8086 Assembler Directives

→ An assembler is a program that converts an assembly language program into an equivalent machine language program.

→ The assembler finds the address of each label and substitutes the value of each constant and variable in the assembly language program during the assembly process, to generate the machine language code.

→ For completing the above task, the assembler needs some command from the programmer —

① Storage class of a particular constant or a variable such as byte, word, double word.

② logical name of the segments such as CODE, STACK or DATA

③ type of procedure or routines such as FAR, NEAR, PUBLIC, EXTRN, end of segment etc.

Such type of commands given to the assembler to generate the machine code for assembly language program are defined as assembler directives.

(1) Assembler directives for variable and constant definitions:

DB : define byte : reserves one byte
DW : define word : reserves one word
DD : define double word : reserves 2 words
DQ : define quad word : reserves 4 words
DT : define ten bytes : reserves 10 bytes in memory

Ex:-

(1) DATA1 DB 20H ; reserve one byte for storing DATA1 and assigns the value 20H to it

(2) ARRAY1 DB 10H, 20H, 30H ; Reserve three bytes for ARRAY1 and initialize with 10H, 20H, 30H.

(3) CITY DB "MADURAI" ; store the ASCII code of characters specified within double quotes in the array or list name as CITY.

(4) DATA2 DW 1020H ; Reserve 1 word for story DATA2 and assign 1020H to it.

(5) ARRAY2 DW 1020H, 1030H, 1040H, 1050H ; Reserve 4 word for story ARRAY2 and initialize with values 1020H, 1030H, 1040H, 1050H.

(6) DATA3 DD 1234ABCDH ; Initialize DATA3 as double word 1234ABCDH.

(7) DATA4 DQ 1234ABCD5678EFBBH ; Initialize DATA4 as quad word 1234ABCD5678EFBBH

(8) DATA5 DT 123456789ABCDEF12345H ; Initialize DATA5 as a reserve 10 byte hexa value 123456789ABCDEF12345H

The directive DUP (duplicate) is used to reserve a series of bytes or words, double words or ten bytes and is used with DB, DW, DD, DT respectively.
The reserved area can be either filled with a specific value or left uninitialized.

'x'

(a) Array DB 20 DUP (0) ; Reserves 20 bytes in memory for the array named Array and initializes all elements to 0.

(b ARRAY1 DB 25 DUP (?) ; Reserves 25 bytes in memory for the array named ARRAY1 and keeps all elements of array uninitialized.

(c) ARRAY2 PB 50 DUP (64H) ; Reserves 50 bytes in memory for the array AR2 and keeps all elements initialized to 64H.

EQU : The directive EQU (equivalent) is used to assign a value to a data name.

Ex:

(a) NUMBER EQU 50H ; Assign the value 50H to NUMBER.

(b) NAME EQU "RAMESH" ; Assign the string "RAMESH" to NAME.

Assembler directives related to code (Program) location :

→ ORG :

The ORG directive directs the assembler to start the memory allocation for a particular segment (data, code, stack) from the declared offset address in ORG statement.

When ORG directive is not mentioned it starts at an offset address 0000H.

Ex:-

```
ORG 100H
```

→ location counter (LC) is initialized to 0100H and first instruction is stored from offset address 0100H if it is mentioned at the beginning of code segment.

ORG 100H

If it is placed in the data segment, the next data storage starts from the offset address 0100H within the data segment.

→ EVEN :

The EVEN directive updates the location counter to the next even address, if the current location counter content is not even number.

Ex:-

```
EVEN
ARRAY2  DW  20 DUP (0)
```

→ These statements in a segment declare an array named ARRAY2 having 20 words starting at an even address.

Advantage of storing at an even address is that 8086 needs only one memory read/write cycle to read/write an entire word. otherwise 8086 takes two memory read/write cycles to read/write word.

Ex:-

```
EVEN
RESULT PROC NEAR
   :
RESULT ENDP
```

Here the procedure RESULT, which is of type NEAR, is stored starting at an even address in the code segment.

ENDP directive indicates the end of RESULT procedure.

→ LENGTH:

This directive is used to determine the length of an array or string in bytes.

Ex:-

```
MOV CX, LENGTH ARRAY1
```

CX is loaded with the no. of bytes in the ARRAY1.

→ OFFSET

This operator is used to determine the offset of a data item in a segment containing it.

Ex:- MOV BX, OFFSET TABLE

If the data item named TABLE is present in the data segment, this statement places the offset address of TABLE, in the BX register.

→ LABEL

The LABEL is used to assign a name to the current value in the location counter. It is used to specify the destination of the branch related instructions such as jump and call.

When LABEL is used to specify the destination, it is necessary to specify whether it is NEAR or FAR.

When the destination is in the same segment the label is specified as NEAR.

When the destination is in another segment, it is specified as FAR.

Ex:-

```
REPEAT   LABEL  NEAR
CALCULATE LABEL FAR
```

LABEL can also be used to specify a data item. When it is used to specify a data item, the type of the data item must be specified. The data may have the type - byte or word.

Ex:-

A stack segment having 100 words of data can be defined as follows:

```
STACK SEGMENT
DW  100  DUP (0)    ; Reserve 100 words for
                      stack
STACK_TOP LABEL WORD
STACK ENDS
```

The second statement reserves 100 words in stack segment and fills them with 0.

The third statement assigns the name STACK_TOP to the location present just after the hundredth word.

The offset address of this label can be assigned to the stack pointer in the code segment using the following instructions

```
MOV SP, OFFSET STACK_TOP
```

Assembler directives for Segment Declaration

1) SEGMENT and ENDS :

SEGMENT and ENDS directives indicate the start and end of a segment respectively.

In some cases, the segment may be assigned a type such as PUBLIC (i.e. it can be used by other modules of program while linking) or GLOBAL (i.e. it can be used / accessed by any other module).

→ Large assembly language programs are usually developed as seperate assembly modules.

Each assembly module is individually assembled, tested and debugged.

When all the assembly modules are working correctly, their object code files are linked together to form the complete program.

→ For the modules to link together correctly, any segment, label or variable name referred to in other modules must be declared PUBLIC in the module in which it is defined.

Ex:-

DATA1 SEGMENT WORD PUBLIC

makes segment DATA1 available to other assembly modules.

Ex:- PUBLIC X1, X2

makes two variables X1, X2 available to other assembly modules.

If an instruction in an assembly module refers to a variable or label which is present in another module, the assembler must be told that it is external, using EXTRN directive.

→ GLOBAL directive may also be used in place of PUBLIC or EXTRN directive.

Ex:- GLOBAL MULTIPLIER makes the variable MULTIPLIER ~~so that~~ public so that it can be accessed from other assembly modules.

Ex:-

```
CODE1  SEGMENT
- - -
CODE1  ENDS
```

→ This example indicates the declaration of a code segment CODE1.

2) ASSUME :

→ The assume directive is used to inform the assembler, the name of logical segments to be assumed for different segments used in the program.

Ex:-

```
ASSUME  CS: CODE1 , DS: DATA1
```

→ This statement informs the assembler that segment address where the logical segments CODE1, DATA1 are loaded in memory during execution is to be stored in CS and DS registers respectively.

3) GROUP :

This directive is used to form a logic group of segments with a similar purpose.

Ex:-

```
PROGRAM1  GROUP  CODE1, DATA1, STACK1
```

This statement directs the loader / linker to prepare an exe file such that CODE1, DATA1, and STACK1 lie within the 64 KB memory segment that is named PROGRAM1.

Now, we can define the segment registers as

```
ASSUME  CS: PROGRAM1 , DS: PROGRAM , SS: PROGRAM1
```

4) SEG :

The segment operator is used to decide the segment address of the label or variable or procedure and substitute the segment address in place of SEG label.

Ex:-

```
MOV AX, SEG ARRAY1 ;  load segment address in which ARRAY1 is present in AX.

MOV DS, AX         ;  move content of AX to DS.
```

Assembler Directives for declaring Procedures :

1) PROC :

The PROC directive indicates the start of a named procedure. The NEAR and FAR directive specify the type of procedure.

Ex:-

```
SQUARE_ROOT PROC NEAR
```

→ This statement indicates the beginning of a procedure called SQUARE_ROOT which is to be called by a program located in the same segment.

2) ENDP :

The ENDP directive is used to indicate the end of a procedure.

Ex:-

```
SALARY PROC NEAR
---
---
---
SOLARY ENDP.
```

5) EXTRN and PUBLIC:

The directive EXTRN informs the assembler that the procedures, label/labels, and names declared after this directive have already been defined in some other segments and in the segments where they actually appear, they must be declared PUBLIC using PUBLIC directive

Ex:-

```
MODULE1  SEGMENT
PUBLIC SQUARE_ROOT
SQUARE_ROOT PROC FAR
- - -
- - -
SQUARE_ROOT ENDP
MODULE1 ENDS

MODULE2 SEGMENT
EXTRN SQUARE_ROOT FAR
- - -
- - -
CALL SQUARE_ROOT
- - -
- - -
MODULE2 ENDS.
```

Concept of Segment Override prefix :

→ The segment override prefix, which can be added to almost any instruction in any memory related addressing mode, allows the programmer to deviate from the default segment and offset register mechanism.

Ex:-

MOV AX, [BP] instruction accesses data within the stack segment by default as BP is the offset register for stack segment. However, if the programmer wants to get data from the data segment, using BP as the offset register, he can do so by

MOV AX, DS:[BP] ← This is called segment override prefix.

Instruction	Default Segment	Accessed Segment
(1) MOV BX, ES:[BP]	Stack (SS)	ES
(2) MOV BX, SS:[DI]	DS	SS
(3) MOV CX, ES:[BX]	DS	ES
(4) MOV CX, ES:[SI]	DS	ES
(5) MOV AX, CS:[BX]	DS	CS

Table: Instructions that use segment override prefix.