Assembler Directives

Directives Expansion

- ASSUME
- DB - Defined Byte.
- DD - Defined Double Word
- DQ - Defined Quad Word
- DT - Define Ten Bytes
- DW - Define Word
- PROC - Procedure
- PTR - Pointer
- END - End Program
- ENDP - End Procedure
- ENDS - End Segment
- EQU - Equate
- EVEN - Align on Even Memory Address
- GROUP - Group Related Segments
- ORG - Originate

- **ASSUME Directive** - The ASSUME directive is used to tell the assembler that the name of the logical segment should be used for a specified segment.
- **DB(define byte)** - DB directive is used to declare a byte type variable or to store a byte in memory location.
- **DW(define word)** - The DW directive is used to define a variable of type word or to reserve storage location of type word in memory.

- **DD(define double word**) :This directive is used to declare a variable of type double word or restore memory locations which can be accessed as type double word.
- **DQ (define quadword) :**This directive is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage in memory .
- **DT (define ten bytes):**It is used to inform the assembler to define a variable which is **10** bytes in length or to reserve 10 bytes of storage in memory.

- **END**- End program .This directive indicates the assembler that this is the end of the program module. The assembler ignores any statements after an END directive.
- **ENDP**-   End procedure: It indicates the end of the procedure (subroutine) to the assembler.
- **ENDS**-End Segment: This directive is used with the name of the segment to indicate the end of that logical segment.
- **EQU** - This EQU directive is used to give a name to some value or to a symbol.

- **PROC** - The PROC directive is used to identify the start of a procedure.
- **PTR** -This PTR operator is used to assign a specific type of a variable or to a label.
- **ORG -Originate :** The ORG statement changes the starting offset address of the data.

Directives examples

- ASSUME CS:CODE   cs=> code segment
- **ORG** 3000
- **NAME DB ‘THOMAS’**
- **POINTER DD 12341234$_H$**
- **FACTOR EQU 03$_H$**

Assembly Language Programming(ALP) 8086

Program 1: Increment an 8-bit number

- MOV AL, 05H — Move 8-bit data to AL.
- INC AL — Increment AL.

After Execution AL = **06H**

Program 2: Increment an 16-bit number

- MOV AX, 0005H — Move 16-bit data to AX.
- INC AX — Increment AX.

After Execution AX = **0006H**

Program 3: Decrement **an 8-bit number**

- MOV AL, 05_H Move 8-bit data to AL.
- DEC AL Decrement AL.

After Execution AL = **04**$_H$

Program 4: Decrement **an 16-bit number**

- MOV AX, 0005_H Move 16-bit data to AX.
- DEC AX Decrement AX.

After Execution AX = **0004**$_H$

Program 5: 1's complement of an 8-bit number.

- MOV AL, 05H — Move 8-bit data to AL.
- NOT AL — Complement AL.

After Execution AL = **FAH**

Program 6: 1's complement of a 16-bit number.

- MOV AX, 0005H — Move 16-bit data to AX.
- NOT AX — Complement AX.

After Execution AX = **FFFAH**

Program 7: 2's complement of an 8-bit number.

- MOV AL, 05H — Move 8-bit data to AL.
- NOT AL — Complement AL.
- INC AL — Increment AL

After Execution AX = **FAH + 1 = FB**

Program 8: 2's complement of a 16-bit number.

- MOV AX, 0005H — Move 16-bit data to AX.
- NOT AX — Complement AX.
- INC AX — Increment AX

After Execution AX = **FFFAH + 1 = FFFB**

Program 9: Add two 8-bit numbers

MOV AL, 05H	Move 1st 8-bit number to AL.
MOV BL, 03H	Move 2nd 8-bit number to BL.
ADD AL, BL	Add BL with AL.

After Execution AL = **08H**

Program 10: Add two 16-bit numbers

MOV AX, 0005H	Move 1st 16-bit number to AX.
MOV BX, 0003H	Move 2nd 16-bit number to BX.
ADD AX, BX	Add BX with AX.

After Execution AX = **0008H**

Program 11: subtract two 8-bit numbers

MOV AL, 05H	Move 1st 8-bit number to AL.
MOV BL, 03H	Move 2nd 8-bit number to BL.
SUB AL, BL	**subtract** BL from AL.

After Execution AL = **02H**

Program 12: subtract two 16-bit numbers

MOV AX, 0005H	Move 1st 16-bit number to AX.
MOV BX, 0003H	Move 2nd 16-bit number to BX.
SUB AX, BX	**subtract** BX from AX.

After Execution AX = **0002H**

Program 13: Multiply two 8-bit unsigned numbers.

MOV AL, 04H	Move 1st 8-bit number to AL.
MOV BL, 02H	Move 2nd 8-bit number to BL.
MUL BL	Multiply BL with AL and the result will be in AX.

Program 14: Multiply two 8-bit signed numbers.

MOV AL, 04H	Move 1st 8-bit number to AL.
MOV BL, 02H	Move 2nd 8-bit number to BL.
IMUL BL	Multiply BL with AL and the result will be in AX.

Program 15: Multiply two 16-bit unsigned numbers.

MOV AX, 0004H	Move 1st 16-bit number to AL.
MOV BX, 0002H	Move 2nd 16-bit number to BL.
MUL BX	Multiply BX with AX and the result will be in **DX:AX** **{4*2=0008=> 08=> AX , 00=> DX}**

Program 16: Divide two 16-bit unsigned numbers.

MOV AX, 0004H	Move 1st 16-bit number to AL.
MOV BX, 0002H	Move 2nd 16-bit number to BL.
DIV BX	Divide BX from AX and the result will be in **AX & DX**

{4/2=0002=> 02=> AX ,00=>DX}

(ie: Quotient => AX , Reminder => DX)

Detailed coding
16 BIT ADDITION

PROGRAM	COMMENTS
MOV CX, 0000H	Initialize counter CX
MOV AX,[1200]	Get the first data in AX reg
MOV BX, [1202]	Get the second data in BX reg
ADD AX,BX	Add the contents of both the regs AX & BX
JNC L1	Check for carry
INC CX	If carry exists, increment the CX
L1 : MOV [1206],CX	Store the carry
MOV [1204], AX	Store the sum
HLT	Stop the program

Detailed coding
16 BIT SUBTRACTION

PROGRAM	COMMENTS
MOV CX, 0000H	Initialize counter CX
MOV AX,[1200]	Get the first data in AX reg
MOV BX, [1202]	Get the second data in BX reg
SUB AX,BX	Subtract the contents of BX from AX
JNC L1	Check for borrow
INC CX	If borrow exists, increment the CX
L1 : MOV [1206],CX	Store the borrow
MOV [1204], AX	Store the difference
HLT	Stop the program

16 BIT MULTIPLICATION

PROGRAM	COMMENTS
MOV AX,[1200]	Get the first data
MOV BX, [1202]	Get the second data
MUL BX	Multiply both
MOV [1206],AX	Store the lower order product
MOV AX,DX	Copy the higher order product to AX
MOV [1208],AX	Store the higher order product
HLT	Stop the program

16 BIT DIVISION

PROGRAM	COMMENTS
MOV AX,[1200]	Get the first data
MOV DX, [1202]	Get the second data
MOV BX, [1204]	Divide the dividend by divisor
DIV BX	Store the lower order product
MOV [1206],AX	Copy the higher order product to AX
MOV AX,DX	Store the higher order product
MOV [1208],AX	Stop the program
HLT	Get the first data

SUM of N numbers

```
      MOV AX,0000
      MOV SI,1100
      MOV DI,1200
      MOV CX,0005       5 NUMBERS TO BE TAKEN SUM
      MOV DX,0000
L1:   ADD AX,[SI]
      INC SI
      INC DX
      CMP CX,DX
      JNZ L1
      MOV [1200],AX
      HLT
```

Average of N numbers

```
        MOV AX,0000
        MOV SI,1100
        MOV DI,1200
        MOV CX,0005       5 NUMBERS TO BE TAKEN AVERAGE
        MOV DX,0000
L1:     ADD AX,[SI]
        INC SI
        INC DX
        CMP CX,DX
        JNZ L1
        DIV CX            AX=AX/5(AVERAGE OF 5 NUMBERS)
        MOV [1200],AX
        HLT
```

FACTORIAL of N

5 Factorial=5*4*3*2*1=120

```
        MOV CX,0005
        MOV DX,0000
        MOV AX,0001
L1:     MUL CX
        DEC DX
        CMP CX,DX
        JNZ L1
        MOV [1200],AX
        HLT
```

ASCENDING ORDER

SORTING IN ASCENDING ORDER:

- Load the array count in two registers C_1 and C_2.
- Get the first two numbers.
- Compare the numbers and exchange if necessary so that the two numbers are in ascending order.
- Decrement C_2.
- Get the third number from the array and repeat the process until C_2 is 0.
- Decrement C_1 and repeat the process until C_1 is 0.