COMPUTER ENGINEERING DEPARTMENT

ASSIGNMENT NO-02

SUB: SPCC

COURSE: T.E. **Year: 2020-2021** **Semester: VI**
DEPT: Computer Engineering
SUBJECT CODE: CSC602 **SUBMISSION DATE: 20/04/2021**

==========

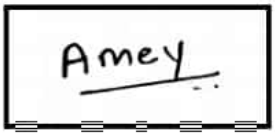
Name: Amey Thakur **Roll No.: 50**

Class: TE Comps B-50 **ID: TU3F1819127**

Assignment No 2

Sr. No.	Question
1	Explain the forward reference problem and how it is handled in assembler design.
2	For the following code what will be the output generated by Pass-I and Pass-II of the assembler. Explain with databases.
3	Explain different types of loaders in details.
4	Explain the working of the direct linking loader with an example, showing entries in different databases built by DLL.
5	Explain the working of the two-pass macro processor with neat flowcharts and databases.
6	Explain different features of macro with example.

Stmt no	Symbol	Op-code	Operands
1	ABC	START	0
2		USING	*, 15
3		L	1, FIVE
4		A	1, FOUR
5		ST	1, TEMP
6	FOUR	DC	F '4'
7	FIVE	DC	F '5'
8	TEMP	DS	1F
9		END	



AMEY B 50 Amey

Page No.:
Date:
YOUVA

Q1 Explain the forward reference problem and how it is handled in assembler design.

Ans:

- The rules of assembly language program state the symbol can be defined anywhere in the program.
- There can be cases in which the symbol is used before it is defined and such a reference is called forward reference.
- Due to forward reference the entry of the symbol cannot be found in the symbol table due to which assembler cannot assemble the instruction. Such a situation is called forward reference problem.
- To solve forward reference problem, assembler will make two passes over the input program.
- The purpose of pass 1 is to define the symbols and the literals encountered in the program.
- The purpose of pass 2 is to assemble the instructions and data.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Q.2 For the following code what will be the output generated by Pass - I and Pass - II of the assemblers. Explain with databases.

Stmt No.	Symbol	OP- Code	Operands
1	ABC	START	0
2		USING	*, 15
3		L	1, FIVE
4		A	1, FOUR
5		ST	1, TEMP
6	FOUR	DC	F '4'
7	FIVE	DC	F '5'
8	TEMP	DS	1F
9		END	

Ans:

Mnemonic Opcode Table (MOT):

Mnemonic OPcode	Binary Opcode	Inst - length	Instruction Format
L	010	4	RX
A	101	4	RX
ST	110	4	RX
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮

AMEY B 50 Amey

Page No.:
Date:

Pseudo Opcode Table [POT] :

Pseudo Opcode	Address of the routine to process Pseudo-OP
START	
DC	
DS	
USING	
DROP	
END	
⋮	
⋮	

Symbol Table [ST] :

Symbol	Value	Length	Relocatable / Absolute
ABC	00	01	R
FIVE	12	04	R
FOUR	16	04	R
TEMP	20	04	R

AMEY B 50 Amey

Page No.:
Date:
youva

Base Table [BT]:

	Availability Indicator	Contents of B.R
1	'N'	
2	'N'	
12	'N'	
15	'Y'	00

Pass N			Pass 1		Pass 2		
LC			LC		LC		
ABC	START	0	0		0		
	USING	*, 15	0		0		
	L	1, FIVE	0	L1, _	0	L	1, 16 (0, 15)
	A	1, FOUR	4	A1, _	4	A	1, 12 (0, 15)
	ST	1, TEMP	8	ST1, _	8	ST	1, 20 (0, 15)
FOUR	DC	F '4'	12	'4'	12	0100	
FIVE	DC	F '5'	16	'5'	16	0101	
TEMP	DS	'1' F	20	_	20		
	END		24		24		

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Q3. Explain different types of loaders in detail.

Ans:

Types of loader

1. Compile and Go Assembler / Compile and Go Loader
2. Absolute Loader
3. Relocating Loader
4. Direct Linking Loader
5. Dynamic Loading Loader
6. Dynamic Linking Loader

Note: These loaders primarily differ in the manner in which the four basic functions are accomplished.

1. Compile and Go Loader

- A compile and go loader is one in which the assembler itself does the processes of compiling then place the assembled instruction in the designated memory locations. The assembly process is first executed and then the assembler causes a transfer to the first instruction of the program.

Advantages

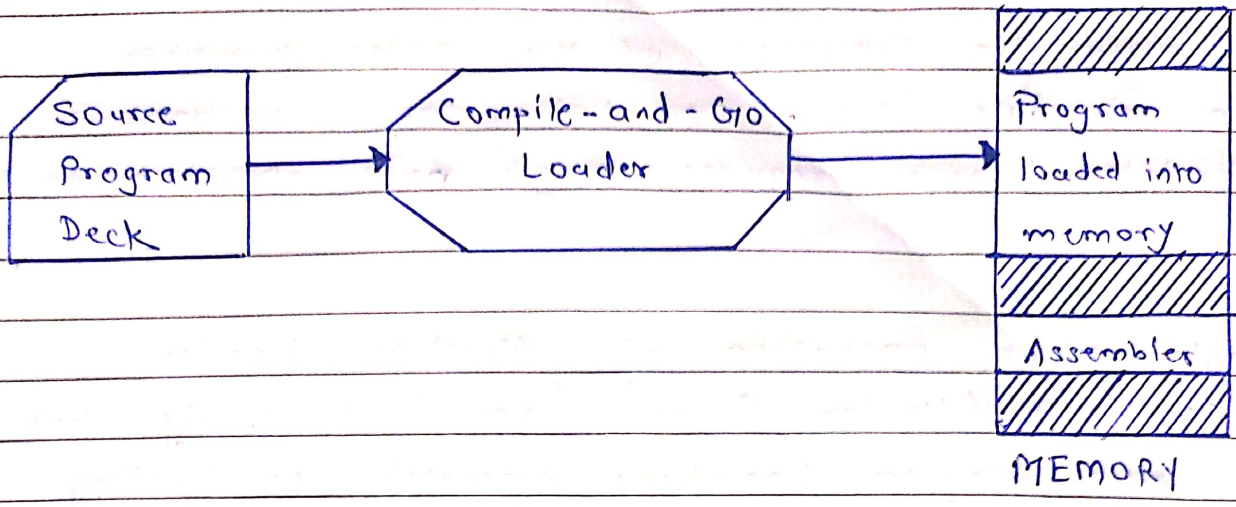
- Simple to implement
- No extra procedures are involved.

Disadvantages

- Wastage of memory
- It retranslates the user's program code every time it is executed.
- When the source code is in different program is in different languages it is hard to handle multiple segments.

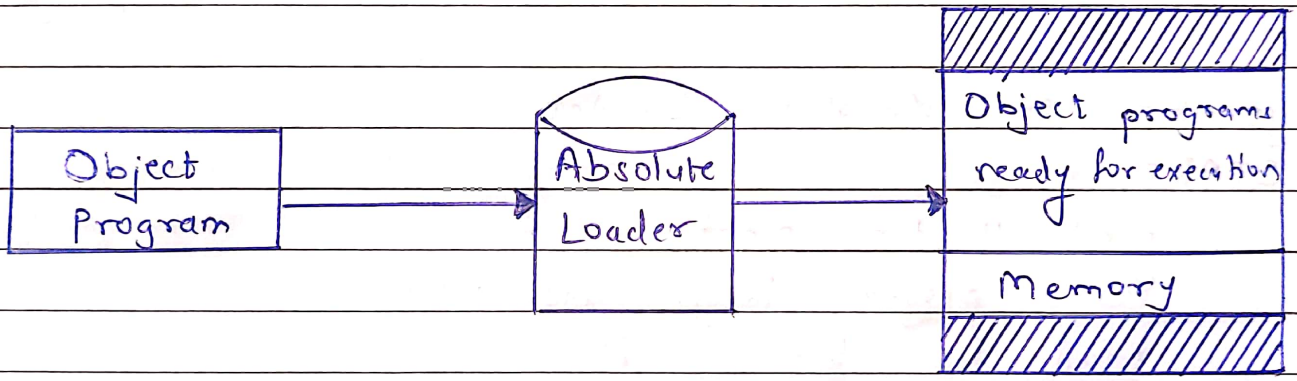
AMEY B 50 Amey

Page No.: youva
Date:



Assemble and Go Loader

(2) Absolute Loader



Absolute Loader

- This type of loader scheme which is used in above figure is called an Absolute Loader.
- The loader simply accepts machine language code and places it into main memory specified by the assembler.
- It is similar to "Compiler and Go" loaders.
- The loader accepts machine language instructions and places it into the core, then initiates execution by transferring control to the starting of the program in this the data is stored on cards instead of being placed in main memory for execution.
The object code is loaded at the specified locations in the memory

AMEY B 50 Amey

Page No.:
Date:
YOUVA

- At the end the loader jumps to the specified location to begin the execution of the loaded program.
- In absolute loader no linking or relocation is done.
- Absolute loaders requires single pass operation, in that it checks.

Single pass operation of Absolute Loader

- Check H (Header Record) record to verify that correct program has been presented for loading.
- Read each T (Text Record) record, and move object code into the indicated address in memory.
- At E (End Record) record, jump to the specified address to begin execution of the loaded program.

Algorithm for Absolute Loader

```
START
    read Header record
    Verify program length and name
    read first text record
    while record type ! = E do
        begin
            if object code is in character form converts it
            into internal representation.
            Move object code to specified location in memory
            read next object program record.
        end
    Jump to address specified in End record.
STOP
```

Page No.:
Date:
YOUVA

Advantages
- Simple and efficient to implement

Disadvantages
- The programmer must specify to the assembler the address in the core where the program is to be loaded.
- In case of multiple subroutines the programmer must remember the address of each and use that absolute address explicitly in other subroutines to perform a subroutine linkage.

The four basic functions of an absolute loader are
(i) Allocation - done by programmer
(ii) Linking - done by programmer
(iii) Relocation - Loaded where assemble assigns
(iv) Loading - By Loader.

(3) Relocating Loader
- The task of relocating loader is to avoid reassembling of all subroutines when a subroutine is changed and to execute tasks of allocation and linking for programmer.
- Relocating loaders adjust addresses of the executable code to compensate for variations in the address at which loading starts.
- The computer which need relocating loaders are those in which pointers are absolute addresses rather than offset from the programs base address.
- Binary symbolic subroutine (BSS) loader is an example of Relocating Loader.

AMEY B 50 Amey:

Page No.: YOUVA
Date:

- BSS performs all four functions of the loader.
  (i) Allocation - It is done with the help of a program length and the information.
  (ii) Linking — It is done by transfer vector.
  (iii) Relocation — With the help of the relocation bits.
  (iv) Loading - It is done by the loader.

For each source program the assembler outputs
- The machine translation code of the program
- Transfer vector are used to store the addresses containing names of the subroutines referenced by the source program.
- Relocation bits are associated with each instruction or address fields which includes:
  - Length of the entire program
  - Length of the transfer vector portion.
  - Operation performed by the Relocating Loaders.
- Assembler assembles each segment independently and passes on to the loader, the text and the information for the relocation and the intersegment references.
- The BSS loader allows many procedure segments and only single data segment.
- The loader uses the transfer vector to solve the problem of the subroutine linkage.
- The length of the whole program and the length of the transfer vector are given by the assembler.
- Relocating loader will load all the information regarding subroutines and would place the equivalent addresses of the subroutines to each position of the transfer vector.
- The output of relocation assembler using a BSS scheme is the object program and the information about all other programs is referenced.

- If the relocation bit is one, then the corresponding address bit must be relocated otherwise field is not relocated.

Disadvantages
- The transfer vector is used only for transfers and not used for loading or storing of external data.
- The size of the object program in the memory is increased due to transfer vector.
- The BSS loader processes procedure segments but does not facilitate access to the data segments that can be shared.

(4) Direct Linking Loader
- A Direct Linking Loader is the most popular loading scheme used.
- It allows the programmer to use the multiple procedure and data segments giving user a complete choice in referencing data or instruction enclosed in other parts of segments.
- It provides flexible intersegment referencing accessing ability.
- Assembler provides following information to loader with respect to each procedure or data segment:
  - Segment length
  - Records of all the symbols that may be referenced by other symbols and their relative location.
  - Records of all the symbols referenced in the segment but not defined in the segment are also maintained.
  - Explaination about the address constants, their location in the segment and information about their revise values.
  - Information about the source codes machine code translation and the relative addresses assigned.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Disadvantages

- In Direct Linking Loader, it is necessary to allocate, relocate, link and load all of the subroutines each time in order to execute the program. So loading process can be extremely time consuming.
- Even though the loader program may be smaller than the assembler, it does absorb a considerable amount of space.

- The mentioned problems can be solved by dividing the loading process into two separate programs :

① Binder - A binder is a program that performs the same function as direct linking loader in "binding" subroutines together.
The binder performs allocation, relocation and linking.

② A Module Loader - It outputs the text in a file rather than memory.
The module loader merely has to physically load the module into the memory.

AMEY B 50 Amey

Page No.: Date: YOUVA

⑤ Dynamic Loading Loader

- If all the subroutines are loaded simultaneously into the core, then there may be chance, that the core available may be insufficient for the subroutines, this may lead to further complications.
- Execution of such program can be possible if all the segments are not required simultaneously to be present in the main memory.
- In such situations only those segments are residents in the memories that are actually needed at the time of execution.
- But the question arises what will happen if the required segment is not present in the memory?
- Naturally the execution process will be delayed until the required segment gets loaded in the memory.
- Overlay structure is used to specify the inter dependency between all the segments.
- It consist of nodes and edges and segment is represented by the nodes.
- If the two are on the same path they can lie in the main memory.
- For solving such problems techniques like segmentation and paging is used.
- In these the subroutines are loaded into core at a different time because the subroutines in a program are needed at different times. i.e. they may be mutually exclusive, by identifying which subroutine can calls other subroutines to produce an overlay structure.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Overlay Structure is used to :

- Keep in memory only those instructions and data that are needed at any given time.
- Overlay structure needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.

- The flipper or overlay supervisor is the portion of the loader that actually intercepts the "calls" and loads the necessary procedure.

Advantages

- No wastage of memory
- A system can be dynamically loaded whenever it is required.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

⑥ Dynamic Linking Loader

- Dynamic Linker Loader is a special part of an operating system that loads external shared libraries into a running process and then binds those shared libraries dynamically to the running process. This approach is also called dynamic linking or late linking.
- It retrieves the addresses of functions and variables contained in the library, execute those functions or access those variables, and unload the library from memory

- It is often used to allow several executing programs to share one copy of subroutine or library. For eg., run time support routines for C could be stored in dynamic link library. A single copy of the routines in this library could be loaded into the memory of the computer. All C programs currently in execution could be linked to this one copy, instead of linking a separate copy into each object program.
- It provides the ability to load the routines only when they are needed so lot of time and memory is saved if subroutines are large with lots of external references.
- It helps in not loading the entire library for execution.
- In dynamic linking loader is used to load the main program

Advantage

- No over head is incurred unless the procedure to be called or referenced is actually used.
- A further advantage is that the system can be dynamically configured.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Disadvantage

- Considerable overhead and complexity incurved, due to the fact that we have postponed most of the binding process until the execution time

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Q.4. Explain working of direct linking loader with example, showing entries in different databases built by DLL.

Ans:

Source	Relative		SAMPLE PROGRAM
1	0	LOOP1	START
2			ENTRY LOSTART0, LOSTART1
3			EXTRN L1START0, L1START1
4	8	LOSTART0	
5	12	LOSTART1	
6	16		DC B(LOSTART0)
7	20		DC B(LOSTART1 + 10)
8	24		DC B(LOSTART1 - LOSTART0 - 2)
9	28		DC B(L1START1)
10	32		DC B(L1START0 + L1START1 + 5)
11			
12	0	LOSTART1	START
13			ENTRY L1START0
14			EXTRN LOSTART0, LOSTART1
15	4	L1START0	
16	14	DC	A(LOSTART0)
17	18	DC	A(LOSTART1 + 10)
18	22	DC	A(LOSTART1 - LOSTART0 - 5)
19	END		

Sample program for LOOP1 and L1START1

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Object deck program for LOOP 1:

ESD Records

Source Card Reference	Variable Name	ESD		Relative Address	Length
		Type	ID		
1	LOOP 1	SD	01	0	36 (START = 0; END = 36)
2	LOSTART 0	LD	02	08	-
2	LOSTART 1	LD	02	12	-
3	L1START 1	ER	03	-	-
3	L1START 0	ER	03	-	-

Where,

SD - Segment Definition (Hexadecimal code = 01)
LD - Local Definition (Hexadecimal code = 02)
ER - External Reference (Hexadecimal code = 03)

TXT Records

Source Card Reference	Relative Address	Content	What the assembler did?
6	16 - 19	08	
7	20 - 23	22	12 + 10 = 22
8	24 - 27	02	12 - 08 - 02 = 02
9	28 - 31	0	Not known to LOOP 1
10	32 - 35	+5	(5) L1START 0. L1START 1 NOT KNOWN TO ASSEMBLER

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Source Card Reference	Variable Name	ESD ID	Length (By K)	Arithmetic Operator	Relative Address
6	LOSTARTO	02	4	+	16
7	LOSTART1	02	4	+	20
9	L1 START1	03	4	+	24
10	L1 STARTO	03	4	+	28
10	L1START1	03	4	+	28

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Q.5 Explain the working of the two-pass macro processor with neat flowcharts and databases.

Ans:

- It is used for identifying the macro name and performing expansion.

Features of macro processor :

1. Recognized the macro definition
2. Save macro definition
3. Recognized the macro call
4. Perform macro expansion.

Forward Reference Problem

- The assembler specifies that the macro definition should occur anywhere in the program.
- So there can be chances of macro call before it's definition witch gives rise to the forwards reference problem od macro
- Due to witch macro is divided into two passes.
  1. Pass - 1
     - Recognize macro definition save macro definition.
  2. Pass - 2
     - Recognize macro call perform macro expansion.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Databases required for Pass-2

- In Pass-2 we perform recognize macro call and perform macro expansion.

① COPY FILE
- It is a file which contains the output given from Pass-1.

② MNT [Macro Name Table]
- It is used for recognizing macro name.

③ MDT [Macro Definition Table]
- It is used to perform macro EXPANSION.

④ MDTP [Macro-Definition Table Pointer]
- It is used to point to the index of MDT. The starting index is given by MNT.

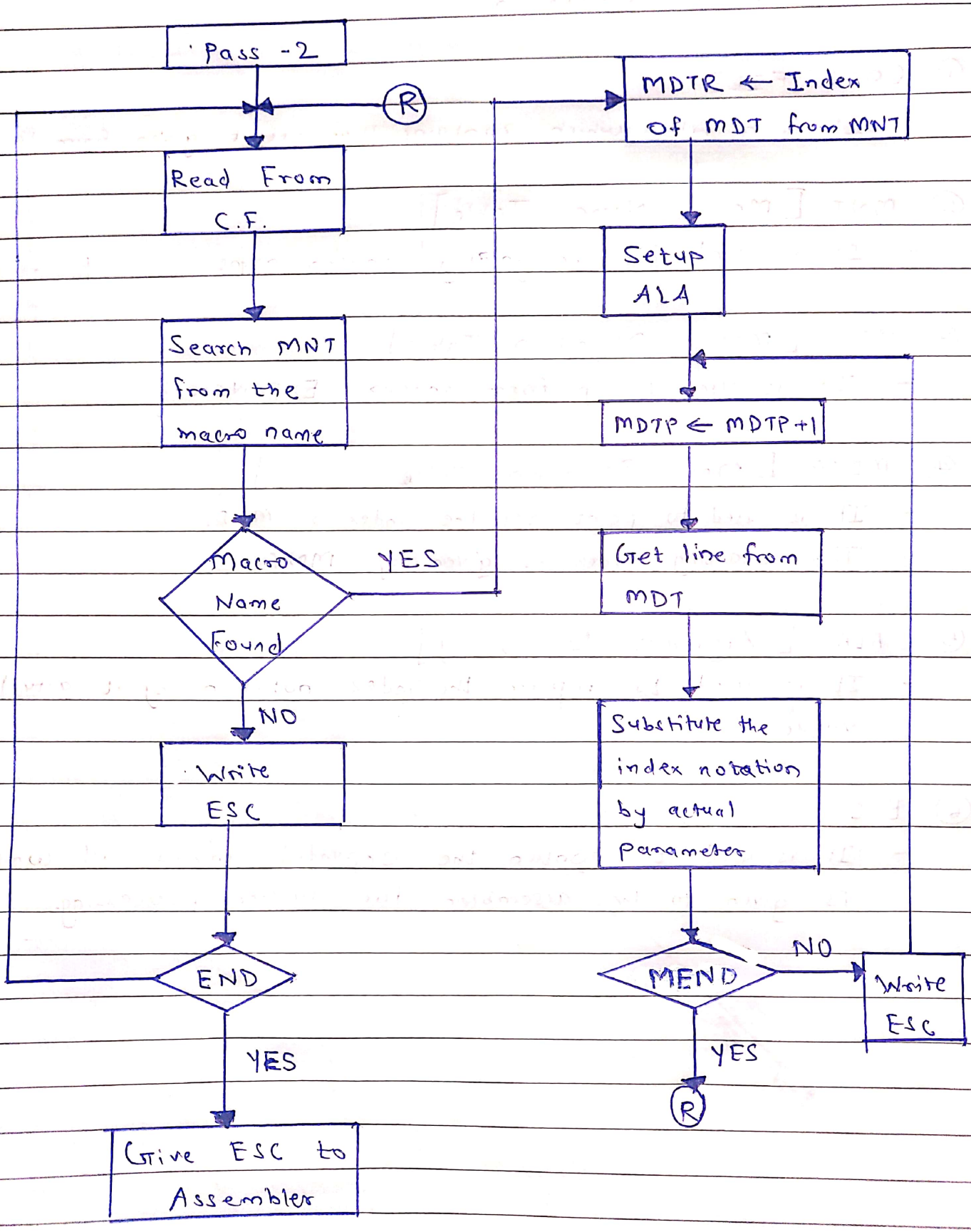
⑤ ALA [Argument List Array]
- It is used to replace the index notation by it actual value.

⑥ ESC
- It is used to contain the expanded macro call which is given to the assembler for further processing.

AMEY B 50 Amey

Page No.:
Date:
YOUVA

Flowchart :



AMEY B 50 Amey

Page No.:
Date:

Q.6. Explain different features of macro with example.

Ans:

Macro Instruction Arguments

Consider the following example: The modified source code with macro definition

```
                    |          MACRO
    A   1, FIVE     |          ADDM  &ARG  ← Macro argument
    A   2, FIVE     |          A     1, &ARG
    A   3, FIVE     |          A     2, &ARG
                    |          A     3, &ARG
    A   1, FOUR     |          MEND          { FIRST A 1, D1
    A   2, FOUR     |                          A 2, D2
    A   3, FOUR     |                          A 3, D3
                    |          ADDM  FIVE  ← Call with argument
                    |          ADDM  FOUR  → { SECOND A 1, D3
                    |                          A 2, D2
                    |                          A 3, D1
FIVE  DC  F '5'     |          FIVE  DC  F '5'
FOUR  DC  F '4'     |          FOUR  DC  F '4'
                    |
```

- Every argument name will start with ampersand symbol (&)
  Example: &ARG is the dummy argument.
- The macro call will be accompanied by the argument
  Example: ADDM FIVE
- Thus when macro is expanded each occurence of &ARG is replaced by FIVE.
- It is possible to pass more than one argument in macro call. When we pass more than one dummy argument there are two ways to specify these arguments.

⇒ Positional Arguments: Here the arguments are matched with dummy arguments in the same order they appear.

⇒ Keyword Arguments: Here the arguments are referenced by their name and position.

Example: ADDM &ARG1 = D1, &ARG2 = D2, &ARG3 = D3.