Conditional Control Flow Directives

Conditional control flow directives in MASM are used to control the flow of execution of a program depending on the result of a condition. These directives are used to implement conditional statements such as if, else, and elseif.

The following are the most common conditional control flow directives in MASM:

Directive	Description
.BREAK	Generates code to terminate a .WHILE or .REPEAT block
.CONTINUE	Generates code to jump to the top of a .WHILE or .REPEAT block
.ELSE	Begins block of statements to execute when the .IF condition is false
.ELSEIF *condition*	Generates code that tests *condition* and executes statements that follow, until an .ENDIF directive or another .ELSEIF directive is found
.ENDIF	Terminates a block of statements following an .IF, .ELSE, or .ELSEIF directive
.ENDW	Terminates a block of statements following a .WHILE directive
.IF *condition*	Generates code that executes the block of statements if *condition* is true.
.REPEAT	Generates code that repeats execution of the block of statements until *condition* becomes true
.UNTIL *condition*	Generates code that repeats the block of statements between .REPEAT and .UNTIL until *condition* becomes true
.UNTILCXZ	Generates code that repeats the block of statements between .REPEAT and .UNTILCXZ until *CX* equals zero
.WHILE *condition*	Generates code that executes the block of statements between .WHILE and .ENDW as long as *condition* is true

Here are some additional things to keep in mind:

Conditions can be complex, but they must evaluate to a single Boolean value (true or false).

If the condition in the .IF directive is true, the assembler will assemble all of the statements between the .IF and .ELSEIF (or .ENDIF) directives.

If the condition in the .IF directive is false, the assembler will skip all of the statements between the .IF and .ELSEIF (or .ENDIF)

directives.

If the .ELSEIF directive is present, the assembler will evaluate the condition in the .ELSEIF directive.

If the condition is true, the assembler will assemble all of the statements between the .ELSEIF and .ELSE (or .ENDIF) directives.

If the .ELSEIF directive is present and the condition is false, the assembler will skip all of the statements between the .ELSEIF and .ELSE (or .ENDIF) directives.

.ELSE directive is optional. If it is present, the assembler will assemble all of the statements between the .ELSE and .ENDIF directives if all of the previous conditions were false.

The .ENDIF directive is required. It tells the assembler the end of the conditional statement. Here is an example of a more complex conditional statement using the .IF, .ELSEIF, and .ELSE directives:

```
0974 .IF eax > 10000h
0975     mov ECX, AX
0976 .ELSEIF eax > 1000h
0977     mov ECX, 1000h
0978 .ELSE
0979     mov ECX, 0
0980 .ENDIF
```

This code will move the contents of the AX register to the ECX register if the value of AX is greater than 10000h. Otherwise, if the value of AX is greater than 1000h, the code will move the value 1000h to the ECX register. Otherwise, the code will move the value 0 to the ECX register.

TABLE 6-8 Runtime Relational and Logical Operators.

Operator	Description
expr1 == *expr2*	Returns true when *expr1* is equal to *expr2*.
expr1 != *expr2*	Returns true when *expr1* is not equal to *expr2*.
expr1 > *expr2*	Returns true when *expr1* is greater than *expr2*.
expr1 ≥ *expr2*	Returns true when *expr1* is greater than or equal to *expr2*.
expr1 < *expr2*	Returns true when *expr1* is less than *expr2*.
expr1 ≤ *expr2*	Returns true when *expr1* is less than or equal to *expr2*.
! *expr*	Returns true when *expr* is false.
expr1 && *expr2*	Performs logical AND between *expr1* and *expr2*.
expr1 || *expr2*	Performs logical OR between *expr1* and *expr2*.
expr1 & *expr2*	Performs bitwise AND between *expr1* and *expr2*.
CARRY?	Returns true if the Carry flag is set.
OVERFLOW?	Returns true if the Overflow flag is set.
PARITY?	Returns true if the Parity flag is set.
SIGN?	Returns true if the Sign flag is set.
ZERO?	Returns true if the Zero flag is set.

The table you sent shows the relational and logical operators in MASM. These operators are used to compare two values and return a Boolean value (true or false). The Boolean value can then be used to control the flow of execution of a program using conditional control flow directives such as .IF, .ELSE, and .ELSEIF.

The following is a detailed explanation of each of the operators in the table:

== (equal to): Returns true if the two values are equal.

!= (not equal to): Returns true if the two values are not equal.

> (greater than): Returns true if the first value is greater than the second value.

>= (greater than or equal to): Returns true if the first value is greater than or equal to the second value.

< (less than): Returns true if the first value is less than the second value.

<= (less than or equal to): Returns true if the first value is less than or equal to the second value.

&& (logical AND): Returns true if both operands are true.

|| (logical OR): Returns true if either operand is true.

! (logical NOT): Returns true if the operand is false.

The following are some examples of how to use the relational and logical operators in MASM:

```
0984 ; Compare the values of the AX and BX registers.
0985 IF AX > BX
0986     mov ECX, AX
0987 ELSE
0988     mov ECX, BX
0989 ENDIF
0990
0991 ; Compare the values of the val1 and val2 variables.
0992 IF val1 <= 100
0993     mov ECX, 100
0994 ELSE
0995     mov ECX, val1
0996 ENDIF
0997
0998 ; Check if the CARRY flag is set.
0999 IF CARRY?
1000     mov ECX, 1
1001 ELSE
1002     mov ECX, 0
1003 ENDIF
```

Here is a simpler explanation of the notes you provided:

Before using MASM conditional directives, be sure you thoroughly

understand how to implement conditional branching instructions in pure assembly language.

This means that you should understand how to use the following assembly language instructions to implement conditional branching:

- **CMP (compare)**
- **JBE (jump if below or equal)**
- **JA (jump if above)**
- **JE (jump if equal)**
- **JNE (jump if not equal)**

Once you understand how to implement conditional branching in pure assembly language, you can use MASM conditional directives to make your code more concise and readable.

In addition, when a program containing decision directives is assembled, inspect the listing file to make sure the code generated by MASM is what you intended.

MASM conditional directives are translated into assembly language instructions by the assembler. It is a good idea to inspect the listing file to make sure that the assembler generated the code that you expected. This can help you to identify any errors in your code.

Generating ASM Code: When you use a MASM conditional directive such as .IF, the assembler generates assembly language instructions to implement the conditional branching. For example, the following .IF directive:

```
1008 .IF eax > val1
1009     mov result,1
1010 .ENDIF
```

would be expanded by the assembler into the following assembly language instructions:

```
1014 mov eax,6
1015 cmp eax,val1
1016 jbe @C0001
1017 ; jump on unsigned comparison
1018 mov result,1
1019 @C0001:
```

The label name @C0001 is created by the assembler to ensure that all labels within the same procedure are unique.

Controlling Whether or Not MASM-Generated Code Appears in the Source Listing File

You can control whether or not MASM-generated code appears in the source listing file by setting the Enable Assembly Generated Code Listing property in the Visual Studio Project Properties dialog box.

To do this, follow these steps: Open the Visual Studio Project Properties dialog box. Select Microsoft Macro Assembler. Select Listing File. Set the Enable Assembly Generated Code Listing property to Yes. Once you have set this property, the MASM-generated code will be included in the source listing file. This can be helpful for debugging purposes.