7.1 Introduction

The Microsoft Macro Assembler provides two types of conditional directives. Conditional-assembly directives test for a specified condition and assemble a block of statements if the condition is true. Conditional error directives test for a specified condition and generate an error if the condition is true.

Both kinds of conditional directives only test assembly-time conditions. They cannot test run-time conditions since these are not known until an executable program is run. Only expressions that evaluate to constants during assembly can be compared or tested.

Since macros and conditional-assembly directives are often used together, you may need to refer to Chapter 8 to understand some of the examples in this chapter. In particular, conditional directives are frequently used with the special macro operators described in Section 8.3.

7.2 Conditional-Assembly Directives

The conditional-assembly directives include the following:

IF
IFE
IF1
IF2
IFDEF
IFNDEF
IFB
IFNB
IFIDN
IFDIF
ELSE
ENDIF

The **IF** directives and the **ENDIF** and **ELSE** directives can be used to

enclose the statements to be considered for conditional assembly. The conditional block takes the following form:

IF
statements
[**ELSE**
statements]
ENDIF

The *statements* following **IF** can be any valid statements, including other conditional blocks. The **ELSE** directive and its *statements* are optional. **ENDIF** ends the block.

The statements in the conditional block are assembled only if the condition specified by the corresponding **IF** directive is satisfied. If the conditional block contains an **ELSE** directive, only the statements up to the **ELSE** directive will be assembled. The statements following the **ELSE** directive are assembled only if the **IF** condition is not met. An **ENDIF** directive must mark the end of any conditional-assembly block. No more than one **ELSE** directive is allowed for each **IF** directive.

IF directives can be nested up to 255 levels. To avoid ambiguity, a nested **ELSE** directive always belongs to the nearest preceding **IF** directive that does not have its own **ELSE**.

7.2.1 IF and IFE Directives

Syntax

IF *expression*
IFE *expression*

The **IF** and **IFE** directives test the value of an *expression*. The **IF** directive grants assembly if the value of *expression* is true (nonzero). The **IFE** directive grants assembly if the value of *expression* is false (0). The *expression* must resolve to an absolute value and must not contain forward references.

Example

```
IF      debug
        EXTRN dump:FAR
        EXTRN trace:FAR
        EXTRN breakpoint:FAR
ENDIF
```

In this example, the variables within the block will only be declared external if the symbol debug evaluates to true (nonzero).

7.2.2 IF1 and IF2 Directives

Syntax

IF1
IF2

The **IF1** and **IF2** directives test the current assembly pass. The **IF1** directive grants assembly only on Pass 1. **IF2** grants assembly only on Pass 2. The directives take no arguments.

Example

```
IF1
      %OUT Beginning Pass 1
ELSE
      %OUT Beginning Pass 2
ENDIF
```

7.2.3 IFDEF and IFNDEF Directives

Syntax

IFDEF *name*
IFNDEF *name*

The **IFDEF** and **IFNDEF** directives test whether or not the given *name* has been defined. The **IFDEF** directive grants assembly only if *name* is a label, variable, or symbol. The **IFNDEF** directive grants assembly if *name* has not yet been defined.

The name can be any valid name. Note that if *name* is a forward reference, it is considered undefined on Pass 1, but defined on Pass 2.

Example

```
IFDEF   buffer
        buf1    DB 10 DUP(?)
ENDIF
```

In this example, `buf1` is allocated only if `buffer` has been previously defined. One way to use this conditional block would be to leave `buffer` undefined in the source file and define it if you needed it by using the **/D***symbol* option when you start **MASM**. For example, if the conditional block is in `test.asm`, you could start the assembler with the command line:

```
MASM test /Dbuffer;
```

The symbol `buffer` would be defined, and as a result the conditional-assembly block would allocate `buf1`. However, if you didn't need `buf1`, you could use the command line:

```
MASM test;
```

7.2.4 IFB and IFNB Directives

Syntax

IFB <*argument*>
IFNB <*argument*>

The **IFB** and **IFNB** directives test *argument*. The **IFB** directive grants assembly if *argument* is blank. The **IFNB** directive grants assembly if *argument* is not blank. The arguments can be any name, number, or expression. The angle brackets (< >) are required.

The **IFB** and **IFNB** directives are intended for use in macro definitions. They can control conditional-assembly of statements in the macro, based on the parameters passed in the macro call. In such cases, *argument* should be one of the dummy parameters listed by the **MACRO** directive.

Example

```
pushall     MACRO   reg1,reg2,reg3,reg4,reg5,reg6
            IFNB    <reg1>          ;; If parameter not blank
                    push    reg1    ;;   push one register and repeat
                    pushall reg2,reg3,reg4,reg5,reg6
            ENDIF
            ENDM

pushall     ax,bx,si,ds
pushall     cs,es
```

In this example, `pushall` is a recursive macro that continues to call itself until it encounters a blank argument. Any register or list of registers (consisting of up to six registers) can be passed to the macro for pushing.

7.2.5 IFIDN and IFDIF Directives

Syntax

IFIDN <*argument1*>,<*argument2*>
IFDIF <*argument1*>,<*argument2*>

The **IFIDN** and **IFDIF** directives compare *argument1* and *argument2*. The **IFIDN** directive grants assembly if the arguments are identical. The **IFDIF** directive grants assembly if the arguments are different. The arguments can be any names, numbers, or expressions. To be identical, each character in *argument1* must match the corresponding character in *argument2*. Case is significant. The angle brackets (< >) are required. The arguments must be separated by a comma (,).

The **IFIDN** and **IFDIF** directives are intended for use in macro definitions. They can control conditional assembly of macro statements, based on the parameters passed in the macro call. In such cases, the arguments should be dummy parameters listed by the **MACRO** directive.

Example

```
divide    MACRO   numerator, denominator
          IFDIF   <denominator>,<0>  ;; If not dividing by zero
          mov     ax, numerator      ;;   divide AX by BX
          mov     bx, denominator
          div     bx                 ;; Result in accumulator
          ENDIF
          ENDM

divide    6,%test
```

In this example, a macro uses the **IFDIF** directive to check against dividing by a constant that evaluates to 0. The macro is then called, using a percent sign (%) on the second parameter so that the value of the parameter, rather than its name, will be evaluated. See Section 8.3.4 for a discussion of the expression (%) operator.