmp.
- The formats of the cmp instructions are given in Table 1.

Table 1. cmp Instructions

Instruction	Meaning
cmp reg,imm	compare a register to an immediate value
cmp imm,reg	compare an immediate value to a register
cmp reg,mem	compare a register to memory
cmp mem,reg	compare memory to a register
cmp mem,imm	compare memory to an immediate value
cmp imm,mem	compare an immediate value to memory
cmp reg,reg	compare a register to a register

If-Then Structure (Cont.)

Flags

- After the two operands have been compared, the corresponding flags will be set.
- Two of the most important flags are zero flag and sign flag.
- Depending on the result of the last instruction executed, they may be set as zero or one as described in Table 2.

Table 2. Zero Flag and Sign Flag

Result	Zero Flag	Sign Flag
Zero	1	0
Negative	0	1
Positive	0	0

If-Then Structure (Cont.)

Flags

- Zero flag and sign flag indicate that the last instruction's execution result was zero or negative or positive.
- If the result was zero, the ZF would be set to 1 and the SF would be set to 0.
- If the result was negative, then SF would be set to 1 and ZF would be set to 0.
- Lastly, if the result was positive, then both ZF and SF would be set to 0.

If-Then Structure (Cont.)

Jump Instructions

- Once the corresponding flags were set, one can branch or jump based on the flags.
- Two of the conditional jump instructions are shown in Table 3.
- The je and jne instructions can be used with either signed or unsigned data.

Table 3. je and jne instructions

Instruction	Meaning
je	Jump equal
jne	Jump not equal

If-Then Structure (Cont.)

Jump Instructions

- For signed numeric data, the conditional jump instructions are listed in Table 4.

Table 4. Signed Conditional Jump Instructions

Instruction	Meaning
jg	Jump greater than
jnle	Jump not less than or equal to
jge	Jump greater than or equal to
jnl	Jump not less than
jl	Jump less than
jnge	Jump not greater than or equal to
jle	Jump less than or equal to
jng	Jump not greater than

If-Then Structure (Cont.)

- To illustrate how to implement if-then-structures by using compare and jump instructions, consider a previous example:

```
.if number == 0
        dec number
.endif
```

```
if01:       cmp number,0        ; compare number and zero
            jne endif01         ; jump to endif01 if not equal
then01:     dec number          ; decrement number by 1
endif01:    nop                 ; end if, no operation
```

If-Then Structure (Cont.)

- According to the flow of the program, the decrementing of number occurs when it is equal to 0; otherwise number remains unchanged.
- Firstly, the comparison is done between number and 0.
- If the two are not equal, a jump not equal (jne) occurs to the endif01 label.
- If they are equal, the flow of control falls through to the decrement statement immediately below.

If-Then Structure (Cont.)

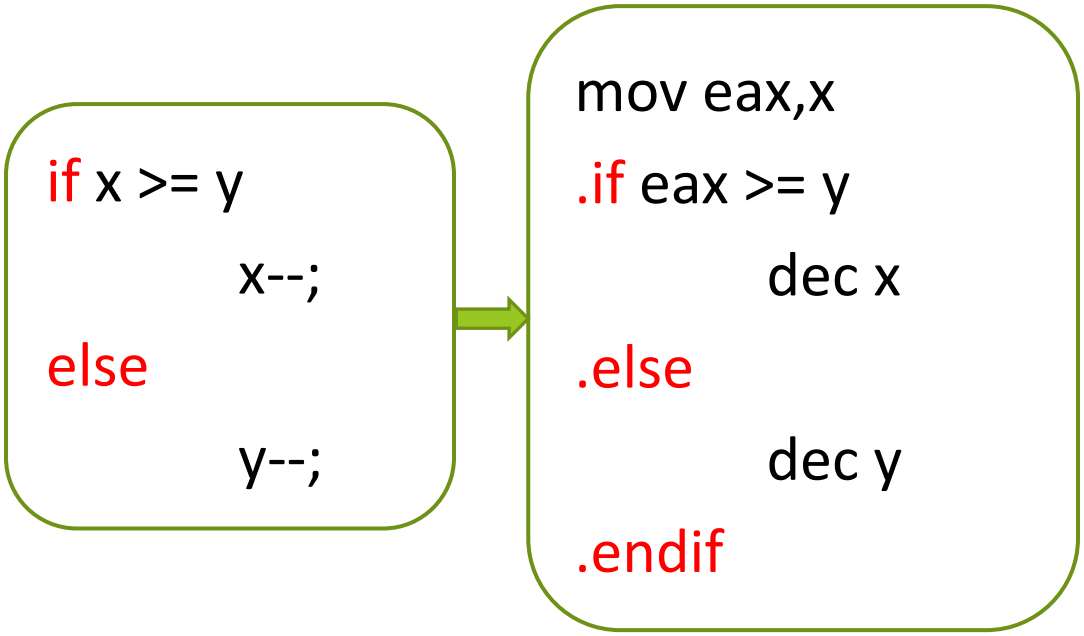
- As another example, consider the following segment to be implemented without using the .if directive.

```
mov eax,count
.if eax > number
        mov flag,-1
.endif
```

```
if02:       mov eax,count
            cmp eax,number
            jle endif02
then02:     mov flag,-1
endif02:    nop
```

If-Then-Else Structure

- It is relatively easy to extend the implementation of if-then structure to the if-then-else structure.
- For example, the following C code on the left would be implemented using the .else directive as shown on the right:



If-Then-Else Structure (Cont.)

- Further, it can be implemented without the use of directives by using compares, jumps, and labels.

```
mov eax,x
.if eax >= y
        dec x
.else
        dec y
.endif
```

```
if03:       mov eax,x
            cmp eax,y
            jl else03
then03:     dec x
            jmp endif03
else03:     dec y
endif03:    nop
```

If-Then-Else Structure (Cont.)

- In the previous implementation, it is important to include the unconditional jump (jmp) at the end of the then section.
- Otherwise, the flow of control will fall through into the else section which would not correctly implement the if-then-else structure.

Nested If Structures

- Just as one can nest if structures in a high-level language, the same can be done in a low-level language.
- This is especially easy with the use of high-level directives in MASM.

Nested If Structures (Cont.)

- For example, the following C code segment on the left can be implemented in MASM as shown on the right:

```
if (x < 50)
        y++;
else
        if (x <= 100)
                y=0;
        else
                y--;
```

```
.if x < 50
        inc y
.else
        .if x <=100
                mov y,0
        .else
                dec y
        .endif
.endif
```

Nested If Structures (Cont.)

- The code can be implemented without using the directives as shown below on the right.

```
.if x < 50
        inc y
.else
        .if x <=100
                mov y,0
        .else
                dec y
        .endif
.endif
```

```
if01:       cmp x,50
            jge else01
then01:     inc y
            jmp endif01
else01:     nop
if02:       cmp x,100
            jg else02
then02:     mov y,0
            jmp endif02
else02:     nop
            dec y
endif02:    nop
endif01:    nop
```

Nested If Structures (Cont.)

- As another example, the C code can be rewritten in MASM by using the directives as shown below.

```
if (x <= 100)
        if (x < 50)
                y++;
        else
                y = 0;
else
        y--;
```

```
.if x <= 100
        .if x < 50
                inc y
        .else
                mov y,0
        .endif
.else
        dec y
.endif
```

Nested If Structures (Cont.)

- Writing the MASM code without using the directives is as shown below.

```
.if x <= 100
        .if x < 50
                inc y
        .else
                mov y,0
        .endif
.else
        dec y
.endif
```

```
if03:       cmp x,100
            jg else03
then03:     nop
if04:       cmp x,50
            jge else04
then04:     inc y
            jmp endif04
else04:     nop
            mov y,0
endif04:    nop
            jmp endif03
else03:     dec y
endif03:    nop
```

Case Structures

- MASM does not have a case structure directive.
- It can be created using a combination of conditional and unconditional jumps.
- Consider the following C switch statement:

```
switch (w)
{
        case 1:         x++;
                        break;
        case 2:
        case 3:         y++;
                        break;
        default:        z++;
}
```

Case Structures (Cont.)

- The above switch structure can be implemented as a series of cmp and je instructions.

```
switch01:       cmp w,1
                je case11;
                cmp w,2
                je case12
                cmp w,3
                je case12
                jmp default01
case11:         inc x
                jmp endswitch01
case12:         inc y
                jmp endswitch01
default01:      inc z
endswitch01:    nop
```

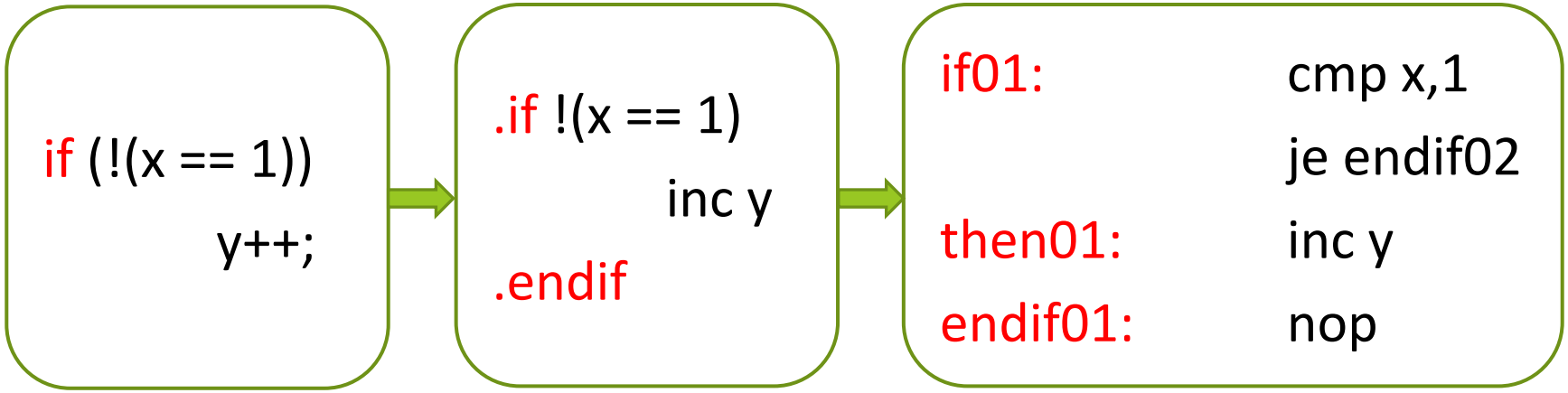
Logical Operations

- Now, how to implement the logical operations using .if directives and using only if-then structures will be examined in this section.
- Consider the following C code segment that contains the logical "not" operator to rewrite in assembly code.

```
if (!(x == 1))
        y++;
```

Logical Operations (Cont.)

- The C code segment on the left can be easily implemented in assembly language using high-level directives and without using .if directives as shown on the right:



Logical Operations (Cont.)

- Further, the C code that contains logical "or" operator is given on the left below.
- It can be implemented by using high-level directives as shown on the right.

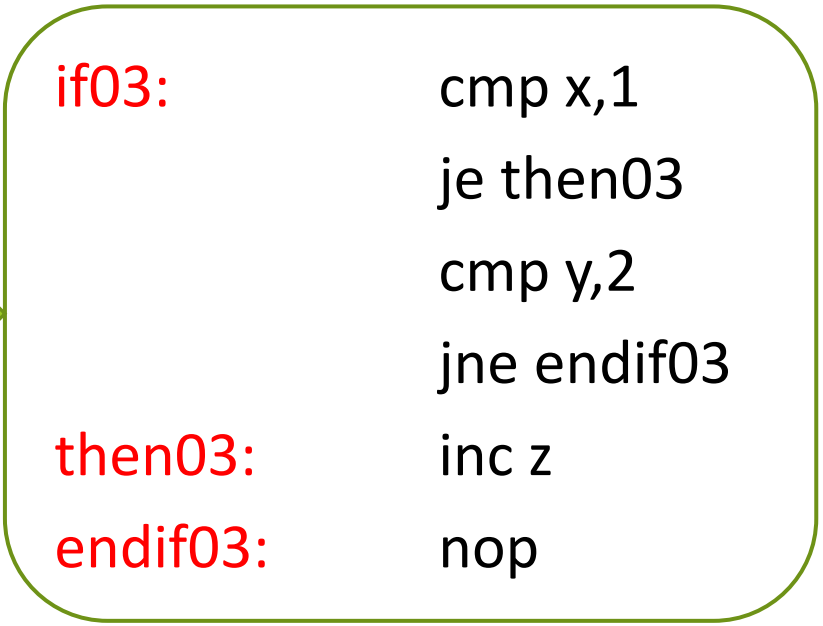
```
if(x==1 || y==2)
        z++;
```

```
.if x==1 || y==2
        inc z
.endif
```

Logical Operations (Cont.)

- The MASM code using .if directives is given on the left below.
- It can be rewritten by using if-then structure as shown on the right.

```
.if x==1 || y==2
        inc z
.endif
```



```
if03:       cmp x,1
            je then03
            cmp y,2
            jne endif03
then03:     inc z
endif03:    nop
```

Logical Operations (Cont.)

- Again, the C code that contains logical "and" operator is given on the left below.
- It can be implemented by using high-level directives as shown on the right.

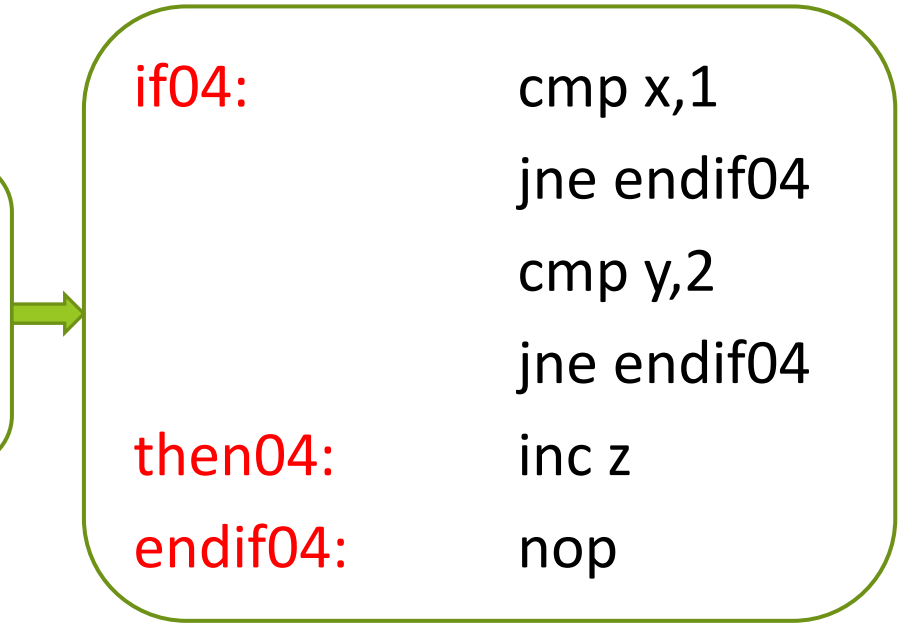
```
if (x==1 && y==2)
    z++;
```

```
.if x==1 && y==2
    inc z
.endif
```

Logical Operations (Cont.)

- Again, the MASM code using .if directives given on the left below can be implemented by using if-then structure as shown on the right.

```
.if x==1 && y==2
        inc z
.endif
```



```
if04:       cmp x,1
            jne endif04
            cmp y,2
            jne endif04
then04:     inc z
endif04:    nop
```

Logical Operations (Cont.)

- In more complicated examples, the rules of precedence for logical operators should be noted.
- First, the logical unary “not” operation (!) has the highest precedence.
- Next, the “and” operator (&&) has higher precedence over the “or” operator (||).
- As with arithmetic, the most nested parentheses are evaluated first.
- Lastly, in the case of a tie between operations, the order is from left to right.

Logical Operations (Cont.)

- To realize the combination of logical operators and their orders, the logical expressions in C and its equivalent assembly language code segments are shown below.

```
if w==1 || (x==2 && y==3)
        z++;
```

```
.if w==1 || x==2 && y == 3
        inc z
.endif
```

Logical Operations (Cont.)

- The assembly language code segments without using high-level directives using only if-then structures are shown below.

```
.if w==1 || x==2 && y == 3
        inc z
.endif
```

```
if05:       cmp x,2
            jne or05
            cmp y,3
            je then05
or05:       cmp w,1
            jne endif05
then05:     inc z
endif05:    nop
```

Complete Program: Using Selection Structures and I/O

- Suppose that we want to input a value representing an alternating current (AC) voltage, indicate whether the voltage was either too high, too low, or at an acceptable level, and then output an appropriate message according to Table 5.

Table 5. Voltages and Messages

Voltage	Message
109 and below	Warning! Voltage too low
110–120, inclusive	Voltage is acceptable
121 and above	Warning! Voltage too high

Complete Program: Using Selection Structures and I/O

- Table 6 contains three samples of the prompt and messages needed in order from left to right.

Table 6. Sample Input/Output

Sample I/O	Sample I/O	Sample I/O
Enter an AC voltage: 109	Enter an AC voltage: 110	Enter an AC voltage: 121
Warning! Voltage too low	Voltage is acceptable	Warning! Voltage too high

Complete Program: Using Selection Structures and I/O

- To understand the logic and I/O better, it is probably best to show the solution in a C program first.

```
#include <stdio.h>
int main () {
   int voltage;
   printf("%s", "Enter an AC Voltage: ");
   scanf("%d", &voltage);
   if (voltage >= 110 && voltage <= 120)
       printf("\n%s\n", "Voltage is Acceptable");
   else {
       printf("\n%s\n","Warning!");
       if voltage < 110)
           printf("%s\n","Voltage too Low");
       else
           printf("%s\n","Voltage too High");
       }
   printf("\n");
   return 0;
}
```

Complete Program: Using Selection Structures and I/O

- The above C program is implemented in MASM using high-level directives as shown below.

```
            .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
msg2fmt     byte 0Ah,"%s",0Ah,0
msg4fmt     byte "%s",0Ah,0
msg6fmt     byte 0Ah,0
msg1        byte "Enter an AC voltage: ",0
msg2        byte "Voltage is Acceptable",0
msg3        byte "Warning!",0
msg4        byte "Voltage too Low",0
msg5        byte "Voltage too High",0
voltage     sdword ?

            .code
main        proc
            INVOKE printf, ADDR msg1fmt, ADDR msg1
            INVOKE scanf, ADDR in1fmt, ADDR voltage
            .if voltage >=110 && voltage <= 120
            INVOKE printf, ADDR msg2fmt, ADDR msg2
            .else
            INVOKE printf, ADDR msg2fmt, ADDR msg3
            .if voltage < 110
            INVOKE printf, ADDR msg4fmt, ADDR msg4
            .else
            INVOKE printf, ADDR msg4fmt, ADDR msg5
            .endif
            .endif
            INVOKE printf, ADDR msg6fmt
            ret
main        endp
            end
```

Summary

- When implementing if statements without high-level directives, the conditional jump often needs to be reversed to implement the if statement correctly.
- MASM does not have a high-level case structure, but it can be constructed by using compare and jump statements.
- When not using high-level directives, use good label names to help readability.
- Note that a logical “and” operation (&&) has precedence over a logical “or” operation (||).

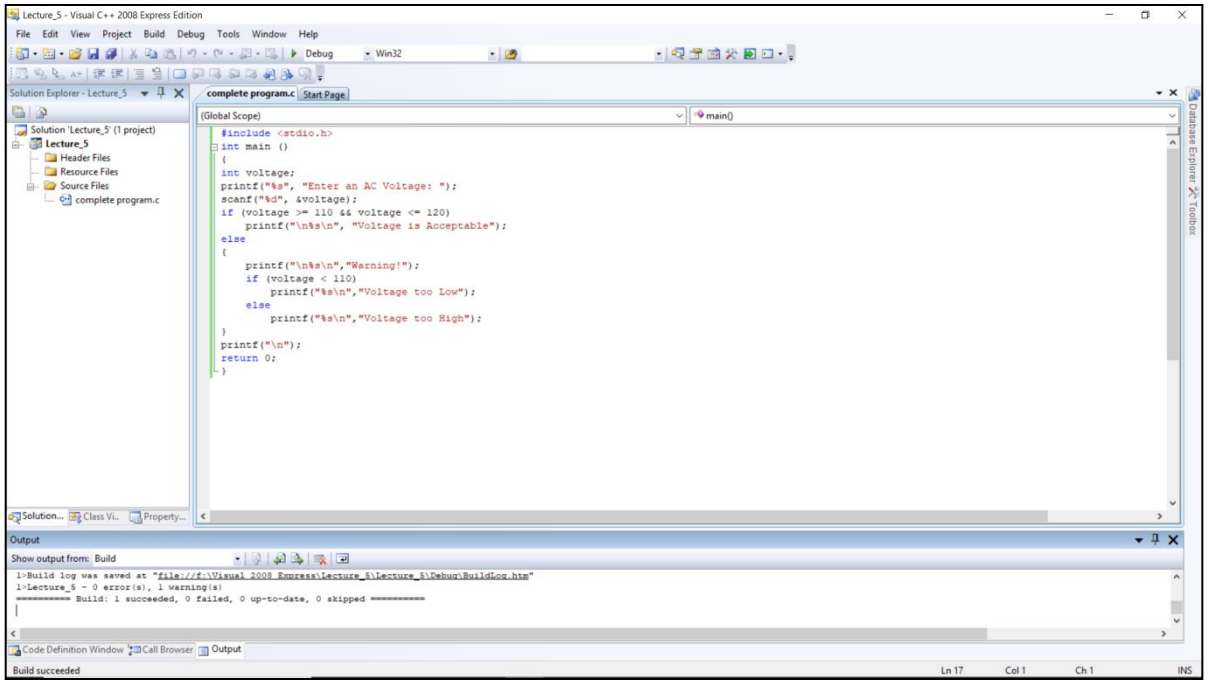
Microprocessor Programming

Practical Works

Using Selection Structures and I/O

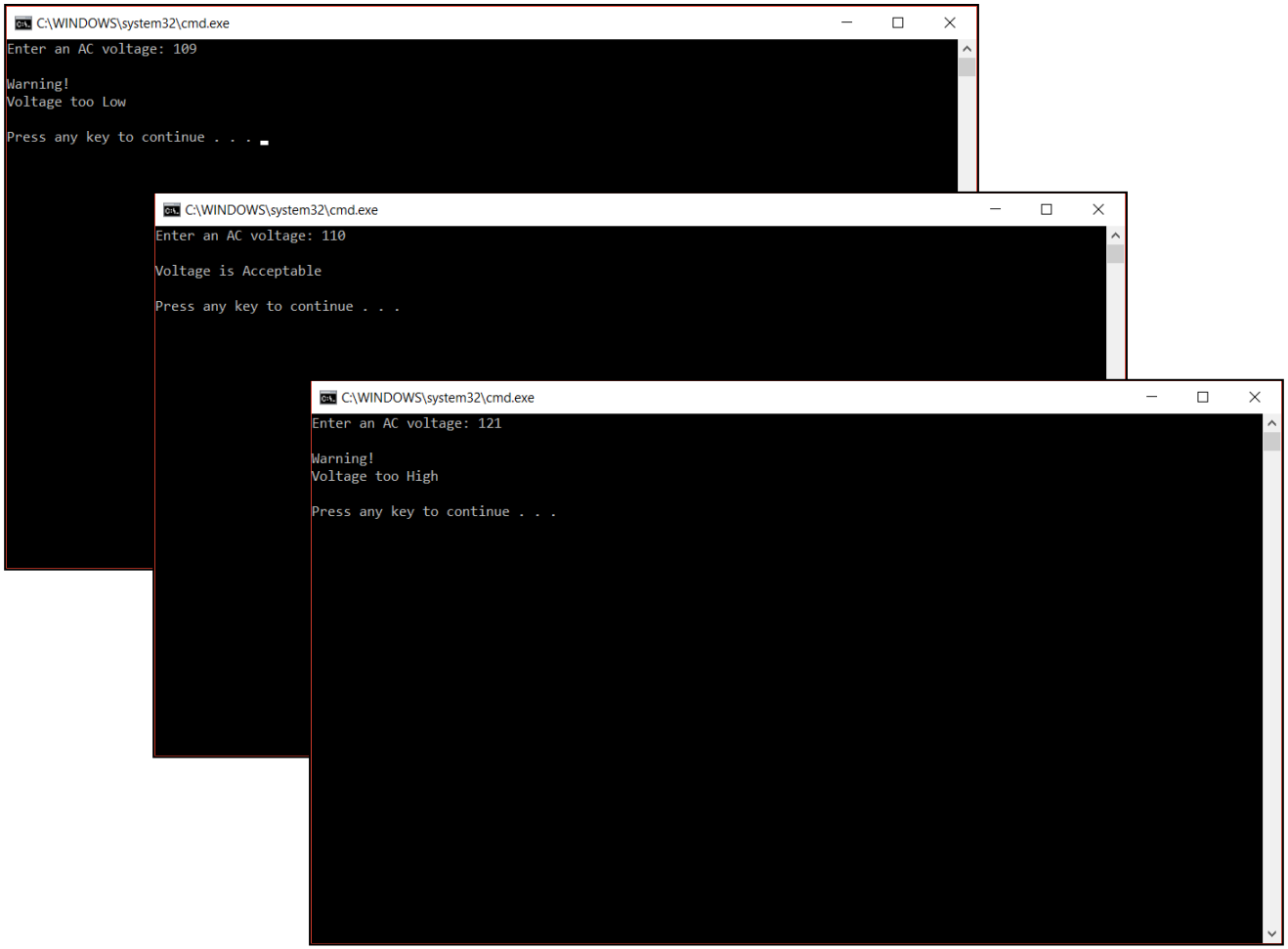
Using Selection Structures and I/O

- Given complete C program is implemented as follow:



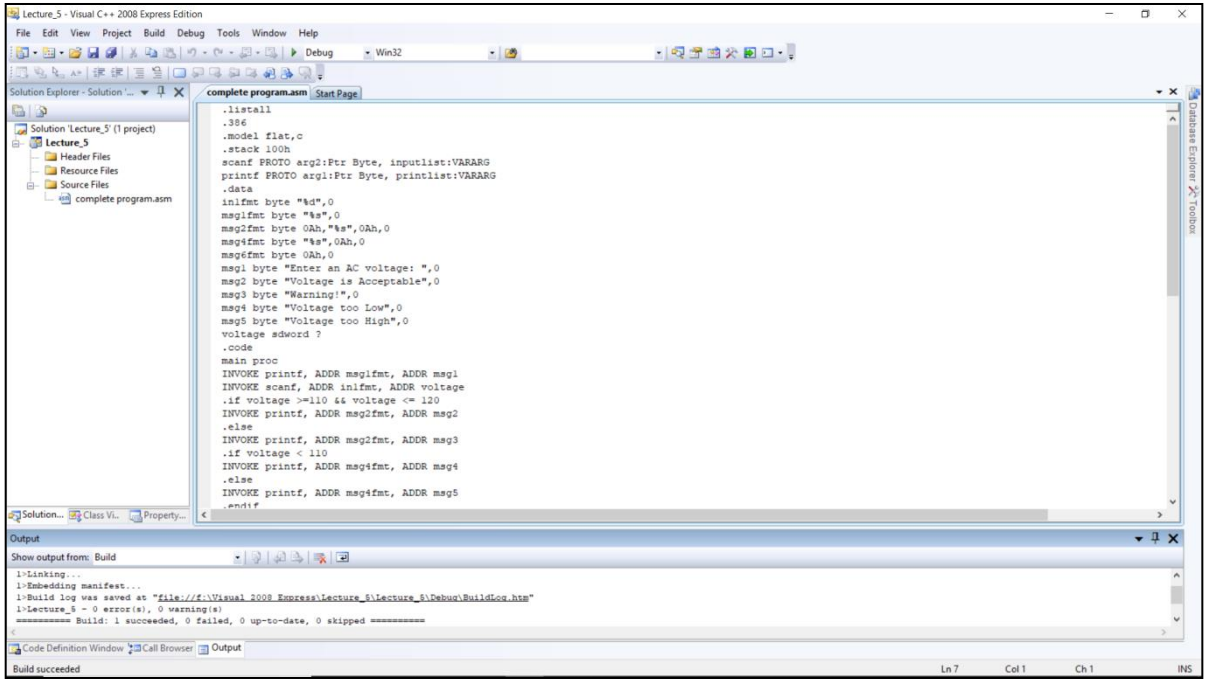
Using Selection Structures and I/O (Cont.)

- For the given complete C program, its outputs are as follows:



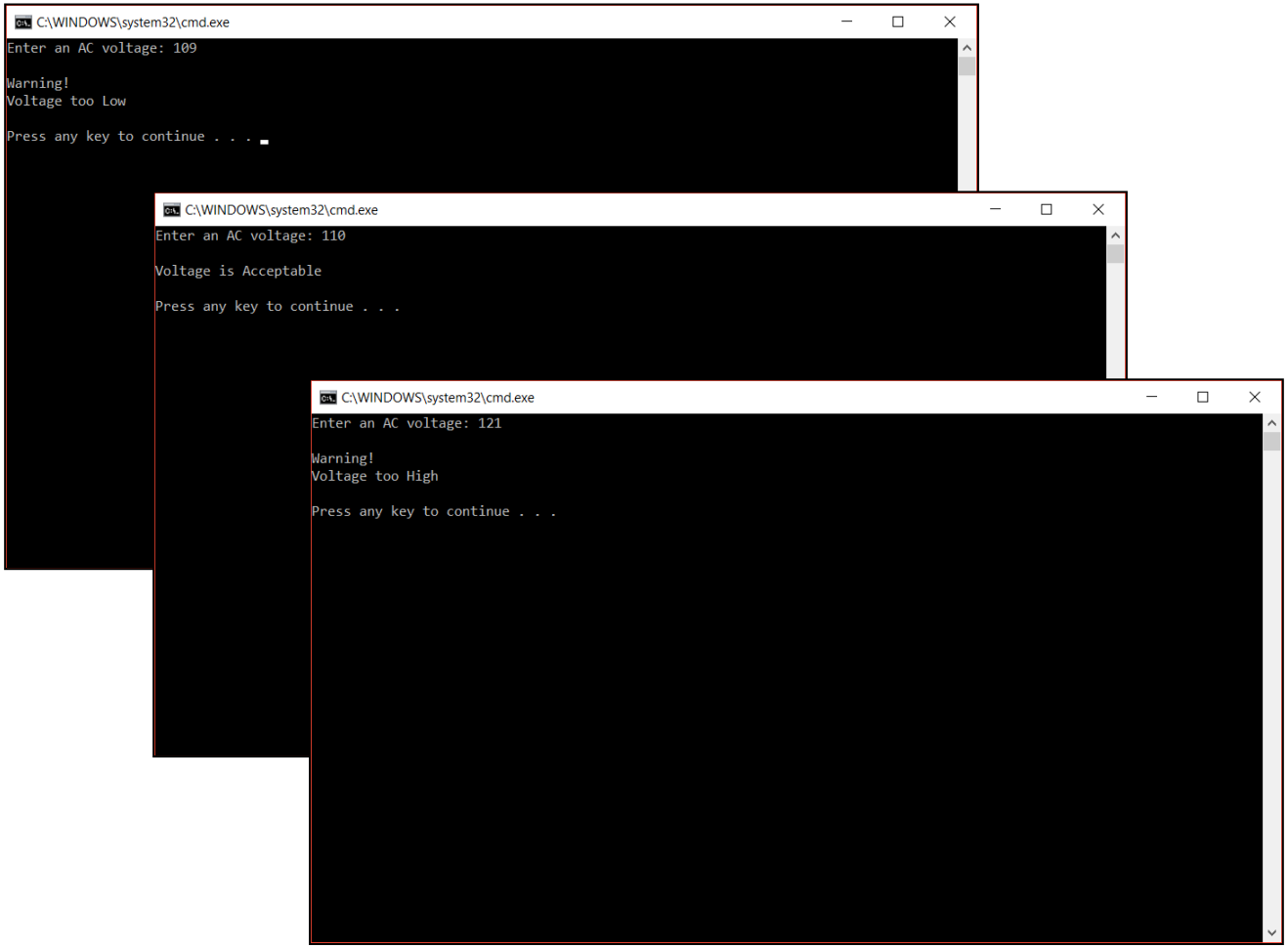
Using Selection Structures and I/O (Cont.)

- We implement the given complete assembly program as follow:



Using Selection Structures and I/O (Cont.)

- For the given complete assembly program, its outputs are as follows:



Microprocessor Programming

Practical Assignments (Instructions)

Assignment 1

- Implement the given complete program in assembly language without using high-level directives with only compares, jumps, and appropriate labels. Also, display its outputs.

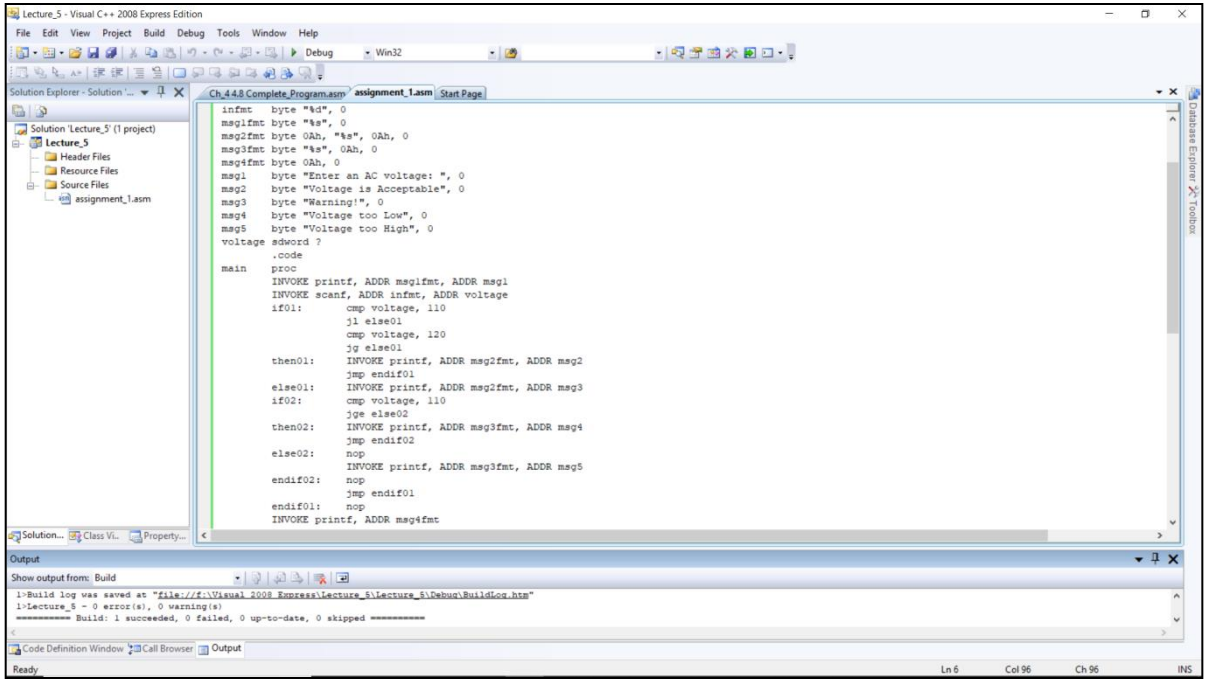
```
            .listall
            .386
            .model flat,c
            .stack 100h
scanf       PROTO arg2:Ptr Byte, inputlist:VARARG
printf      PROTO arg1:Ptr Byte, printlist:VARARG

            .data
in1fmt      byte "%d",0
msg1fmt     byte "%s",0
msg2fmt     byte 0Ah,"%s",0Ah,0
msg4fmt     byte "%s",0Ah,0
msg6fmt     byte 0Ah,0
msg1        byte "Enter an AC voltage: ",0
msg2        byte "Voltage is Acceptable",0
msg3        byte "Warning!",0
msg4        byte "Voltage too Low",0
msg5        byte "Voltage too High",0
voltage     sdword ?

            .code
main        proc
            INVOKE printf, ADDR msg1fmt, ADDR msg1
            INVOKE scanf, ADDR in1fmt, ADDR voltage
            .if voltage >=110 && voltage <= 120
            INVOKE printf, ADDR msg2fmt, ADDR msg2
            .else
            INVOKE printf, ADDR msg2fmt, ADDR msg3
            .if voltage < 110
            INVOKE printf, ADDR msg4fmt, ADDR msg4
            .else
            INVOKE printf, ADDR msg4fmt, ADDR msg5
            .endif
            .endif
            INVOKE printf, ADDR msg6fmt
            ret
main        endp
            end
```

Microprocessor Programming

Practical Assignments (Report)

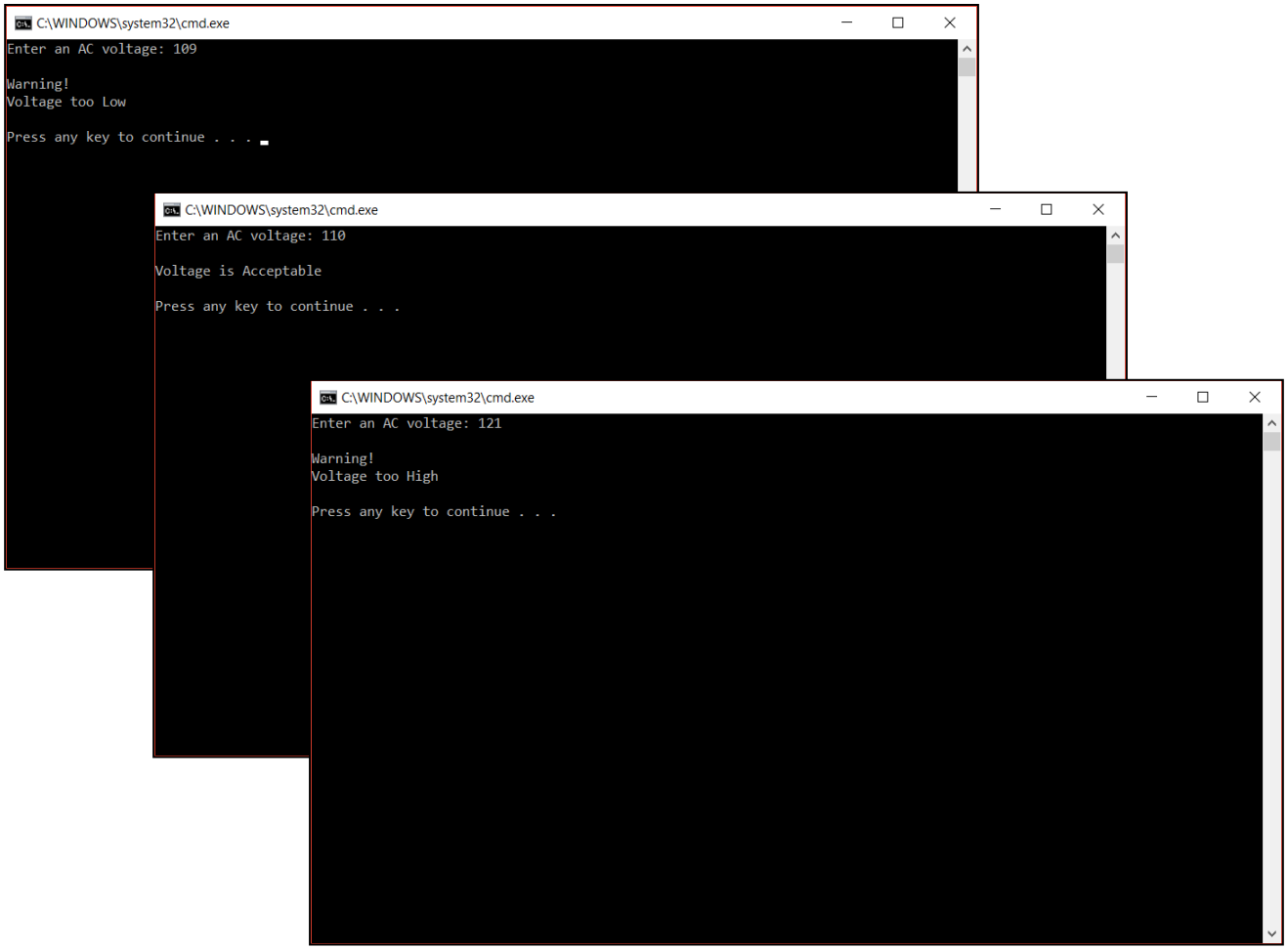
Assignment 1

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



Using Selection Structures and I/O

- For the above assembly program, its outputs are as follows:



Next Lecture

- Introduction to Iteration Structures
- Loop Structures
- Loops and I/O
- Nested Loops

Thank You