ASSEMBLER DIRECTIVES

- Assembler directives are the commands to the assembler that direct the assembly process.
- They indicate how an operand is treated by the assembler and how assembler handles the program.
- They also direct the assembler how program and data should arrange in the memory.
- ALP's are composed of two type of statements.

  (i) The instructions which are translated to machine codes by assembler.

  (ii) The directives that direct the assembler during assembly process, for which no machine code is generated.

1. ASSUME: Assume logical segment name.

The ASSUME directive is used to inform the assembler the names of the logical segments to be assumed for different segments used in the program .In the ALP each segment is given name.

Syntax: ASSUME segreg:segname,...segreg:segname

Ex: ASSUME CS:CODE

ASSUME CS:CODE,DS:DATA,SS:STACK

2. **DB:** Define Byte

The DB directive is used to reserve byte or bytes of memory locations in the available memory.

Syntax: Name of variable DB initialization value.

Ex: MARKS DB 35H,30H,35H,40H

NAME DB "VARDHAMAN"

3. **DW:** Define Word

The DW directive serves the same puposes as the DB directive,but it now makes the assembler reserve the number of memory words(16-bit) instead of bytes.

Syntax: variable name DW initialization values.

Ex: WORDS DW 1234H,4567H,2367H

WDATA DW 5 Dup(522h)

(or) Dup(?)

4. **DD:** Define Double:

The directive DD is used to define a double word (4bytes) variable.

Syntax: variablename DD 12345678H

Ex: Data1 DD 12345678H

5. **DQ:** Define Quad Word

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

Syntax: Name of variable DQ initialize values.

Ex: Data1 DQ 123456789ABCDEF2H

6. **DT:** Define Ten Bytes

The DT directive directs the assembler to define the specified variable requiring 10 bytes for its storage and initialize the 10-bytes with the specified values.

Syntax: Name of variable DT initialize values.

Ex: Data1 DT 123456789ABCDEF34567H

7. **END:** End of Program

The END directive marks the end of an ALP. The statement after the directive END will be ignored by the assembler.

8. **ENDP:** End of Procedure

The ENDP directive is used to indicate the end of procedure. In the AL programming the subroutines are called procedures.

Ex: Procedure Start

:

Start ENDP

9. ENDS: End of segment

The ENDS directive is used to indicate the end of segment.

Ex: DATA SEGMENT

:

DATA ENDS

10.EVEN: Align on Even memory address

The EVEN directives updates the location counter to the next even address.

Ex: EVEN

Procedure Start

:

Start ENDP

- The above structure shows a procedure START that is to be aligned at an even address.

11.EQU: Equate

The directive EQU is used to assign a label with a value or symbol.

Ex: LABEL EQU 0500H

ADDITION EQU ADD

12.EXTRN: External and public

- The directive EXTRN informs the assembler that the names, procedures and labels declared after this directive have been already defined in some other AL modules.
- While in other module, where names, procedures and labels actually appear, they must be declared public using the PUBLIC directive.

Ex: MODULE1 SEGMENT

PUBLIC FACT FAR

MODULE1 ENDS

MODULE2 SEGMENT

EXTRN FACT FAR

MODULE2 END

13.GROUP: Group the related segments

This directive is used to form logical groups of segments with similar purpose or type.

Ex: PROGRAM GROUP CODE, DATA, STACK

*CODE, DATA and STACK segments lie within a 64KB memory segment that is named as PROGRAM.

14.LABEL: label

The label is used to assign name to the current content of the location counter.

Ex: CONTINUE LABEL FAR

The label CONTINUE can be used for a FAR jump, if the program contains the above statement.

15.LENGTH: Byte length of a label

This is used to refer to the length of a data array or a string

Ex : MOV CX, LENGTH ARRAY

16.LOCAL: The labels, variables, constant or procedures are declared LOCAL in a module are to be used only by the particular module.

Ex : LOCAL a, b, Data1, Array, Routine

17.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 refer to by its declared name.

Ex : Name "addition"

18.OFFSET: offset of a label

When the assembler comes across the OFFSET operator along with a label, it first computing the 16-bit offset address of a particular label and replace the string 'OFFSET LABEL' by the computed offset address.

Ex : MOV SI, offset list

19.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.

Ex: ORG 1000H

20.PROC: Procedure

The PROC directive marks the start of a named procedure in the statement.

Ex: RESULT PROC NEAR

ROUTINE PROC FAR

21.PTR: pointer

The PTR operator is used to declare the type of a label, variable or memory operator.

Ex : MOV AL, BYTE PTR [SI]

MOV BX, WORD PTR [2000H]

22.SEG: segment of a label

The SEG operator is used to decide the segment address of the label, variable or procedure.

Ex : MOV AX, SEG ARRAY

MOV DS, AX

23.SEGMENT: logical segment

The segment directive marks the starting of a logical segment

Ex: CODE SEGMENT

:

CODE ENDS

24.SHORT: The SHORT operator indicates to the assembler that only one byte is required to code the displacement for jump.

Ex : JMP SHORT LABEL

25.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 word variable, the data type is 2.

For double word variable, the data type is 4.

For byte variable, the data type is 1.

Ex : STRING DW 2345H, 4567H

MOV AX, TYPE STRING

AX=0002H

26.GLOBAL: The labels, variables, constants or procedures declared GLOBAL may be used by other modules of the program.

Ex : ROUTINE PROC GLOBAL.

27.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 of offset followed by 2-bytes of segment address.

Ex : JMP FAR PTR LABEL

28.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

Ex : JMP NEAR PTR LABEL

CALL NEAR PTR ROUTINE

Procedures and Macros:

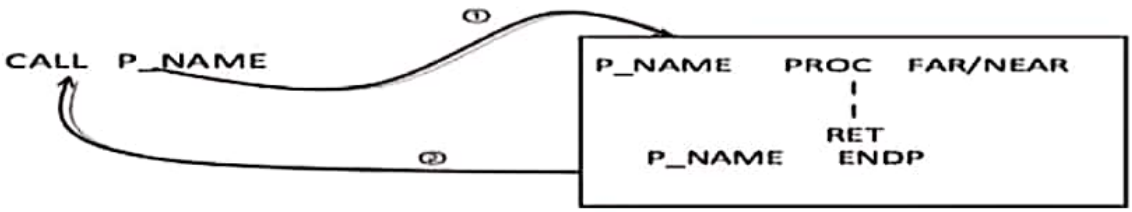
- When we need to use a group of instructions several times throughout a program there are two ways we can avoid having to write the group of instructions each time we want to use them.
  1. One way is to write the group of instructions as a separate **procedure.**
  2. Another way we can use **macros.**

Procedures:

- The procedure is a group of instructions stored as a separate program in the memory and it is called from the main program whenever required using CALL instruction.
- For calling the procedure we have to store the return address (next instruction address followed by CALL) onto the stack.
- At the end of the procedure RET instruction used to return the execution to the next instruction in the main program by retrieving the address from the top of the stack.

- Machine codes for the procedure instructions put only once in memory.
- The procedure can be defined anywhere in the program using assembly directives PROC and ENDP.

Format of procedure in 8086.



① Return address is saved in stack.
Program branches to P_NAME.
② Return address is retrieved from stack.
Program branches to main program.

- **The four major ways of passing parameters to and from a procedure are:**
  1. In registers
  2. In dedicated memory location accessed by name
  3 .With pointers passed in registers
  4. With the stack
- The type of procedure depends on where the procedure is stored in the memory.
- If it is in the same code segment where the main program is stored the it is called near procedure otherwise it is referred to as far procedure.
- For near procedure CALL instruction pushes only the IP register contents on the stack, since CS register contents remains unchanged for main program.
- But for Far procedure CALL instruction pushes both IP and CS on the stack.

Syntax:

Procedure name PROC near

instruction 1

instruction 2

RET

Procedure name ENDP

Example:

near procedure:

```
ADD2 PROC near
ADD AX,BX
RET
ADD2 ENDP
```

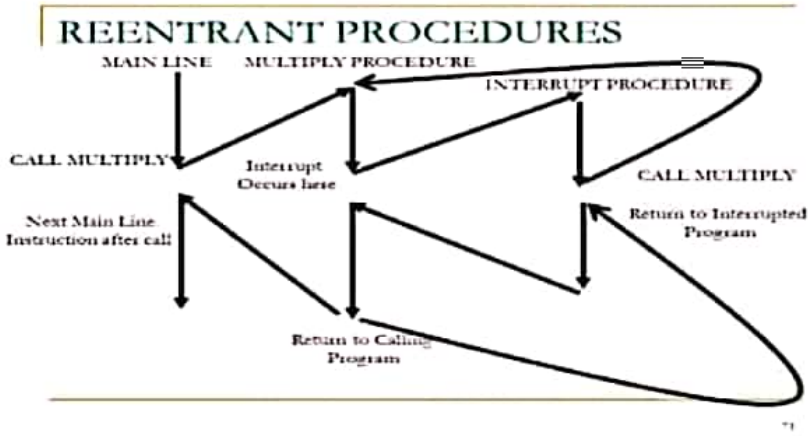
far procedure:

```
Procedures segment
Assume CS : Procedures
ADD2 PROC far
ADD AX,BX
RET ADD2 ENDP
Procedures ends
```

- Depending on the characteristics the procedures are two types
  1. Re-entrant Procedures
  2. Recursive Procedures

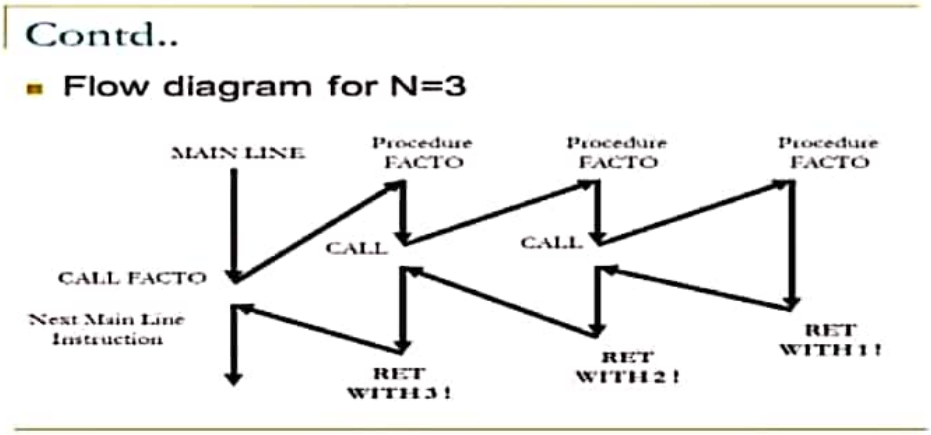
Reentrant Procedures

- The procedure which can be interrupted, used and "reentered" without losing or writing over anything.



Recursive Procedure

- A recursive procedure is procedure which calls itself.



-

ALP for Finding Factorial of number using procedures

```
CODE SEGMENT
ASSUME CS:CODE
START: MOV AX,7
CALL FACT
MOV AH,4CH
INT 21H
FACT PROC NEAR
MOV BX,AX
DEC BX
BACK: MUL BX
DEC BX
JNZ BACK
RET
ENDP
CODE ENDS
END START
```