DB: Define Byte The DB directive is used to reserve byte or bytes of memory locations in the available memory. While preparing the EXE file, this directive directs the assembler to allocate the specified number of memory bytes to the said data type that may be a constant, variable, string, etc. Another option of this directive also initialises the reserved memory bytes with the ASCII codes of the characters specified as a string. The following examples show how the DB directive is used for different purposes.

Example 2.47

```
RANKS    DB 01H, 02H, 03H, 04H
```

This statement directs the assembler to reserve four memory locations for a list named RANKS and initialise them with the above specified four values.

```
MESSAGE DB 'GOOD MORNING'
```

This makes the assembler reserve the number of bytes of memory equal to the number of characters in the string named MESSAGE and initialise those locations by the ASCII equivalent of these characters.

```
VALUE    DB  50H
```

This statement directs the assembler to reserve 50H memory bytes and leave them uninitialised for the variable named VALUE.

DW: Define Word The DW directive serves the same purposes as the DB directive, but it now makes the assembler reserve the number of memory words (16-bit) instead of bytes. Some examples are given to explain this directive.

Example 2.48

```
WORDS    DW  1234H, 4567H, 78ABH, 045CH,
```

This makes the assembler reserve four words in memory (8 bytes), and initialize the words with the specified values in the statements. During initialisation, the lower bytes are stored at the lower memory addresses, while the upper bytes are stored at the higher addresses. Another option of the DW directive is explained with the DUP operator.

```
WDATA    DW 5 DUP (6666H)
```

This statement reserves five words, i.e. 10-bytes of memory for a word lable WDATA and initialises all the word locations with 6666H.

DQ: Define Quadword This directive is used to direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and may initialise it with the specified values.

DT: Define Ten Bytes The DT directive directs the assembler to define the specified variable requiring 10-bytes for its storage and initialise the 10-bytes with the specified values. The directive may be used in case of variables facing heavy numerical calculations, generally processed by numerical processors.

ASSUME: Assume Logical Segment Name The ASSUME directive is used to inform the assemble, the names of the logicals segments to be assumed for different segments used in the program. In the assembly language program, each segment is given a name. For example, the code segment may be given the name CODE, data segment may be given the name DATA etc. The statement ASSUME CS : CODE directs the assembler that the machine codes are available in a segment named CODE, and hence the CS register is to be loaded with the address (segment) allotted by the operating system for the label CODE, while loading. Similary, ASSUME DS : DATA indicates to the assembler that the data items related to the program, are available in a logical segment named DATA, and the DS register is to be initialised by the segment address

value decided by the operating system for the data segment, while loading. It then considers the segment DATA as a default data segment for each memory operation, related to the data and the segment CODE as a source segment for the machine codes of the program. The ASSUME statement is a must at the starting of each assembly language program, without which a message 'CODE/DATA EMITTED WITHOUT SEGMENT' may be issued by an assembler.

END: END of Program The END directive marks the end of an assembly language program. When the assembler comes across this END directive, it ignores the source lines available later on. Hence, it should be ensured that the END statement should be the last statement in the file and should not appear in between. Also, no useful program statement should lie in the file, after the END statement.

ENDP: END of Procedure In assembly language programming, the subroutines are called procedures. They may be independent program modules which return particular results or values to the calling programs. The ENDP directive is used to indicate the end of a procedure. A procedure is usually assigned a name, i.e. label. To mark the end of a particular procedure, the name of the procedure, i.e. label may appear as a prefix with the directive ENDP. The statements, appearing in the same module but after the ENDP directive, are neglected from that procedure. The structure given below explains the use of ENDP.

```
PROCEDURE STAR
        ⋮
STAR ENDP
```

ENDS: END of Segment This directive marks the end of a logical segment. The logical segments are assigned with the names using the ASSUME directive. The names appear with the ENDS directive as prefixes to mark the end of those particular segments. Whatever are the contents of the segments, they should appear in the program before ENDS. Any statement appearing after ENDS will be neglected from the segment. The structure shown below explains the fact more clearly.

```
DATA        SEGMENT
              ⋮
DATA        ENDS
ASSUME      CS : CODE, DS : DATA
CODE        SEGMENT
              ⋮
CODE        ENDS
END
```

The above structure represents a simple program containing two segments named DATA and CODE. The data related to the program must lie between the DATA SEGMENT and DATA ENDS statements. Similary, all the executable instructions must lie between CODE SEGMENT and CODE ENDS statements.

EVEN: Align on Even Memory Address The assembler, while starting the assembling procedure of any program, initialises a location counter and goes on updating it, as the assembly proceeds. It goes on assigning the available addresses, i.e. the contents of the location counter, sequentially to the program variables, constants and modules as per their requirements, in the sequence in which they appear in the program. The EVEN directive updates the location counter to the next even address, if the current location counter contents are not even, and assigns the following routine or variable or constant to that address. The structure given below explains the directive.

```
        EVEN
PROCEDURE       ROOT
                 :
        ROOT    ENDP
```

The above structure shows a procedure ROOT that is to be aligned at an even address. The assembler will start assembling the main program calling ROOT. When the assembler comes across the directive EVEN, it checks the contents of the location counter. If it is odd, it is updated to the next even value and then the ROOT procedure is assigned to that address, i.e. the updated contents of the location counter. If the content of the location counter is already even, then the ROOT procedure will be assigned with the same address.

EQU: Equate The directive EQU is used to assign a label with a value or a symbol. The use of this directive is just to reduce the recurrence of the numerical values or constants in a program code. The recurring value is assigned with a label, and that label is used in place of that numerical value, throughout the program. While assembling, whenever the assembler comes across the label, it substitutes the numerical value for that label and finds out the equivalent code. Using the EQU directive, even an instruction mnemonic can be assigned with a label, which can then be used in the program in place of that mnemonic. Suppose, a numerical constant which appears in a program ten times. If that constant is to be changed at a later time, one will have to make the correction 10 times. This may lead to human errors, because it is possible that a human programmer may miss one of those corrections. This will result in the generation of wrong codes. If the EQU directive is used to assign the value with a label that can be used in place of each recurrence of that constant, only one change in the EQU statement will give the correct and modified code. The examples given below show the syntax.

Example 2.49

```
LABEL     EQU   0500H
ADDITION  EQU   ADD
```

The first statement assigns the constant 500H with the label LABEL, while the second statement assigns another label ADDITION with mnemonic ADD.

EXTRN: External and PUBLIC: Public The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have already been defined in some other assembly language modules. While in the other module, where the names, procedures and labels actually appear, they must be declared public, using the PUBLIC directive. If one wants to call a procedure FACTORIAL appearing in MODULE1 from MODULE 2; in MODULE1, it must be declared PUBLIC using the statement PUBLIC FACTORIAL and in module 2, it must be declared external using the declaration EXTRN FACTORIAL. The statement of declaration EXTRN must be accompained by the SEGMENT and ENDS directives of the MODULE 1, before it is called in MOBULE 2. Thus the MODULE1 and MODULE 2 must have the following declarations.

```
MODULE1     SEGMENT
PUBLIC      FACTORIAL FAR
MODULE1     ENDS
MODULE2     SEGMENT
EXTRN       FACTORIAL FAR
MODULE2     ENDS
```

GROUP: Group the Related Segments This directive is used to form logical groups of segments with similar purpose or type. This directive is used to inform the assembler to form a logical group of the following segment names. The assembler passes an information to the linker/loader to form the code such that the group declared segments or operands must lie within a 64Kbyte memory segment. Thus all such segments and labels can be addressed using the same segment base.

```
PROGRAM GROUP CODE, DATA, STACK
```

The above statement directs the loader/linker to prepare an EXE file such that CODE, DATA and STACK segment must lie within a 64kbyte memory segment that is named as PROGRAM. Now, for the ASSUME statement, one can use the label PROGRAM rather than CODE, DATA and STACK as shown.

```
ASSUME CS: PROGRAM, DS: PROGRAM, SS: PROGRAM
```

LABEL: Label The Label directive is used to assign a name to the current content of the location counter. When the assembly process starts, the assembler initialises a location counter to keep track of memory locations assigned to the program. As the program assembly proceeds, the contents of the location counter are updated. During the assembly process, whenever the assembler comes across the LABEL directive, it assigns the declared label with the current contents of the location counter. The type of the label must be specified, i.e. whether it is a NEAR or a FAR label, BYTE or WORD label, etc.

A LABEL directive may be used to make a FAR jump as shown below. A FAR jump cannot be made at a normal label with a colon. The label CONTINUE can be used for a FAR jump, if the program contains the following statement.

```
CONTINUE  LABEL FAR
```

The LABEL directive can be used to refer to the data segment along with the data type, byte or word as shown.

```
DATA          SEGMENT
DATAS DB 50H DUP (?)
DATA-LAST LABEL BYTE FAR
DATA ENDS
```

After reserving 50H locations for DATAS, the next location will be assigned a label DATA-LAST and its type will be byte and far.

LENGTH: Byte Length of a Label This directive is not available in MASM. This is used to refer to the length of a data array or a string.

```
MOV CX, LENGTH ARRAY
```

This statement, when assembled, will substitute the length of the array ARRAY in bytes, in the instruction.

LOCAL The lables, variables, constants or procedures declared LOCAL in a module are to be used only by that particular module. After some time, some other module may declare a particular data type LOCAL, which was previously declared LOCAL by an other module or modules. Thus the same label may serve different purposes for different modules of a program. With a single declaration statement, a number of variables can be declared local, as shown.

```
LOCAL a, b, DATA, ARRAY, ROUTINE
```

NAME: Logical Name of a Module The NAME directive is used to assign a name to an assembly language program module. The module, may now be referred to by its declared name. The names, if selected to be suggestive, may point out the functions of the different modules and hence may help in the documentation.

OFFSET: Offset of a Label When the assembler comes across the OFFSET operator along with a label, it first computes the 16-bit displacement (also called as offset interchangebly) of the particular label, and replaces the string 'OFFSET LABEL' by the computed displacement. This operator is used with arrays, strings, lables and procedures to decide their offsets in their default segments. The segment may also be decided by another operator of similar type, viz, SEG. Its most common use is in the case of the indirect, indexed, based indexed or other addressing techniques of similar types, used to refer to the memory indirectly. The examples of this operator are as follows:

Example 2.50

```
CODE SEGMENT
MOV SI, OFFSET LIST
CODE ENDS
DATA SEGMENT
LIST DB 10H
DATA ENDS
```

ORG : Origin The ORG directive directs the assembler to start the memory allotment for the particular segment, block or code from the declared address in the ORG statement. While starting the assembly process for a module, the assembler initialises a location counter to keep track of the allotted addresses for the module. If the ORG statement is not written in the program, the location counter is initialised to 0000. If an ORG 200H statement is present at the starting of the code segment of that module, then the code will start from 200H address in code segment. In other words, the location counter will get initialised to the address 0200H instead of 0000H. Thus, the code for different modules and segments can be located in the available memory as required by the programmer. The ORG directive can even be used with data segments similarly.

PROC: Procedure The PROC directive marks the start of a named procedure in the statement. Also, the types NEAR or FAR specify the type of the procedure, i.e whether it is to be called by the main program located within 64K of physical memory or not. For example, the statement RESULT PROC NEAR marks the start of a routine RESULT, which is to be called by a program located in the same segment of memory. The FAR directive is used for the procedures to be called by the programs located in different segments of memory. The example statements are as follows:

Example 2.51

```
RESULT    PROC  NEAR
ROUTINE   PROC  FAR
```

PTR: Pointer The POINTER operator is used to declare the type of a label, variable or memory operand. The operator PTR is prefixed by either BYTE or WORD. If the prefix is BYTE, then the particular label, variable or memory operand is treated as an 8-bit quantity, while if WORD is the prefix, then it is treated as a 16-bit quantity. In other words, the PTR operator is used to specify the data type—byte or word. The examples of the PTR operator are as follows:

Example 2.52

```
MOV AL, BYTE PTR [SI] -     Moves content of memory location
                            addressed by SI (8-bit) to AL
INC BYTE PTR [BX]-          Increments byte contents of memory
                            location addressed by BX
MOV BX, WORD PTR [2000H]-   Moves 16-bit content of memory
                            location 2000H to BX, i.e. [2000H] to BL
                            [2001H] to BH
INC WORD PTR [3000H] -      Increments word contents of memory
                            location 3000H considering contents
                            of 3000H (lower byte) and 3001H
                            (higher byte) as a 16-bit number
```

In case of JMP instructions, the PTR operator is used to specify the type of the jump, i.e. near or far, as explained in the examples given below.

```
JMP NEAR PTR [BX]-NEAR Jump
JMP FAR PTR [BX]-FAR Jump.
```

PUBLIC As already discussed, the PUBLIC directive is used along with the EXTRN directive. This informs the assembler that the labels, variables, constants, or procedures declared PUBLIC may be accessed by other assembly modules to form their codes, but while using the PUBLIC declared lables, variables, constants or procedures the user must declare them externals using the EXTRN directive. On the other hand, the data types declared EXTRN in a module of the program, may be declared PUBLIC in at least any one of the other modules of the same program. (Refer to the explanation on EXTRN directive to get the clear idea of PUBLIC.)

SEG: Segment of a Label The SEG operator is used to decide the segment address of the label, variable, or procedure and substitutes the segment base address in place of "SEG" label. The example given below explains the use of SEG operator.

Example 2.53

```
MOV AX, SEG ARRAY  ; This statement moves the segment address of ARRAY in
MOV DS, AX         ; which it is appearing, to register AX and then to DS.
```

SEGMENT: Logical Segment The SEGMENT directive marks the starting of a logical segment. The started segment is also assigned a name, i.e. label, by this statement. The SEGMENT and ENDS directive must bracket each logical segment of a program. In some cases, the segment may be assigned a type like PUBLIC (i.e. can be used by other modules of the program while linking) or GLOBAL (can be accessed by any other modules). The program structure given below explains the use of the SEGMENT directive.

```
EXE.CODE SEGMENT GLOBAL; Start of Segment named EXE.CODE,
                       ; that can be accessed by any other module.
EXE.CODE ENDS          ; END of EXE.CODE logical segment.
```

SHORT The SHORT operator indicates to the assembler that only one byte is required to code the displacement for a jump (i.e. displacement is within –128 to +127 bytes from the address of the byte next to the jump opcode). This method of specifying the jump address saves the memory. Otherwise, the assembler may reserve two bytes for the displacement. The syntax of the statement is as given below.

```
JMP SHORT LABEL
```

TYPE The TYPE operator directs the assembler to decide the data type of the specified label and replaces the 'TYPE' label by the decided data type. For the word type variable, the data type is 2, for double word type, it is 4, and for byte type, it is 1. Suppose, the STRING is a word array. The instruction MOV AX, TYPE STRING moves the value 0002H in AX.

GLOBAL The labels, variables, constants or procedures declared GLOBAL may be used by other modules of the program. Once a variable is declared GLOBAL, it can be used by any module in the program. The following statement declares the procedure ROUTINE as a global label.

```
ROUTINE PROC      GLOBAL
```

'+ & –' Operators These operators represent arithmetic addition and subtraction respectively and are typically used to add or subtract displacements (8 or 16 bit) to base or index registers or stack or base pointers as given in the example:

Example 2.54

```
MOV AL, [ SI +2 ]
MOV DX, [ BX – 5 ]
MOV BX, [ OFFSET LABEL + 10 H ]
MOV AX, [ BX + 9I ]
```

FAR PTR This directive indicates the assembler that the label following FAR PTR is not available within the same segment and the address of the label is of 32-bits i.e. 2 bytes offset followed by 2 bytes segment address.

Example 2.55

```
JMP FAR PTR LABEL
CALL FAR PTR ROUTINE
```

Both the above instructions indicate to the assembles that the target address is going to require four bytes; Lower byte of offset, higher byte of offset, lower byte of segment and higher byte of segment; indicating intersegment addressing mode.

NEAR PTR This directive indicates that the label following NEAR PTR is in the same segment and needs only 16 bit i.e. 2 byte offset to address it.

Example 2.56

```
JMP NEAR PTR LABEL
CALL NEAR PTR ROUTINE
```

If a label is not preceded by NEAR PTR or FAR PTR, then it is by default considered a NEAR PTR label and two bytes are reserved by the assembler for its address during the process of assembling.