Operators in 8086

- Operator can be applied in the operand which uses the immediate data/address.
- Being active during assembling and no machine language code is generated.
- Different types of operators are:

1) **Arithmetic**: + , - , * , /
2) **Logical** : AND, OR, XOR, NOT
3) **SHL and SHR**: Shift during assembly
4) **[ ]**: index
5) **HIGH**: returns higher byte of an expression
6) **LOW**: returns lower byte of an expression. E.g. NUM EQU 1374 H

   MOV AL HIGH Num ; ([AL] ← 13)
7) **OFFSET**: returns offset address of a variable
8) **SEG**: returns segment address of a variable
9) **PTR**: used with type specifications

   BYTE, WORD, RWORD, DWORD, QWORD

   E.g. INC BYTE PTR [BX]
10) **Segment override**

    MOV AH, ES: [BX]
11) **LENGTH**: returns the size of the referred variable
12) **SIZE:** returns length times type

    E.g.: BYTE VAR DB?

    WTABLE DW **10** DUP (?)

```
MOV AX, TYPE BYTEVAR       ; AX = 0001H
MOV AX, TYPE WTABLE        ; AX = 0002H
MOV CX, LENGTH WTABLE      ; CX = 000AH
MOV CX, SIZE WTABLE        ; CX = 0014H
```

Coding in Assembly language:

Assembly language programming language has taken its place in between the machine language (low level) and the high level language.

- High level language's one statement may generate many machine instructions.
- Low level language consists of either binary or hexadecimal operation. One symbolic statement generates one machine level instructions.

Advantage of ALP

- They generate small and compact execution module.
- They have more control over hardware.
- They generate executable module and run faster.

Disadvantages of ALP:

- Machine dependent.
- Lengthy code
- Error prone (likely to generate errors).

Assembly language features:

The main features of ALP are program comments, reserved words, identifies, statements and directives which provide the basic rules and framework for the language.

Program comments:

- The use of comments throughout a program can improve its clarity.
- It starts with semicolon (;) and terminates with a new line.
- E.g. ADD AX, BX ; Adds AX & BX

Reserved words:

- Certain names in assembly language are reserved for their own purpose to be used only under special conditions and includes
- Instructions : Such as MOV and ADD (operations to execute)
- Directives: Such as END, SEGMENT (information to assembler)
- Operators: Such as FAR, SIZE
- Predefined symbols: such as @DATA, @ MODEL

Identifiers:

- An identifier (or symbol) is a name that applies to an item in the program that expects to reference.
- Two types of identifiers are Name and Label.
- Name refers to the address of a data item such as NUM1 DB 5, COUNT DB 0
- Label refers to the address of an instruction.
- E. g: MAIN PROC FAR
- L1: ADD BL, 73

Statements:

- ALP consists of a set of statements with two types
- Instructions, e. g. MOV, ADD
- Directives, e. g. define a data item

	Identifiers	operation	operand	comment
Directive:	COUNT	DB	1	; initialize count
Instruction:	L30:	MOV	AX, 0	; assign AX with 0

Directives:

The directives are the number of statements that enables us to control the way in which the source program assembles and lists. These statements called directives act only during the assembly of program and generate no machine-executable code. The different types of directives are:

1) **The page and title listing directives:**

   The page and title directives help to control the format of a listing of an assembled program. This is their only purpose and they have no effect on subsequent execution of the program.

   The page directive defines the maximum number of lines to list as a page and the maximum number of characters as a line.

   PAGE [Length] [Width]
   Default : Page [50][80]

   TITLE gives title and place the title on second line of each page of the program. TITLE text [comment]

2) **SEGMENT directive**

   It gives the start of a segment for stack, data and code.

   Seg-name Segment *align+*combine+*‘class’+
   Seg-name ENDS

- Segment name must be present, must be unique and must follow assembly language naming conventions.
- An ENDS statement indicates the end of the segment.
- Align option indicates the boundary on which the segment is to begin; PARA is used to align the segment on paragraph boundary.
- Combine option indicates whether to combine the segment with other segments when they are linked after assembly. STACK, COMMON, PUBLIC, etc are combine types.
- Class option is used to group related segments when linking. The class code for code segment, stack for stack segment and data for data segment.

3) **PROC Directives**

   The code segment contains the executable code for a program, which consists of one or more procedures, defined initially with the PROC directives and ended with the ENDP directive.

   PROC - name PROC [FAR/NEAR]

…………….
…………….
…………….
PROC - name ENDP

- FAR is used for the first executing procedure and rest procedures call will be NEAR.
- Procedure should be within segment.

4) END Directive

- An END directive ends the entire program and appears as the last statement.
- ENDS directive ends a segment and ENDP directive ends a procedure. END PROC-Name

5) ASSUME Directive

- An .EXE program uses the SS register to address the stack, DS to address the data segment and CS to address the code segment.
- Used in conventional full segment directives only.
- Assume directive is used to tell the assembler the purpose of each segment in the program.
- Assume SS: Stack name, DS: Data Segname CS: codesegname

6) Processor directive

- Most assemblers assume that the source program is to run on a basic 8086 level computer.
- Processor directive is used to notify the assembler that the instructions or features introduced by the other processors are used in the program.
  E.g. **.386** - program for 386 protected mode.

7) Dn Directive (Defining data types)

Assembly language has directives to define data syntax [name] Dn expression
The Dn directive can be any one of the following:

DB	Define byte	1 byte
DW	Define word	2 bytes
DD	Define double	4 bytes
DF	defined farword	6 bytes
DQ	Define quadword	8 bytes
DT	Define 10 bytes	10 bytes

VAL1 DB **25**
ARR DB 21, 23, 27, 53
MOV AL, ARR [2] or
MOV AL, ARR + 2 ; Moves 27 to AL register

8) The EQU directive

- It can be used to assign a name to constants.
- E.g. **FACTOR** EQU **12**

- MOV BX, FACTOR ; MOV BX, 12
- It is short form of equivalent.
- Do not generate any data storage; instead the assembler uses the defined value to substitute in.

9) DUP Directive

- It can be used to initialize several locations to zero. e. g. SUM DW 4 DUP(0)
- Reserves four words starting at the offset sum in DS and initializes them to Zero.
- Also used to reserve several locations that need not be initialized. In this case (?) is used with DUP directives.

  E. g. **PRICE** DB 100 DUP(?)
- Reserves 100 bytes of uninitialized data space to an offset PRICE.

<u>Program written in Conventional full segment directive</u>

```
Page 60,132
TITLE SUM program to add two numbers
;---------------------------------------------------
STACK SEGMENT PARA STACK 'Stack'
DW 32 DUP(0)
STACK ENDS ;---------------------------------
------------------
DATA SEG SEGMENT PARA 'Data'
NUM1 DW 3291
NUM 2 DW 582
SUM DW? DATA
SEG ENDS
;-----------------------------------------------------
CODE SEG SEGMENT PARA 'Code'
MAIN PROC FAR
    ASSUME SS: STACK, DS:DATASEG, CS:CODESEG
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, NUM1
    ADD AX, NUM2
    MOV AX, 4C00H
    INT 21H
MAIN ENDP
CODESEG ENDS
END MAIN
```

Description for conventional program:

- STACK contains one entry, DW (define word), that defines 32 words initialized to zero, an adequate size for small programs.
- DATASEG defines 3 words NUM1, NUM2 initialized with 3291 and 582 and sum uninitialized.
- CODESEG contains the executable instructions for the program, PROC and ASSUME generate no executable code.
- The ASSUME directive tells the assembler to perform these tasks.
- Assign STACK to SS register so that the processor uses the address in SS for addressing STACK.
- Assign DATASEG to DS register so that the processor uses the address in DS for addressing DATASEG.
- Assign CODESEG to the CS register so that the processor uses the address in CS for addressing CODESEG.

  When the loading a program for disk into memory for execution, the program loader sets the correct segment addresses in SS and CS.

Program written using simplified segment directives:

```
        .Model memory
    model Memory model can be
    TINY, SMALL, MEDIUM, COMPACT, LARGE, HUGE or
    FLAT TINY for .com program
    FLAT for program up to 4 GB
```

- **Assume** is automatically generated

```
        .STACK [size in bytes]
        Creates stack segment
        .DATA: start of data segment
        .CODE: start of code segment
```

- DS register can be initialized as

```
MOV AX, @DATA
MOV DS, AX
```

<u>ALP written in simplified segment directives:</u>

```
Page 60, 132
TITLE Sum program to add two numbers.
.MODEL SMALL
.STACK 64
.DATA
    NUM1 DW 3241
    NUM 2 DW 572
```

```
     SUM DW  ?
.CODE
MAIN PROC FAR
     MOV AX, @ DATA          ; set address of data segment in DS
     MOV DS, AX
     MOV AX, NUM1
     ADD AX, NUM2
     MOV SUM, AX
     MOV AX, 4C00H           ; End processing
     INT 21H
MAIN ENDP            ; End of procedure
END MAIN                     ; End of program
```

<u>DOS Debug(TASM)</u>

1) Save the code text in **.ASM** format and save it to the same folder where masm and link files are stored.
2) Open dos mode and reach within that folder.
3) \> tasm filename.asm ⇨ makes.obj
4) \> tlink filename ⇨ makes .exe
5) \> filename.exe ⇨ run the code
6) \> td filename.exe ⇨ debug the code [use F7 and F8]