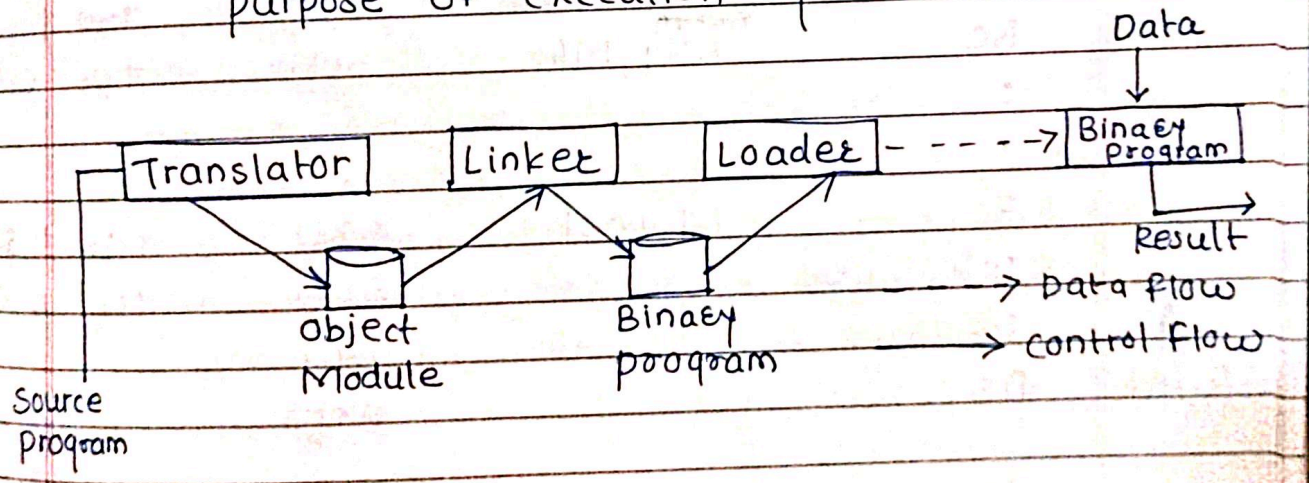
5. Linkers

Q. 1] What steps are involved in execution of program? explain translated, linked & load time addresses with example.

A] Execution of program written in a language L involves following steps :-

1. Translation of program by translator of language L.
2. Linking of program with other program needed for its execution by linker.
3. Relocation of program to execute from the specific memory area allocated to it by a linker.
4. Loading of a program in memory for the purpose of execution by a loader.



Schematic of program execution

B] 1. Translated address :- address assign by translator
2. Linked address :- address assign by linker
3. Load time address :- address assign by loader
4. Translated origin :- address of origin assumed by translator whis is specified by the programmer in an ORIGIN statment.

(5) Linked origin :- address of the origin assigned by the linkee while producing a binaey program.

(6) Load time address :- address of the origin assigned by loadee while loading the program for execution.

ex. :-

	Statement		address	code
	START	500		
	ENTRY	TOTAL		
	EXTRN	MAX, ALPHA		
	READ	A	500)	+90 0 540
LOOP	⋮		501)	
	MOVER	AREG, ALPHA	518)	+40 1 000
	BC	ANY, MAX	519)	+06 6 000
	BC	LT, LOOP	538)	+06 1 501
	STOP		539)	+00 0 000
A	DS	1	540)	
TOTAL	DS		541)	

Translated origin of the program = 500
Translation time address of loop = 501
Suppose load time origin = 900
load time address of loop = 901

Q. 2] explain program relocation & how to perform program relocation with example.

→ A] Program relocation :-

(i) Program relocation is the process of modifying the addresses used in address sensitive instructions of program such that the program can execute correctly from the designated area of memory.

(ii) If linked origin ≠ translated origin, relocation must be performed by the linker.

(iii) If load origin ≠ linked origin, relocation must be performed by the loader.

(iv) If load origin = linked origin, such loaders are called absolute loader.

B] Performing Relocation :-

- t_{origin_P} translated origin of program P
- l_{origin_P} linked origin of program P
- t_{symp} translation time address
- l_{symp} link time address
- d_{symp} offset of symbol

[Program P]

	Statement		Address	Code
	START	500		
	ENTRY	TOTAL		
	EXTRN	MAX, ALPHA		
	READ	A	500)	+09 0 540
LOOP	.		501)	
	.			
	MOVER	AREG, ALPHA	518)	+04 1 000
	BC	ANY, MAX	519)	+06 6 000
	.			
	.			
	BC	LT, LOOP	538)	+06 1 501
	STOP		539)	+00 0 000
A	DS	1	540)	
TOTAL	DS			
	END			

Translated origin of the program = 500
Suppose link origin = 900
Relocation factor = 900 - 500
= 400

Q.3] Explain concept of linking. What are binary program & object modules used in linking?

→ A] Linking :- linking is the process of binding an external reference to the correct link time address.

Program consist of

1] Public definition :- a symbol may be referenced in other program unit. The ENTRY statement list the public definition of a program unit.

2] external reference :- a reference to a Symbol which is not defined in the program unit containing reference. The EXTRN statement lists the symbol to which external references are made in the program unit.

ex.:- refer [program P]

[Program Q]

statment			Address	code
START	200			
ENTRY	ALPHA			
---.				

ALPHA	DS	25	23L)	100 0025
	END			

- let the link origin of P be 900 & its size be 42 words.
- The link origin of Q is therefore 942 & link time address of ALPHA is 973.
- linking is performed by putting link time address of ALPHA in the instruction of P using ALPHA.

B] Binary program: - 1] A binary program is a machine language program comprising a set of program units SP such that ∀ SP ∈ P_i.

(i) P_i has been relocated to memory area starting at its link origin and

(ii) Linking has been performed for each external references in P_i.

2] To form a binary from a set of object module, the programmer invokes the linker using the command:

linker <link Origin>, <object module names> [<execution start address>]

C] object Module :- (1) Object module of a program contains all information necessary to relocate & link the program with other program.

2] It consist of 4 components:

1. Header: It contains (a) Translated origin
(b) size
(c) execution start address of program P.

2. Program: It contains machine language program corresponding to program P

3. Relocation Table (RELOCTAB):

(a) It describe set of instruction requiring relocation.

(b) It contains Translated address of address sensitive instruction.

4. Linking Table (LINKTAB): 1] It contains information concerning public definition and external references.

2] This table contains:

(a) Symbol: symbolic name

(b) Type : PD/EXT

(c) Translated address

- For public def^{n}., this is the address of first memory word allocated to symbol.
- For external reference, it is address of memory word which is required to contain the address of the symbol.

ex.. refer [Program P]

-for program P

1. Header : translated origin = 500, size=42, execution start address=500
2. Machine language instruction shown in fig.
3. Relocation table:

translated address
500
530

linking table:

Symbol	Type	Translated addres
ALPHA	EXT	518
MAX	EXT	519
TOTAL	PD	541

q. 4] explain program Relocation algorithm.

→ 1. Program linked origin := <link origin> from linker command;

2. For each object module
   a) t origin := translated origin of the object module
      OM_size := size of the object module
   b) relocation factor := program linked origin − t origin;
   c) Read the machine language program in work area.
   d) Read RELOCTAB OF object module.
   e) for each entry in RELOCTAB
      i) translated address := address in RELOCTAB entry;
      ii) address_in work area := address of work_area + translated address − t origin;
      iii) Add relocation_factor to operand address in word with the address address_in_work_area.
   f) program linked origin := program linked origin + OM size;

Q.5] Explain program linking algorithm with example.

→ 1. Program linked origin := <link origin> From linker command.

2. For each object module
   a) }
   b) } same steps as relocation algorithm.
   c) }
   d) Read LINKTAB of object module.
   e) For each LINKTAB entry with type=PD
      name := symbol;
      linked_address:=translated + relocation_factor;
      Enter (name, linked_address) in NTAB.
   f) Enter (object module name, program linked origin) in NTAB;
   g) program linked origin := program lined origin + OM size

3. For each object module
   a) t origin := translated origin OF object module;
      program linked origin := load address from NTAB;
   b) For each LINKTAB entry with type = EXT
      i) address_in_work_area := address OF work_area +
         program_linked_origin_<link origin> +
         translated address - t_origin;
      ii) Search symbol in NTAB & copy it linked address. Add the linked address to the operand

address in the word with the address address - in work area.

Q.6] Discuss about Self-Relocating programs.

→ Self-Relocating programs have two types:-

(1) Non relocatable program: It is a program which can not be executed in any memory area other than the area starting on its translated origin.

(2) Relocatable program: It can be processed to relocate it to a desired area of memory.

(1) It is program which can perform the relocation of its own address sensitive instruction. It contains two provision.

(a) A table of information concerning the address sensitive instruction exists as a part of program.

(b) Code to perform the relocation of address sensitive inst. also exists as a part of program which is called the relocatting logic.

Q.7] explain linking for overlays with example.

→ (1) An overlay is a part of program or software package which has the same load origin as some other part of the program.

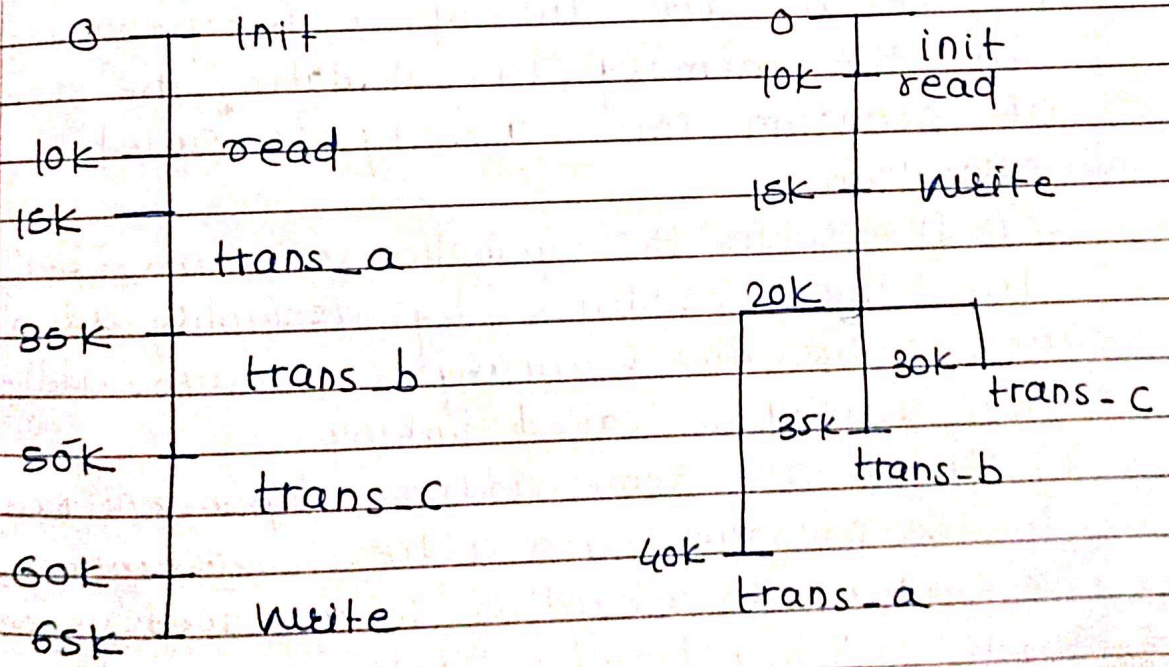
(2) overlays structured program consist of

- A permanently resident portion called the root
- A set of overlays.

③ The overlay structure of program is designed by identifying mutually exclusive modules.

④ example: consider a program with 6 sections named init, read, trans-a, trans-b, trans-c and print.

- init perform some initialization & transfer control to read.
- read reads one set of data & invokes one of trans-a, trans-b or trans-c depending on the values of data.
- print is called to print the result.
- trans-a, trans-b and trans-c are mutually exclusive.



Q.8] What is loader? What are the fun^n. of loaders.

→ (1) Loader is utility program which takes object code as input prepares it for execution & loads the executable code into the memory. Thus loader is actually responsible for initiating the execution process.

(2) Functions of loader:

The loader is responsible for the activities such as allocation, linking, relocation & loading

(i) It allocates the space for program in the memory, by calculating the size of program. This activity is called allocation.

(ii) It resolves the symbolic references bet^n. the object modules by assigning all the user subroutine & library subroutine addresses. This activity is called linking.

(iii) There are some address dependent locations in the program, such address constants must be adjusted according to allocated space, such activity done by loader is called relocation.

(iv) Finally it places all the machine instructions and data of corresponding programs & subroutines into the memory. Thus program now becomes ready for execution, this activity is called loading.

Q.9] Explain with schematic absolute & direct linking loader.

→ (1) absolute loader :-

i) absolute loader is a king of loader in which relocated object files are created, loader accepts these files and places them at specified locations in the memory.

ii) This type of loader is called absolute because no relocation information information is needed; rather it is obtained from the programmer or assembler.

iii) ~~The~~ In this scheme, the programmer or assembler should have knowledge of memory managment.

iv) The resolution of external references or linking of different subroutines are the issues which need to be handled by the programmer.

(2) Direct linking loader :-

i) The direct linking order is most common type of relocated loader. The loader cannot have the direct access to source code. And to place the object code in memory there are two situations : either the address of the object code could be absolute which then can be directly placed at the specified location or the address can be relative. If at all the address is relative then it is the assemblee who inform the loader about the relative addresses. The assembee should give the following information to loader:

(1) The length of the object code segment

(2) The list of all the symbols which are not defined in the current segment but can be used in current segment.

(3) The list of all the symbols that are defined in current segment but can be reffered by the other segments.

(4) The list of symbols which are not defined in current segment but can be used in the current segment are stored in table. The USE table holds the information such as name of the symbol, address, address relativity.

Q.10 explain with schematic bootstrap & compile and go loader.

→ A] compile and go loader : In this type of loader, the instruction is read line by line, its machine code is obtained & it is directly put in the main memory at some known address.

i) That means the assembler runs in one part of memory & the assembled machine instruction and data is —
-directly put into their assigned memory locations.

ii) After completion of assembly process, assign starting address of the program to the location counter.

iii) The typical example is WATFOR-77, it's a FORTRAN compiler which uses such "load and go" schema.

iv) This loading schema is also called as "assemble and go".

B] Bootstrap loader:

i) Automatically executed when the computer is first turned on.

ii) Loads the first program to be run: usually the o/s itself begins at address 0 in memory.

- loads the o/s starting at address 80
- After all code is loaded, bootstrap jumps to address 80
- NO H or E records, no control information.