8.7 DC — Define Constant

Syntax: [<label>:] `DC` [<size>] <expression> [, <expression>]...

where

<size> = `B` (default), `W`, or `L`.

Description: The `DC` directive defines constants in memory. It can have one or more <expression> operands, which are separated by commas. The <expression> can contain an actual value (binary, octal, decimal, hexadecimal, or ASCII). Alternately, the <expression> can be a symbol or expression that can be evaluated by the assembler as an absolute or simple relocatable expression. One memory block is allocated and initialized for each expression.

These rules apply to size specifications for `DC` directives:

- `DC.B` — One byte is allocated for numeric expressions. One byte is allocated per ASCII character for strings.
- `DC.W` — Two bytes are allocated for numeric expressions. ASCII strings are right aligned on a 2-byte boundary.
- `DC.L` — Four bytes are allocated for numeric expressions. ASCII strings are right aligned on a 4-byte boundary.

Example for `DC.B`:

```
000000 4142 4344   Label: DC.B "ABCDE"
000004 45
000005 0A0A 010A          DC.B %1010, @12, 1, $A
000009 xx                 DC.B PAGE(Label)
```

Example for `DC.W`:

```
000000 0041 4243   Label: DC.W "ABCDE"
000004 4445
000006 000A 000A          DC.W %1010, @12, 1, $A
00000A 0001 000A
00000E xxxx               DC.W Label
```

Example for `DC.L`:

```
000000 0000 0041   Label: DC.L "ABCDE"
000004 4243 4445
000008 0000 000A         DC.L %1010, @12, 1, $A
00000C 0000 000A
000010 0000 0001
000014 0000 000A
000018 xxxx xxxx           DC.L Label
```

If the value in an operand expression exceeds the size of the operand, the value is truncated and a warning message is generated.

8.8 DCB — Define Constant Block

Syntax: [<label>:] `DCB` [<size>] <count>, <value>

where

<size> = B (default), W, or L

Description: The `DCB` directive causes the assembler to allocate a memory block initialized with the specified <value>. The length of the block is <size> * <count>.

<count> may not contain undefined, forward, or external references. It may range from 1 to 4096.

The value of each storage unit allocated is the sign-extended expression <value>, which may contain forward references. The <count> cannot be relocatable. This directive does not perform alignment.

These rules apply to size specifications for `DCB` directives:

- `DCB.B` — One byte is allocated for numeric expressions.
- `DCB.W` — Two bytes are allocated for numeric expressions.
- `DCB.L` — Four bytes are allocated for numeric expressions.

Example:

```
000000 FFFF FF     Label: DCB.B 3, $FF
000003 FFFE FFFE          DCB.W 3, $FFFE
000007 FFFE
000009 0000 FFFE          DCB.L 3, $FFFE
00000D 0000 FFFE
000011 0000 FFFE
```

8.9 DS — Define Space

Syntax: [<label>:] DS [.<size>] <count>

where

<size> = B (default), W, or L

Description:

The DS directive is used to reserve memory for variables. The content of the reserved memory is not initialized. The length of the block is <size> * <count>.

<count> may not contain undefined, forward, or external references. It may range from 1 to 4096.

Example:

```
Counter: DS.B  2   ; 2 contiguous bytes in memory
         DS.B  2   ; 2 contiguous bytes in memory
                   ; can only be accessed through the
                   ; label Counter
         DS.L  5   ; 5 contiguous longwords in memory
```

The label, `Counter,` references the lowest address of the defined storage area.

8.11 END — End Assembly

Syntax: `END`

Description: The `END` directive indicates the end of the source code. Subsequent source statements in this file are ignored. An `END` directive in included files causes subsequent source statements in the include file to be skipped.

Example: When assembling the code:

```
Label: NOP
       NOP
       NOP
       END

       NOP ; No code generated
       NOP ; No code generated
```

The generated listing file is:

```
000000 A7      Label: NOP
000001 A7             NOP
000002 A7             NOP
                      END
```

8.12 ENDIF — End Conditional Assembly

Syntax: `ENDIF`

Description: The `ENDIF` directive indicates the end of a conditional block. Nesting of conditional blocks is allowed. The maximum level of nesting is limited by the available memory at assembly time.

Example: See an example of directive `IF` in **8.17 IF — Conditional Assembly**.

8.13 ENDM — End Macro Definition

Syntax: `ENDM`

Description: The `ENDM` directive terminates both the macro definition and macro expansion.

Example:

```
5    5          cpChar: MACRO   ; start macro definition
6    6                               LDD \1
7    7                               STD \2
8    8           ENDM           ; end of macro definition
9    9          codeSec: SECTION
10   10         Start:
11   11                         cpChar char1, char2
12    6m  000000 FC xxxx  +          LDD char1
13    7m  000003 7C xxxx  +          STD char2
14   12   000006 A7                NOP
15   13   000007 A7                NOP
```

8.14 EQU — Equate Symbol Value

Syntax: <label>: `EQU` <expression>

Description: The `EQU` directive assigns the value of the <expression> in the operand field to <label>. The <label> and <expression> fields are both required, and the <label> cannot be defined anywhere else in the program. The <expression> cannot include a symbol that is undefined or not yet defined.

The `EQU` directive does not allow forward references.

Example:

```
         0000 0014  MaxElement: EQU  20
         0000 0050  MaxSize:    EQU  MaxElement * 4

  000000            Time:   DS.W 3
         0000 0000  Hour:   EQU Time  ; first word addr
         0000 0002  Minute: EQU Time+2; second word addr
         0000 0004  Second: EQU Time+4; third word addr
```

8.19 INCLUDE — Include Text from Another File

Syntax:	`INCLUDE` <filename>
Description:	This directive causes the included file to be inserted in the source input stream. The <file specification> is not case sensitive and must be enclosed in quotation marks. The assembler attempts to open <filename> relative to the current working directory. If the file is not found, then it is searched for in each path specified in the environment variable GENPATH.
Example:	`INCLUDE "..\LIBRARY\macros.inc"`

8.23 MACRO — Begin Macro Definition

Syntax: <label>: `MACRO`

Description: The <label> of the `MACRO` directive is the name by which the macro is called. This name must not be a processor machine instruction or assembler directive name. For more information on macros, refer to **Section 9. Macros**.

Example:

```
 5   5             cpChar: MACRO; start macro definition
 6   6                     LDD \1
 7   7                     STD \2
 8   8                     ENDM ; end of macro definition
 9   9             codeSec: SECTION
10  10             Start:
11  11             cpChar char1, char2
12  6m    000000 FC xxxx  +          LDD char1
13  7m    000003 7C xxxx  +          STD char2
14  12    000006 A7               NOP
15  13    000007 A7               NOP
```

8.27 NOPAGE — Disable Paging

Syntax: `NOPAGE`

Description: Disables pagination in the listing file. Program lines are listed continuously without headings or top or bottom margins.

8.28 ORG — Set Location Counter

Syntax: `ORG` <expression>

Description: The `ORG` directive sets the location counter to the value specified by <expression>. Subsequent statements are assigned memory locations starting with the new location counter value. The <expression> must be absolute and may not contain any forward, undefined, or external references. The `ORG` directive generates an internal section, which is absolute.

Example:

```
        org   $2000
b1:     nop
b2:     rts
```

8.29 OFFSET — Create Absolute Symbols

Syntax: `OFFSET` <expression>

Description: The `OFFSET` directive declares an offset section and initializes the location counter to the value specified in <expression>. The <expression> must be absolute and may not contain references to external, undefined, or forward defined labels.

The `OFFSET` section is useful to simulate data structure or a stack frame.

Example: The following example shows how `OFFSET` can be used to access elements of a structure.

```
6    6                                OFFSET 0
7    7  000000        ID:      DS.B     1
8    8  000001        COUNT:   DS.W     1
9    9  000003        VALUE:   DS.L     1
10   10 0000 0007     SIZE:    EQU *
11   11
12   12               DataSec:SECTION
13   13 000000        Struct: DS.B SIZE
14   14
15   15               CodeSec:SECTION
16   16               entry:
17   17 000003 CE xxxx         LDX  #Struct
18   18 000006 8600            LDAA #0
19   19 000008 6A00            STAA ID, X
20   20 00000A 6201            INC COUNT, X
21   21 00000C 42              INCA
22   22 00000D 6A03            STAA VALUE, X
```

As soon as a statement affecting the location counter (other than `EVEN`, `LONGEVEN`, `ALIGN`, or `DS`) is encountered after the `OFFSET` directive, the offset section is ended. The preceding section is activated again, and the location counter is restored to the next available location in this section.

8.31 PLEN — Set Page Length

Syntax: `PLEN` <n>

Description: Sets the page length to <n> lines. <n> may range from 10 to 10,000. If the number of lines already listed on the current page is greater than or equal to <n>, listing will continue on the next page with the new page length setting. The default page length is 65 lines.

8.32 SECTION — Declare Relocatable Section

Syntax: <name>: `SECTION [SHORT]`[<number>]

Description: This directive declares a relocatable section and initializes the location counter for the following code. The first `SECTION` directive for a section sets the location counter to 0. Subsequent `SECTION` directives for that section restore the location counter to the value that follows the address of the last code in the section.

<name> is the name assigned to the section. Two `SECTION` directives, where the same name is specified, refer to the same section.

<number> is optional and only specified for compatibility with the MASM assembler.

A section is a code section if it contains at least an assembly instruction. It is considered to be a constant section if it contains only `DC` or `DCB` directives. A section is considered to be a data section if it contains at least a `DS` directive or if it is empty.

Example: The following example demonstrates the definition of a section `aaa`, which is split into two blocks, with section `bbb` between them. The location counter associated with label `zz` is 1, because a `NOP` instruction was already defined in this section at label `xx`.

```
2    2                       aaa:    section 4
3    3   000000 A7           xx:      nop
4    4                       bbb:    section 5
5    5   000000 A7           yy:     nop
6    6   000001 A7                   nop
7    7   000002 A7                   nop
8    8                       aaa:    section 4
9    9   000001 A7           zz:      nop
```

The optional qualifier `SHORT` specifies that the section is a short section. Objects defined there can be accessed using the direct addressing mode.

Example: The following example demonstrates the definition and usage of a `SHORT` section. On line number 12, the symbol `data` is accessed using the direct addressing mode.

```
2    2                     dataSec: SECTION  SHORT
3    3   000000            data: DS.B 1
4    4
5    5       0000 0AFE     initSP:  EQU $AFE
6    6
7    7                     codeSec: SECTION
8    8
9    9                     entry:
10   10   000000 CF 0AFE           LDS  #initSP
11   11   000003 C600              LDAB #0
12   12   000005 5Bxx              STAB data
```

8.33 SET — Set Symbol Value

Syntax: <label>: `SET` <expression>

Description: Similar to the `EQU` directive, the `SET` directive assigns the value of the <expression> in the operand field to the symbol in the <label> field. The <expression> cannot include a symbol that is undefined or not yet defined. The <label> is an assembly time constant; `SET` does not generate machine code.

The value is temporary; a subsequent `SET` directive can redefine it.

Example:

```
2    2          0000 0002   count: SET 2
3    3   000000 02          loop:  DC.B count
4    4          0000 0002          IFNE count
5    5          0000 0001   count: SET count - 1
6    6                             ENDIF
7    7   000001 01                 DC.B count
8    8          0000 0001          IFNE count
9    9          0000 0000   count: SET count - 1
10   10                             ENDIF
11   11   000002 00                 DC.B count
12   12          0000 0000          IFNE count
```

The value associated with the label count is decremented after each `DC.B` instruction.