Unit II : The Art of Assembly Language Programming.

(Weightage - 08 marks)

2:1 Program development steps: Defining problem and constraints, writing algorithm, Flowchart, Initialization checklist, Choosing instructions, Converting algorithms to assembly language programs.

Questions .....

Q// Describe how an assembly language program is developed. (6M).

Q// Explain assembly language program development steps. (4M)

Q// Demonstrate indetail the program development steps in Assembly language programming

Program Development Steps

* The programming in Assembly language or any otherlanguage follows specific sequences of step which is called as program development steps.
* It is also called as PDLC (Program development life cycle) also Software Development life Cycle.

Steps :-

- Defining the problem
- Algorithm
- Flowchart
- Initialization checklist
- Choosing instructions
- Convert Algorithm to ALP.

DAF ICC

1) Defining the ~~program~~ problem.

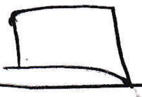
* The First step in writing ~~problem~~ ~~program~~ problem is to think carefully about the problem to solve.
* Write a problem statement use modular approach to define sub-problems.
* At this point you need not write down program but you must know what you would like to do.
* It is important part of stage.

② Algorithm

* Algorithm is step-by-step instruction to solve the problem.
* The formula or sequence of operations or task need to performed by your program can be specified as a step in general English is called algorithm.
* Define algorithmic steps in English statements. i.e. pseudocode is also a way.

③ Flowchart

* It is a graphical representation of program operation or task.
* It is diagramatic representation of specific program logic.
* Specific operation or task is represented by graphical symbol such as circle, rectangle, diagonal and parallelogram etc...

Symbol	Name	Function
	Start/ termination symbol	Start and End points.
	Arrow	Connects.
	Input/ Output	Shows I/O of program.
	process	Shows process
	Descision	decision-making if-else, if, etc...

④ Initialization Checklist :-

* Inialize the variables, constants also inialize segment, stack etc... must be inialize properly.
* e.g.,
  mov AX, 0500 H
  mov DS, AX

⑤ Choosing Instructions

We should choose those instructions that make program smaller in size and more importantly efficient in execution.
study instruction set & select proper instruction.

⑥ Converting algorithm to ALP.

Every step in algorithm is converted into program statement using correct and efficient instructions or group of instructions.

2.2 Assembly Language Programming Tools:- Editors, Linker, Assembler, Debugger.

Assembly Language Programming Tools.

Questions.....

Q// State the function ALP tools :-
i) Assembler ii) Linker iii) Debugger. (6 m)

Q// State fun^n of editor & assembler. (2m)

Q// split question -> Debuggin using tools. (3m) ✱ (2m)

Q// List assembly language programming tools. (2m) ✱✱

✱ These tools are program which help you run the program, to perform some function on the program you are writing.

✱ These tools are pre-developed program software that comes to aid of developer / programmer for writing, assembling / linking, running, testing and debugging the program at various steges the Program Developing Steps.

① Editor

✱ An editor is a tool which allows you to create a file that contain Assembly language statements for your program.

✱ Program is typed using Editor.

✱ Used to edit program & is called source program.

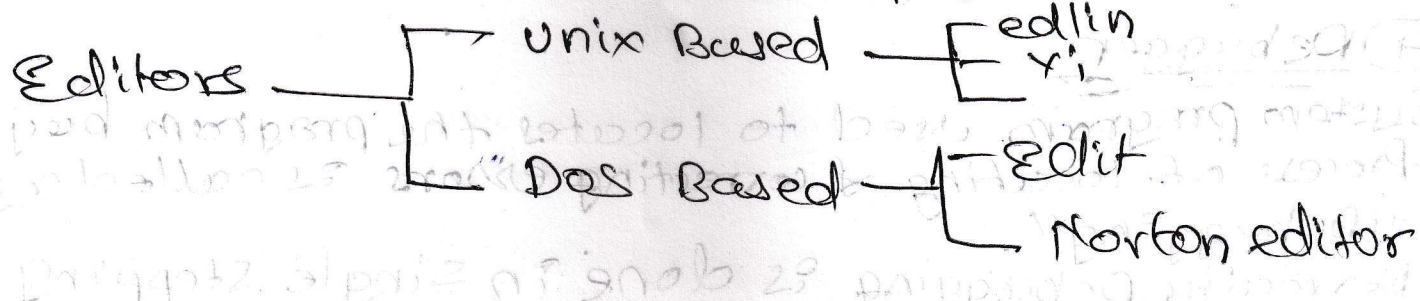
✱ It is needed to store ALP source code file with .Asm extension.

✱ Example :- Filename.Asm

✱ With editor, we are able to create, edit, save, update, delete the assembly language source files of program.

✱ Example, DOS, Notepad & Wordpad.

✱

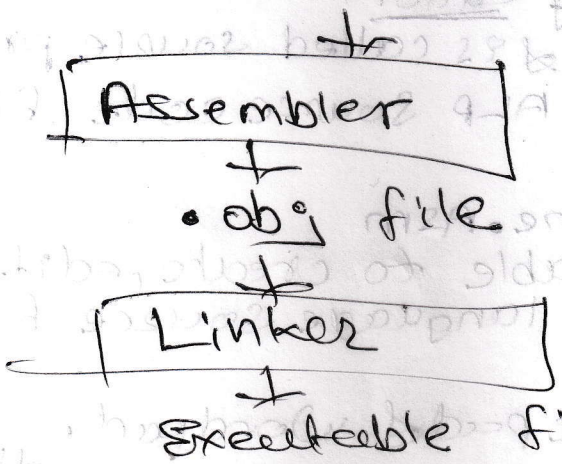


② Assembler

* System program that converts ALP (Source program) into machine code.
* Common assemblers: NASM, TASM, MASM etc...
* Assembler is a program used to convert ALP instruction to corresponding binary code (machine code) what is called object code.
* It uses source file with .asm extension as input.
* Generates the object file with extension .obj.
* Other file extension .LST (Debugging & Testing)
* Normally, 2-pass assemblers are used.

③ Linker

* Linker is a program use to convert machine language object code of program into executable code of program.
* System program that links (combines) all the object modules to generate executable module.
* Converts object files to executable file in short.
* Generates executable module with extension .exe. module refers to part of program.
* Common linkers: LINK, TLINK etc...



④ Debugger

- System program used to locates the program bugs
- Process of locating & correcting errors is called as "Debugging".
- Normally Debugging is done in single stepping

- IF program does not require any extended hardware or console input or output then debugger is used to load .Exe file in main memory & run it.
- Debugger Generates (.DBG) file, while help programmer to test programs.
- Logical errors in the program are deleted in process of debugging.
- Common debuggers: Turbo Debugger (TD), DEBUG

2.3 Assembler directives

Questions -- -- --

Q Describe following assembler directives. (4M)
i) DB ii) EQU iii) Segment iv) Assume

Q Explain use of Assembler directives
i) DW ii) EQU iii) ASSUME iv) OFFSET v) SEGMENT vi) EVEN (6M)

Q Explain any two assembler directives of 8086 (4M)

Q Write ALP to find largest number in array. (4M)

Q Write ALP to MUL two 16 bit signed no. (4M)

Q Write ALP to add two 8 bit numbers. (2M)

Q Write ALP to count No of 1's in 16 bit number.

Q Write ALP to count no of '0' in 8 bit. (4M) (4M)

Q Write ALP to arrange no in ascending order.

Q ALP to add two 16-bit numbers. (4M)

Q ALP to count no of 0's in 16 bit number. (4M)

Q Write ALP to find largest number in array elements 10H, 24H, 02H, 05H, 17H. (4M)

Q Write ALP to count no. of positive and negative number in array. (4M)

Q Write ALP to count ODD and Even number in array. (4M)

Q Explain following directives (4M)
i) DB ii) DUP iii) EQU iv) SEGMENT

Q i) DW ii) EQU iii) ASSUME iv) OFFSET v) SEGMENT vi) EVEN.

Assembler directives

- Assembler directives give the direction to assembler.
- called as "Pseudo-instructions".
- Assembler directives do not generate machine executable code.

various types of Assembler directives

1. Data Definition & storage allocation directives.
2. Program organization directives.
3. Procedure definition directives.
4. macro definition directives.
5. Data control directives.

Data Definition & storage allocation directives.

DB: (Define Byte) type variable
This directive is used to declare a BYTE-2BYTE variable - A BYTE is made of 8 bit.

Syntax: variable_name DB value

Example: Num DB 20h
A2 DB 30h.

DW: DW (Define word) type variable.
This is used to ~~decare~~ declare a word type variable. A word occupies (16 bits)/2 byte.

DD: Double define word (32-bit)
2 words or 4 BYTE.

DT/DQ

EQU: Equate to.
This directive is used to give name to some value or symbol.
Each time assembler finds the given names in program it will replace the name with value or symbol.
Used to assign constant value to symbol.

Symbol_name EQU Value.

PI EQU 3.14

Num EQU 100.

ASSUME

- Conveys logical name of segment to the assembler.
- Segment register points the logicael segment.
- Eg Assume CS: code, DS: Datea.
- The statement Assume CS: CODE, tells assembler that instreetions for program are in logicael segment name code.
- The statement ASSUME DS: DATA, tells a assembler that instreetion for program are in logicael segment called DATA.

SEGMENT

- Conveys the / Indicates the begining of logicael segment.
- It is name given to the segment.
- Exampde, the code segment is used to indicaate the assembler to start a logicael segment.
- Syntax: Data SEGMENT
  ; declaration / logic
  Data ENDS ; ENDS: End of segment.

DUP

- The DUP directive tells the assembler to duplicate an expression a given number of times.

for, example

4 DUP(2), eqivalent to 2,2,2,2.

mainly used to perform array operations

As it can be seen an used in conjuction with Assembler directive. (DB, DD, DT)

- allocate memory location.

A DB 9 DUP(?)

This would occupy 9 memory location and locate to A.

OFFSET

Offset is a operator which tells assembles to determine the offset address of variable.

when the Assembles reads the statements (mov BX, OFFSET STR) it will determine the offset of STR and will load this value in BX.

- Ex, A DB "abcd"; here A stores string abcd mov BX, offset A; will load BX register, offset

EVEN

This directive is used to align the memory allocation of a variable on an even boundary.
As a assembler uses the location counter to keep track of data.
EVEN directive tells the assembler to increment the location counter to next even address.
If it is not an even address.

A DB 9 DUP(?)
This would occupy 9 memory location and allocate to A EVEN. SKIPS location, start next memory location at even boundary.

Procedure definition directive

+ detailed units

PROC: Indicates the begining of procedure

Name of procedure followed by PROC & then the type (near or far)

- near → within segment
- far → other segment.

Syntax: Procedure name PROC {Near/Far}

Example: FACTORIAL PROC NEAR
{Procedure code}
FACTORIAL ENDP ; ENDP = End of Procedure.

Programs

Addition/Subtraction of 2-, 8bit & 16bit number.

multiplication/ Division of 2, 8bit & 16bit number.

Addition of 2, 8bit numbers.

```
data segment
a db 20h
b db 30h
c dw ?
data ends

code segment
Assume cs:code, ds:data
start: mov ax, Data
       mov ds, ax
       mov al, a
       mov bl, b
       add al, bl ; sub al, bl
       mov c, ax
       int 3
       code ends
       End
```

Addition of 2 16-bit no

```
data segment
a dw 2020h
b dw 1010h
c dd ?
data ends
Code segment
Assume CS:code, DS: Data
Start: mov ax, Data
       mov DS, ax
       mov ax, a
       mov ax, b
       add ax, bx ; Sub ax, bx
       mov c, ax
       int 3
       code ends
       End
```

MULTIPLICATION OF 16-bit number:

```
data Segment
a dw 1020h
b dw 2020h
c dd ?
data Ends

Code Segment
assume ds:data, cs:code
Start:
    mov ax, data
    mov ds, ax
    mov ax, a
    mov bx, b
    mul bx
    mov c, ax
    int 3
    Code Ends
    End.
```