Assembler Directives

This appendlx describes the most important assembler directives. To explain the syntax, we will use the following notation:

|	separates choices
{ }	enclosed items are optional
[ ]	repeat the enclosed items 0 or more times

If syntax is not given, the directive has no required or optional arguments.

ALPHA

Tells the assembler to arrange segments in alphabetical order. Placed before segment definitions.

ASSUME

Syntax: ASSUME `segment_register:name[,segment_register:name]`

Tells the assembler In associate a segment register with a segment name.

Example: ASSUME `CS:C_SEG, DS:D_SEG, SS:S_SEG, ES:D_SEG`

Note: the name NOTHING cancels the current segment register association. In particular, ASSUME NOTHING cancels segment register associations made by previous ASSUME statements.

.CODE

syntax: `.CODE` (name)

A simplified segment directive (MASM 5.0) for defining a code segment.

.COMM

syntax: `.COMM definition [,definition]`

where definition has the syntax NEAR|FAR label:size[:count]
label is a variable name
size is BYTE, WORD, DWORD, QWORD, or TBYTE
count is the number of elements contained In the variable (default = 1)

Defines a communal variable; such a variable has both PUBLIC and EXTRN attributes, so it can be used in different assembly modulus.

Examples:
```
COMM    NEAR WORD1:WORD
COMM    FAR  ARR1:BYTE:10, ARR2:BYTE:20
```

COMMENT

Syntax:
```
COMMENT delimiter [text]
[text]
delimiter [text]
```

where delimiter is the first nonblank character after the COMMENT directive. Used to define a comment. Causes the assembler to ignore all text between the first and second delimiters. Any text on the same line as the second delimiter is ignored as well.

Examples:

```
COMMENT * Use an asterisk as the delimiter. All this
text is ignored *
COMMENT + This text and the following instruction is ig
nored too +     MOV     AX,BX
```

.CONST

A simplified segment directive for defining a segment containing data that will not he changed by the program. Used mostly in assembly language routines to be called by a high-level language.

.CREF and .XCREF

Syntax:
```
.CREF [name [,name ]]
.XCREF [name [,name] ]
```

In the generation of the cross-reference (.CRF) file, .CREF directs the generation of cross-referencing of names in a program. .CREF with no arguments causes cross-referencing of all names. This is the default directive.
.XCREF turns off cross-referencing in general, or just for the specified names.

Example:

```
XCREF                   ;turns off cross-referencing

CREF                    ;turns cross-referencing back on

XCREF   NAME1,NAME2     ;turns off cross-referencing
                        ;of NAME1 and NAME2
```

.DATA and .DATA?

Simplified segment directives tor defining data segments. .DATA defines an initialized data segment and .DATA? defines an uninitialized data segment. Uninitialized data consist of variables defined with "?". .DATA? is used mostly with assembly language routines to he called from a high-level language. For stand-alone assembly language programs, the .DATA segment may contain uninitialized data.

Data-Defining Directives

Directive	Meaning
DB	define byte
DD	define doubleword (4 bytes)
DF	define farword (6 bytes); used only with 80386 processor
DQ	define quadword (8 bytes)
DT	define tenbyte (10 bytes)
DW	define word (2 bytes)

Syntax: `{name} directive initializer [,initializer]`

where name is a variable name. If name is missing, memory is allocated but no name is associated with it. initializer is a constant, constant expression, or ?. Multiple values may be defined by using the DUP operator. See Chapter 10.

DOSSEG

Tells the assembler to adopt the DOS segment-ordering convention. For a SMALL model program, the order is code, data, stack. This directive should appear before any segment definitions.

ELSE

Used in a conditional block. The syntax is

```
ondition
    statements1
ELSE
    statements2
ENDIF
```

If Condition is true, statements1 are assembled; if Condition is false, statements2 are assembled. See Chapter 13 for the form of Condition.

END

Syntax: `END [start-address]`

Ends a source program. Start-address is a name where execution is to begin when the program is loaded. For a program with only one source module, start-address would ordinarily be the name of the main procedure or a label indicating the first instruction. For a program with several modules, each module must have an END but only one of them can specify a start-address.

ENDIF

Ends a conditional block. See Chapter 13.

ENDM

Ends a macro or repeat block. See MACRO and REPT.

ENDP

Ends a procedure. See PROC.

ENDS

Ends a segment or structure. See SEGMENT and STRUC

EQU

Syntax: There are two forms, numeric equates and string equates. A numeric equate has the form

```
name  EQU  numeric-expression
```

A string equate has the form

```
name  EQU  <string>
```

The EQU directive assigns the expression following EQU to the constant symbol name. Numeric_expression must evaluate to a number. The assembler replaces each occurrence of name in a program by numeric_expression or string. No memory is allocated for name. Name may not be redefined.

Examples:

```
MAX                    EQU 32767
MIN                    EQU MAX - 10
PROMPT                 EQU <'type a line of text:$'>
ARG                    EQU <[DI + 2]>
```

Use in a program:

```
.DATA
                MSG  DB   PROMPT
.CODE
MA , NPROC
                MOV  AX,MIN    ;equivalent to MOV AX,32757
                MOV BX, ARG    ;equivalent to MOV BX, [DI+2]
```

= (equal)

Syntax: `name = expression`

where expression is an integer, constant expression, or a one or two-character string constant.
The directive = works like EQU, except that names defined with = can bc redefined later in a program.

Examples:

```
CTR    1
                MOV  AX,CTR    ;translates to MOV AX,1
CTR = CTR + 5
                MO"  RX, CTR   ;translates to MOV BX,6
```

'The = directive is often used in macros. See Chapter 13.

.ERR Directives

These are conditional error directives that can be used to force the assembler to display an error message during assembly, for debugging purposes. The assembler displays the message "Forced error", with an identifying number. See Chapter 13.

Directive	*Number*	*Forced* ermr *if*
.ERR1	a7	encountered during assembly pass 1
ERR2	88	encountered during assembly pass 2
.ERR	a9	encountered
.ERRE expression	90	expression is false (0)

ERRNZ expression	91	expression is true (nonzero)
ERRNDEF name	92	name has not been defined.
ERRDEF name	93	name has been defined.
ERRB <argument>	94	argument is blank
ERRNB <argument>	95	argument is not blank
.ERRIDN <arg1>,<arg2>	96	arg1 and arg2 are identical
ERRDIF <arg1>,<arg2>	97	arg1 and arg2 are different

EVEN

Advances the location counter to the next even address.

EXITM

Used in a macro or repeat block. Tells the assembler to terminate the macro or repeat block expansion.

EXTRN

Syntax: `EXTRN name:type [,name:type]`

Informs the assembler of an external name; that is, a name defined in another module. Type must match the type declared for the name in the other module. Type can be NEAR, FAR, PROC, BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, or ABS. See Chapter 14.

.FARADATA and .FARDATA?

Syntax:
```
.FARDATA {name}
.FARDATA? {name}
```

Used primarily with compilers for defining extra data segments.

GROUP

Syntax: `name GROUP segment [,segment]`

A group is a collection of segments that are associated with the same starting address. Variables and labels defined in the segments of the group are assigned addresses relative to the start of the group, rather than relative to the beginning of the segments in which they arc defined. 'This makes it possible to refer to all the data in the group by initlalizing a single segment register. *Note:* the same result can be obtained by giving the same name and a PUBLIC attribute to all the segments.

IF directives

These directives arc used to grant the assembler permission to assemble the statements following the directives, depending on conditions. A list may he found in Chapter 13.

INCLUDE

Syntax: `INCLUDE filespec`

whcrr filespec specifies a file containing valid assembly language statements. In addition to a file name, filespec may Include a drive and path. The directlve causes the assembler to insert the contents of the file at the position of the INCLUDE in the source file, and to begin processing the file's statements.

Examples:
```
INCLUDE MACLIB
INCLUDE C:\BIN\PROG1.ASM
```

LABEL

Syntax: `name LABEL type`

where type is BYTE, WORD, DWORD, FWORD, QWORD, TBYTE, or the name of a previously-defined structure.
This directive provides a way to define or redefine the type associated with a name.

Example:

```
WORD-ARR        LABEL    WORD
BYTE  ARR       DB  100 DUP (0)
```

Here WORD_ARR defines a SO-word array, and BYTE_ARR defines a 100-byte array. The same address is assigned to both variables.

.LALL

Causes the assembler to list all statements in a macro expansion, except those preceded by a double semicolon.

.LIST and .XLIST

.LIST causes the assembler to include the statements following the .LIST directive in the source program listing. .XLIST causes the listing of the statements following the .XLIST directive to be suppressed.

LOCAL

Syntax: `LOCAL name [,name]`

Used inside a macro. Each time the assembler encounters a LOCAL name during macro expansion, it replaces it by a unique name of form ??number. In this way duplicate names are avoided if the macro is called several times in a program. See Chapter 13.

MACRO and ENDM

Syntax: `name MACRO [parameter [,parameter]]`

These directives are used to define a macro,

Example:

```
EXCHANGE        MACRO           WORD1,WORD2
                PUSH   WORD1
                PUSH   WORD2
                POP    WORD1
                POP    CORD2
                ENDM
```

See Chapter 13.

.MODEL

Syntax: `. MODEL memory_model`

A simplified segment directive for defining a memory model. Memory model can be any of the following:

Mode/	Description
TINY	code and data in one segment
SMALL	code in one segment data in one segment
MEDIUM	code in more than one segment data in one segment
COMPACT	code in one segment data in more than one segment
LARGE	code in more than one segment data in more than one segment no array larger than 64 KB
HUGE	code in more than one segment data in more than one segment arrays may be larger than 64 KB

ORG

Syntax: ORG expression

where expression must resolve to a 2-byte number.
Sets the location counter to the value of expression. For example, in a .COM program, the directive ORG 100h sets the location counter to 100h, so that variables will be assigned addresses relative to the start of the program, rather than in the 100h-byte program segment prefix, which precedes the program in *memory*. Another use of ORG is to define a data area that can be shared by several variables. For example,

```
.DATA
WORD1_ARR  DW    100 CUP (?)
ORG  0
WORD2_ARR  DW    50 DUP (?)
WORD3 ARR  DW    50 DUP (?)
```

This definition causes WORD2_ARR and the first 50 words in WORD1_ARR to occupy the same memory space. Similarly, WORD3_ARR and The last 50 words of WORD1_ARR occupy the same space.

%OUT

syntax: %OUT text

where text is a line of ASCII characters.
Used to display a message at a specified place in an assembly listing. Often used during conditional assembly.

Example:

```
IFNDEF          SUM
                SUM  DW        ?
                %OUT   SUM is defined here
ENDIF
```

If SUM had not been previously defined, it would be defined here and the message would be displayed.

PAGE

Syntax: PAGE [[length],width]

where length is IO-255 and width is 60-132. Default values are length = 50 and width = 80.
Used to create a page break or to specify the maximum number of lines per page and the maximum number of characters per line in a program listing.

Examples:

```
PAGE                    ;creates a page break
PAGE  50,70             ;sets maximum page length to 50
                        ;and maximum page width to 70
PAGE  ,132              ;sets maximum page width to 132
```

PROC and ENDP

Syntax:
```
name  PROC  distance
name  ENDP
```

where distance is NEAR or FAR. Default is NEAR.
Used to begin and end a procedure definition. See Chapter 8.

Processor and Coprocessor Directives

The following directives define the instruction set recognized by MASM. These directives must be placed outside segment declarations. In the following, 8086 includes 8088. 8087, 80287, and 80387 are coprocessors.

Directive	*Enables* assembly *of* instructions for *processors end coprocessors*
.8086	8086, 8087
.186	8086, 8087, and 80186 additional instructions
.286	8086, 80287, and additional 80286 nonprivileged instructions
286P	same as .286 plus 80286 privileged instructions
.386	8086, 80387 and 80286 and 80386 nonprivileged instructions
.386P	same as .386 plus 80386 privileged instructions
.8087	8087; disables instructions unique to the 80287 and 80387
,287	8087, and 80287 additional instructions
.387	8087, 80287, and 80387 additional instructions

PUBLIC

Syntax: PUBLIC name [,name]

where name is a variable, label, or numeric equate defined in the module containing the directive.
Used to make names in this module available for use in other modules. Not to be confused with the PUBLIC combine-type, which is used to combine segments. See Chapter 14.

PURGE

Syntax: PURGE macroname [,macroname]

where macroname is the name of a macro,

Used to delete macros from memory during assembly. This might be necessary if the system does not have enough memory to keep all the macros a program needs in memory at the same time.

Example:

```
MAC1                    ;expand macro MAC1
PURGE MAC1              ;don't need it anymore
```

.RADIX

syntax: `.RADIX base`

where base is a decimal number in the range 2–16.
Specifies the default radix for representation of integer constants. This means that in the absence of a "b", "d", or "h" as the last character in the representation of an integer, the assembler will assume the number is represented in the base specified by the directive. The default is 10 (decimal).

Examples:

```
. DATA
.RADIX 16               ;hexadecimal
A    DW    1101         ; interpreted as 1101h
.RADIX 2
B    DW    1101         ;interpreted as 1101b
```

RECORD

Used to define a record variable. This is a byte or word variable in which specific bit fields can be accessed symbolically. See the Microsoft Macro Assembler Programmer's Guide.

REPT and ENDM

syntax:
```
REPT expression
  statements
ENDM
```

where expression must evaluate to a 16-bit unsigned number.
Defines a repeat block. REPT causes the statements in the block to be assembled the number of times equal to the value of expression. A repeat block can be placed at the position where the statements are to be repeated, or it can be put inside a macro. See Chapter 13.

.SALL

Causes the assembler to suppress the listing of macro expansions.

SEGMENT end ENDS

Syntax:
```
name SEGMENT (align) (combine) ('class')
     statements
name ENDS
```

where
align is PARA, BYTE, WORD, or PAGE
combine is PUBLIC, COMMON, STACK, or AT
class is a name enclosed in single quotes

These directive define a program segment. Align, combine, and class specify how the segment will be aligned in memory, combined with other segments, and ordered with respect to other segments. See Chapter 14.

.SEQ

Directs the assembler to leave the segments in their original order. Has the same effect as .ALPHA.

.STACK

Syntax: .STACK (size)

where size is a positive integer.
A simplified segment directive which defines 3 stack segment of size bytes. Default size is 1 kilobyte.

STRUC and ENDS

Used to declare a structure. This is a collection of data objects that can he accessed symbolically as a single data object. See the *Microsoft Macro Assembler Programmer's Guide.*

SUBTTL

Syntax: SUBTTL [text]

Causes a subtitle of up to GO characters to be placed on the third line of each page in an assembly listing. May be used more than once.

TITLE

Syntax: TITLE [text]

Causes a title to be placed on each page of an assembly listing. May be used only once.

.XALL

Causes the assembler to list all statements in a macro expansion that produce code. Comments are suppressed.

.XCREF

See .CREF.

.XLIST

See .LIST.