Gramin Shikshan Prasarak Mandal's

GRAMIN POLYTECHNIC

(ISO 9001:2008 Certified Institute)

Chapter 2- The Art of Assembly language programming

By

Shinde G. B.

M.TECH. (Electronics Engineering)

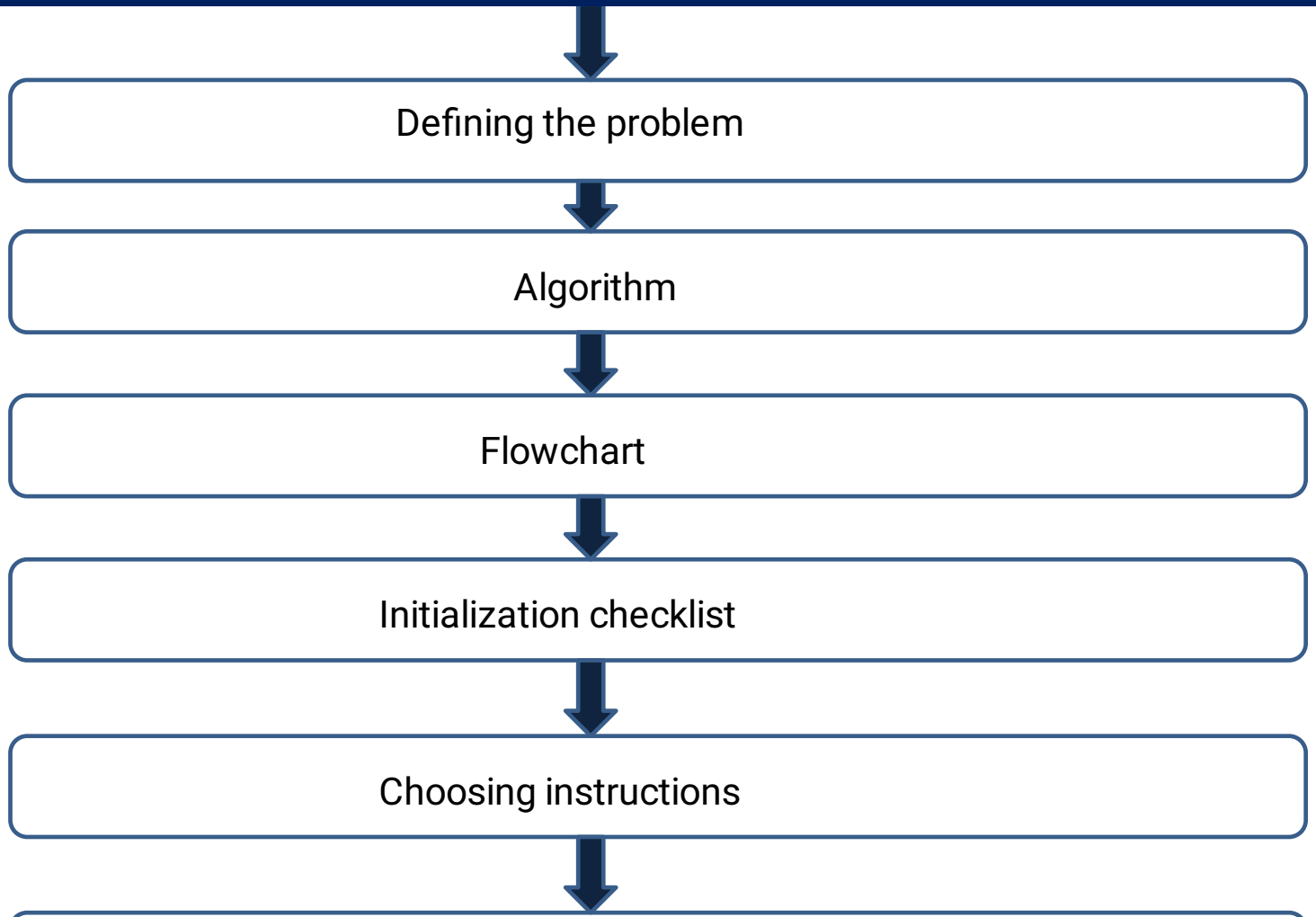
2.1 program development steps: defining problem and constrains, writing algorithms,flowchart,initialization checklist, choosing instructions, converting algorithms to assembly language programs.

2.2 Assembly Language Programming Tools: Editor,Assembler,Linker Debugger.

2.3 Assembler Directives

Unit Outcomes :

- Describes the given steps of program development/execution.
- Write steps to develop a code for the given problem using assembly language programming.
- Use relevant commands of debugger to correct the specified programming error.
- Describe the functions of the given assembler directives with examples.


Defining the problem
Algorithm
Flowchart
Initialization checklist
Choosing instructions

Defining the problem

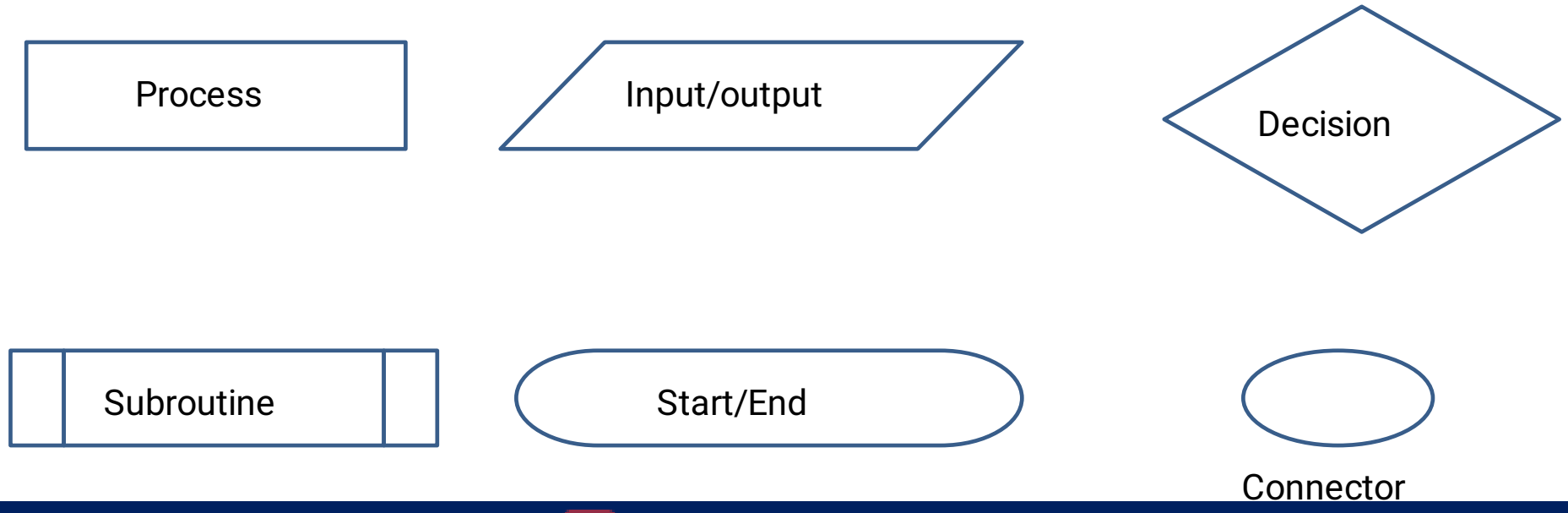
- Define very carefully about the problem that you want the program to solve.
- You need not to write down program but you must know what you would like to do.

Algorithm

- The formula or sequence of operations or task need to perform by your program can be specified as a steps in general English and is often called as algorithm.
- It is step by step method or statement written in general English language of solving problem.

Flowchart

- The flowchart is graphically representation of the program operation or task.



Initialization Checklist

- In program there are many variables, constants and various part of the system such as segment registers ,flag, stack, programmable ports.
- Which must be initialize properly

Choosing Instructions

- Next Step is to choose appropriate instruction that performs your problem's operations and task.
- You must know instruction set of microprocessor.

Converting Algorithms to Assembly Language Program

- Once you have selected the instructions for the operations to be performed then arrange these instructions in sequence as per algorithm.
- So that desired output must be obtaining after execution.

Assembly Language Program Development Tools

- Editor
- Assembler
- Linker
- Debugger

Editors

- An Editor is a program which helps you to construct your assembly language program in right format.
- You can type your program using editor.
- The DOS based editor such as EDIT, WordStar and Norton Editor is used.

Assembler

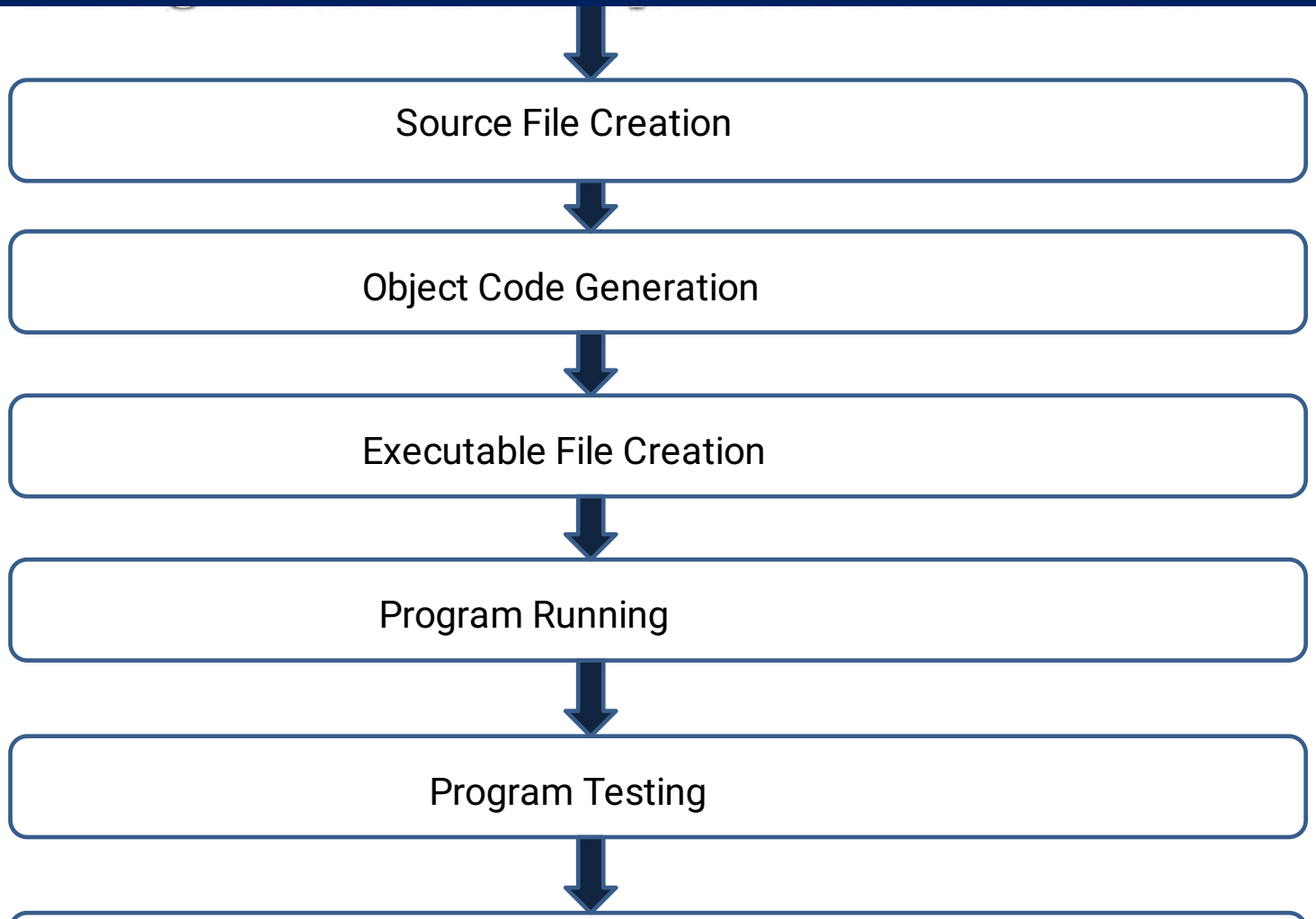
- An assembler is a correct program that translate assembly language program to the binary code for each instruction.
- File is generated called as object file with extension .obj
- We use TASM assembler.

Linker

- A Linker is a program which combines if requested more one assembled module into one executable program.
- File is called as executable file with extension .exe
- We use TLINK Linker.

Debugger

- Debugger is a program that allows the execution of program in single step mode under the control of the user.
- The process of locating and correcting errors using a debugger is known as debugging.
- We use Turbo Debugger(TD).


Source File Creation
Object Code Generation
Executable File Creation
Program Running
Program Testing

Source file creation: The file containing the program statement in assembly language is known as a source file.

The source file is created and edited using the text editor and must have an extention.ASM

Object code generation: the language translator is used to translate source program to re-locatable object file.

The assembler is used to translate assembly language

Executable file creation: linker is used to create the executable file.

Program running: the executable file can be run by entering the name of executable file on prompt and by pressing ENTER key on the keyboard.

Program testing: the results or output generated by the program has to be tested for their validity.

If any errors occurs in the results then the program should be debug.

Program debugging:

The errors in the program can be located using debugger.

Assembler Directives and Operator

- Assembly Language program supports a number of reserve word. i.e. key word
- It is also called as assembler directives.
- A statement to give the direction to the assembler to perform task to the assembly process.
- It control the organization of the program.
- It provides the necessary information to the assembler to understand the ALPs

They indicate the how an operand or section of a program is to be processed by the assembler

Program consist two types of statements.

- Instructions and directives.

Instructions: translated to the machine code by the assembler.

Directives: directives are not translated to the machine codes.

Data Definition and Storage Allocation Directives

- DB(Define Byte)
- DW(Define Word)
- DD(Define Double Word)
- DQ(Define Quad Word)
- DT(Define Ten Byte)
- STRUCT(Structure Declaration)
- RECORD

- EQU(Equal to)
- ORG(Originate)
- ALIGN(Alignment of memory addresses)
- EVEN(Align as even memory location)
- LABEL
- DUP(Duplicate memory Location)

a) DB(Define Byte)

- It is used to define a byte type variable.
- It can be used to single or multiple byte variable.
- Can store 0 to 255 for unsigned no.
- Can store -128 to +127 for signed no.
- Syntax:

Name_of_variable DB Initialization_value

EX:

Num DB ? ; allocate one memory location

num DB 04H ; allocate one memory location.

name DB ‘Gramin’;

b) DW(Define Word)

- It is used to define a word type i.e. 2 bytes variable.
- It can be used to define the single or multiple word variable.
- The DW directive is used to tell the assembler to define a variable of type word or to reserve storage locations of type word in memory.
- Can store 0 to 65535 for unsigned no.
- Can store -32768 to + 32767 for signed no.

Syntax:

Name_of_variable DW Initialization_value

EX:

- NUM DW ? ; allocate two memory locations.
- NUM DW 78 ; allocate two memory locations.
- MULTIPLIER DW 437AH ; declares a variable of type word named MULTIPLIER, and initialized with the value 437AH when the program is loaded into memory to be run.
- WORDS DW 1234H, 3456H; Declares an array of 2 words and initialize them with the specified values.

c) DD(Define Double Word)

- It is used to define a double word type i.e. 4 bytes variable.
- The DD directive is used to declare a variable of type double word or to reserve memory locations, which can be accessed as type double word.

Syntax:

Name_of_variable DD Initialization_value

EX:

- NUM DD ? ;allocate four memory locations.
- Table DD 1,3,5,7,9 ;allocate 20 memory locations.
- ARRAY DD 25629261H ; This will define a double word named ARRAY and initialize the double word with the specified value when the program is loaded into memory to be run. The low word, 9261H, will be put in memory at a lower address than the high word.
- name DD 'Gramin Polytechnic'

d) DQ(Define Quad Word)

- It is used to define a quad word type i.e. 8 bytes variable.
- The DQ directive is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage in memory.

Syntax:

Name_of_variable DQ Initialization_value

EX:

- NUM DQ ? ; allocate eight memory locations.
- Table DQ 1,2,3,5,9 ; allocate fourty memory locations.
- BIG_NUMBER DQ 243598740192A92BH

  This will declare a variable named BIG_NUMBER and initialize the 4 words set aside with the specified number when the program is loaded into memory to be run.
- name DQ ‘Gramin Polytechnic’

c) DT(Define Ten Byte)

- It is used to define a ten byte type variables
- It is useful in math coprocessor instructions.
- The DT directive is used to tell the assembler to declare a variable, which is 10 bytes in length or to reserve 10 bytes of storage in memory.

- Syntax:

Name_of_variable DT Initialization_value

EX:

- PACKED_BCD DT 1122334455667889900 ; This will declare an array named PACKED_BCD, which is 10 bytes in length. It will initialize the 10 bytes with the values 11, 22, 33, 44, 55, 66, 77, 88, 99, and 00 when the program is loaded into memory to be run

•Table DT 1,2,3,4,5,9 ; allocate sixty memory locations.

•NUM DT ? ; allocate ten memory locations.

f) STRUCT(Structure Declaration)

- The directives STRUCT is used to declare the data type which a collection of primary data types(DB,DW,DD)
- Syntax:

Structure_name STRUCT

....

.....

.....

Structure_name ENDS

- Example

Employee STRUCT
Emp_num DW 1234
Emp_name DB 25 DUP(0)
Emp_dept DB 30 DUP(0)
Emp_age DB 20
Employee ENDS

g) RECORD

- The directives RECORD is used to define a bit pattern within a byte or word.
- Syntax:

Record_name RECORD

....

.....

.....

- Example
- Bit 7,6,5 :Baud Rate
- Bit 4,3 :Parity
- Bit 2 :Stop Bits
- Bit 1,0 :Word Length

SERIAL_COM RECORD BAUD:3, PARITY:2,STOP:1 WORDLENGTH:2

h) EQU(Equate to)

- The directives is used to declare the symbols to which some constant value is assigned

Syntax:

Symbol name EQU expression

EX:

FACTOR EQU 03H

ADD AL,FACTOR ; when it codes this instruction the assembler will code it as

ADD AL,03H

1) ORG(Originate)

- The directive ORG assigns the location counter with the value specified in the directive.
- It helps to place machine code to location
- Syntax:

ORG Numeric_value

- EX:

ORG 100H ;tells the assembler to set the

Assembler counter to 100H

j) ALIGN(Alignment of memory address)

- The directive ALIGN is used to force the assembler to align the next data item or instruction according to given value.
- Syntax:

ALIGN Numeric_value

- EX:

ALIGN 2 ; storing from an even address.(the assembler advances the location counter to the next address that is evenly divisible by 2)

2,4,8,16.....

k) EVEN(Align as even memory address)

- The directive EVEN is used to inform the assembler to increment the location counter to the next even memory address.
- If it is location counter is already pointing to even memory address, it should not be incremented.
- Syntax:

EVEN

- EX:

```
DATA SEGMENT
Name db 'Computer$'
EVEN
PRIZE db 3000
DATA ENDS
```

I)LABEL

- The directive LABEL enables you to redefine the attribute of a data variable or instruction label
- Syntax:

Variable_name LABEL type specifier

- EX:

TEMP LABEL BYTE

NUM LABEL WORD

m) DUP(Duplicate memory location)

- The directive DUP used to generate multiple bytes or words with known as un-initialized values.
- Syntax:

Variable_name type specifier DUP(value)

- EX:

LIST DB 100 DUP(0) ;allocate 100 memory locations.

LIST DW 50 DUP(0) ; allocate 100 memory locations.

LIST DD 10 DUP(0) ;

Program Organization Directives

- ASSUME
- SEGMENT
- ENDS(End of segment)
- END(End of program)

- .CODE
- .DATA
- .STACK
- .MODEL

ASSUME

- The directive ASSUME informs the assembler the name of the logical segment that should be used for a specified segment.
- ASSUME tells the assembler what names have been chosen for Code, Data Extra and Stack segments. Informs the assembler that the register CS is to be initialized with the address allotted by the loader to the label CODE and DS is similarly initialized with the address of label DATA.
- Syntax:

  ASSUME Seg_Reg:Seg_Name
- EX:
- ASSUME CS:My_code,DS:My_data
- **ASSUME CS:** Name of code segment
- **ASSUME DS:** Name of the data segment
- **ASSUME CS:** Code1, DS: Data1

SEGMENT

- The directive SEGMENT is used to indicate the beginning of the logical segment.
- Syntax:

  Seg_name SEGMENT [WORD/PUBLIC]
- EX:
- My_data SEGMENT

.....

.....

My_data ENDS

ENDS(End of the segment)

- The directive ENDS informs the assembler the end of the segment
- Syntax:

  Segment Name ENDS
- EX:
- My_data SEGMENT

.....

.....

My_data ENDS

END(End of the program)

- The directive END informs the assembler the end of the program
- Syntax:

  END
- EX:
- My_data SEGMENT

.....

.....

My_data ENDS

END

.CODE

- The simplified segment directive defines the code segment.
- Syntax:

  .CODE
- EX:
- .CODE

.....

.....

ENDS

.DATA

- The simplified segment directive defines the data segment for initialized near data.
- Syntax:

  .DATA
- EX:
- .DATA

.....

.....

ENDS

.STACK

- The simplified segment directive defines the stack segment and default size of stack is 1024 bytes.
- Syntax:

  .STACK
- EX:
- .STACK

.....

.....

ENDS

.MODEL

- The simplified segment directive creates default segments.
- Syntax:

  .MODEL type of memory
- EX:
- .model small

.....

.....

end

Value Returning Attribute Directives

- LENGTH
- SIZE
- OFFEST
- SEG : Segment
- TYPE

LENGTH

- The directive LENGTH informs the assembler about the number of elements in a data item.
- Syntax:

  LENGTH variable_name
- EX:

NAME DW ‘COMPUTER$’

MOV CX,LENGTH NAME

SIZE

- The directive SIZE informs the assembler about the number of bytes allocated to the data item.
- Syntax:

  SIZE variable_name
- EX:

NAME DW ‘COMPUTER$’

MOV CX,LENGTH NAME

OFFSET

- The directive OFFSET informs the assembler to determine the displacement of the specified variable with respect to the base of the segment.
- Syntax:

  OFFSET variable _name
- EX:

MOV DX,OFFSET ARRAY

SEG: Segment

- The directive SEG is used to determine the segment in which the specified data item is defined.
- Syntax:

  SEG variable _name
- EX:

MOV DS,SEG MSG

TYPE

- The directive TYPE is used to determine the type of the data item.(no. of bytes allocated)
- Syntax:

  TYPE variable _name
- EX:

NUM DB 10

ADD BX, TYPE NUM

Procedure Definition Directives

- PROC : Procedure
- ENDP : End of procedure

PROC : Procedure

- The directive PROC indicates beginning of procedure and follows with the name of the procedure.
- The term FAR and NEAR follow the PROC directive indicating the types of the procedure.
- Syntax:

  Procedure_name PROC [NEAR/FAR]
- EX:

ADD PROC NEAR

……..

…….

ADD ENDP

ENDP : End of the Procedure

- The directive ENDP informs the assembler the end of a procedure.
- Syntax:

  Procedure_name ENDP
- EX:

```
ADD PROC NEAR
……..
…….

ADD ENDP
```

Macro Definition Directives

- MACRO : Procedure
- ENDM : End of macro

MACRO

- The directive MACRO informs the assembler the beginning of a macro.
- Syntax:
  Macro_Name MACRO [Argument1,…Argument]
- EX:

```
DISP MACRO MSG
PUSH AX
PUSH DX
MOV AH,09H
LEA DX,MSG
INT 21H
POP DX
POP AX
ENDM
```

```
.DATA
MSG1 DB ‘Welcome to computer department’
MSG2 DB ‘Hardware Lab’
.CODE
…….
DISP MSG1
DISP MSG2
……..
ENDS
END
```

MACRO:

- Small sequence of codes of the same pattern are repeated frequently at different places which perform the same operation on the different data of the same data type.
- Such repeated code can be written separately as a macro.

ENDM : End of the MACRO

- The directive ENDM informs the assembler the end of a macro.
- Syntax:
  ENDM
- EX:

```
DISP MACRO MSG
PUSH AX
PUSH DX
MOV AH,09H
LEA DX,MSG
INT 21H
POP DX
POP AX
ENDM
```

DATA Control Directives

- PUBLIC
- EXTERN : External
- PTR : Pointer

PUBLIC

- The directive PUBLIC informs the assembler that the specified variable or segment can be accessed from other program modules.
- Syntax:

PUBLIC var1,var2…..varN

EX

PUBLIC MSG,NAME,NUM

PUBLIC ARRAY

EXTERN : External

- The directive EXTERN informs the assembler that the data items or label following the directive will be used in a program module which is defined in the other program modules.
- Syntax

Var1:ref_type,……VarN:ref_type

EX:

EXTERN msg:byte, name:word, num:byte

PTR : Pointer

- The directive PTR is used to indicate the type of the memory access i.e. BYTE/ WORD/ DWORD.
- EX:

INC BYTE PTR [DI]

DEC WORD PTR [SI]

File Inclusion Directives

- The file inclusion directive is used to define include file header and directive in INCLUDE.
- The part of assembler which processes the include file is known as pre-processor.
- Syntax:

INCLUDE file path with file name

INCLUDE C:\TASM\MYPROC.asm

Target Machine Code Generation Control Directives

- The special directive can be used at the beginning of a program to inform to the assembler to generate code for the specific processor.
- Syntax:

.186

.286

.386

.486

.586

Thank You!!!