8086 | ASSEMBLER DIRECTIVES, PSEUDO OPCODES

Assembly language has 2 types of statements:

1. **Executable**: Instructions that are translated into Machine Code by the assembler.

2. **Assembler Directives**:

Statements that direct the assembler to do some special task.
No M/C language code is produced for these statements.
Their main task is to inform the assembler about the start/end of a segment, procedure or program, to reserve appropriate space for data storage etc.
Some of the assembler directives are listed below

1. DB (Define Byte) ; Used to define a Byte type variable.
Eg: SUM DB 0 ; Assembler reserves 1 Byte of memory for the variable
; named SUM and initialize it to 0.

2. DW (Define Word) ; Used to define a Word type variable (2 Bytes).

3. DD (Double Word) ; Used to define a Double Word type variable (4 Bytes).

4. DQ (Quad Word) ; Used to define a Quad Word type variable (8 Bytes).

5. DT (Ten Bytes) ; Used to define 10 Bytes to a variable (10 Bytes).

6. DUP() ; Copies the contents of the bracket followed by this
; keyword into the memory location specified before it.
Eg: LIST DB 10 DUP (0) ; Stores LIST as a series of 10 bytes initialized to Zero.

7. SEGMENT ; Used to indicate the beginning of a segment.

8. ENDS ; Used to indicate the end of a segment.

9. ASSUME ; Associates a logical segment with a processor segment.
Eg: Assume CS:Code ; Makes the segment "Code" the actual Code Segment.

10. PROC ; Used to indicate the beginning of a procedure.

11. ENDP ; Used to indicate the end of a procedure.

12. END ; Used to indicate the end of a program.

13. EQU ; Defines a constant

E.g.:: AREA EQU 25H ; Creates a constant by the name AREA with a value 25H
Do remember, in the class, you have been clearly made to understand the difference between using a variable and using a constant.

14. EVEN / ALIGN ; Ensures that the data will be stored by the assembler in the memory in an aligned form. Aligned data works faster as it can be accessed in One cycle. Misaligned data, though is valid, requires two cycles to be accessed hence works slower.

15. OFFSET ; Can be used to tell the assembler to simply substitute the offset address of any variable.
E.g.:: MOV Si, OFFSET String1; SI gets the offset address of String1

16. Macro ; Used to begin a macro
17. Endm ; Used to end a macro
18. Org ; Used to give the starting address from where the subsequent information will be stored in the memory.
19. Byte Ptr ; Used as a memory pointer to an 8-bit data
20. Word Ptr; Used as a memory pointer for a 16-bit data
21. Near; Used for an intra segment branch like a procedure call. It means the branch location is within the same code segment.
22. Far; Used for an inter segment branch. It means the branch location is in a different code segment.

23. Model Directives

.MODEL SMALL **;** All Data Fits in one 64 KB segment.
All Code fits in one 64 KB Segment

.MODEL MEDIUM **;** All Data Fits in one 64 KB segment.
Code may be greater than 64 KB

.MODEL LARGE **;** Both Data and Code may be greater than 64 KB

Combined Example:

```
Data    SEGMENT
        A DB 25H; creates a byte size variable "A" with value 25H
        B DW 1234H; creates a word size variable "B" with value 1234H
        ALIGN; Makes the following data aligned starting from an even address
        LIST DB 10 DUP (0)     ; Stores LIST as a series of 10 bytes initialized to zero          …
Data    ENDS

Code    SEGMENT
        Assume CS: Code, DS: Data   ; Informs the assembler about the segments
Start:          …
                …
                …

Code    ENDS
END Start
```

	PROCEDURE (FUNCTION)	MACRO
1	A procedure (Subroutine/ Function) is a set of instruction needed repeatedly by the program. It is **stored as a subroutine and invoked from several places by the main program.**	A Macro is similar to a procedure but is not invoked by the main program. Instead, the **Macro code is pasted into the main program wherever the macro name is written in the main program.**
2	A subroutine is **invoked by a CALL** instruction and control returns by a RET instruction.	A Macro is simply accessed by **writing its name**. The entire macro code is pasted at the location by the assembler.
3	**Reduces the size** of the program	**Increases the size** of the program
4	**Executes slower** as time is wasted to push and pop the return address in the stack.	**Executes faster** as return address is not needed to be stored into the stack, hence push and pop is not needed.
5	**Depends on the stack**	**Does not depend on the stack**
6	Both CALL and RET are branch operations. **Pipelining fails on a branch.**	**Does not affect pipelining** as there is no branch at all.

https://www.bharatacharyaeducation.com
Learn...
8085 | 8086 | 80386 | Pentium |
8051 | ARM7 | COA

Order our Books here...

8086 Microprocessor book
Link: https://amzn.to/3qHDpJH

8051 Microcontroller book
Link: https://amzn.to/3aFQkXc

#bharatacharya
#bharatacharyaeducation
#8086 #8051 #8085 #80386 #pentium
#microprocessor #microcontrollers