System Programming : Unit 3 Linkers and Loaders

Prof. Reshma Pise
Comp Engg. Dept
Vishwakarma University

Course Outcomes :

On completion of the course, the students will be able to –

1. Discriminate among different System software and their functionalities.

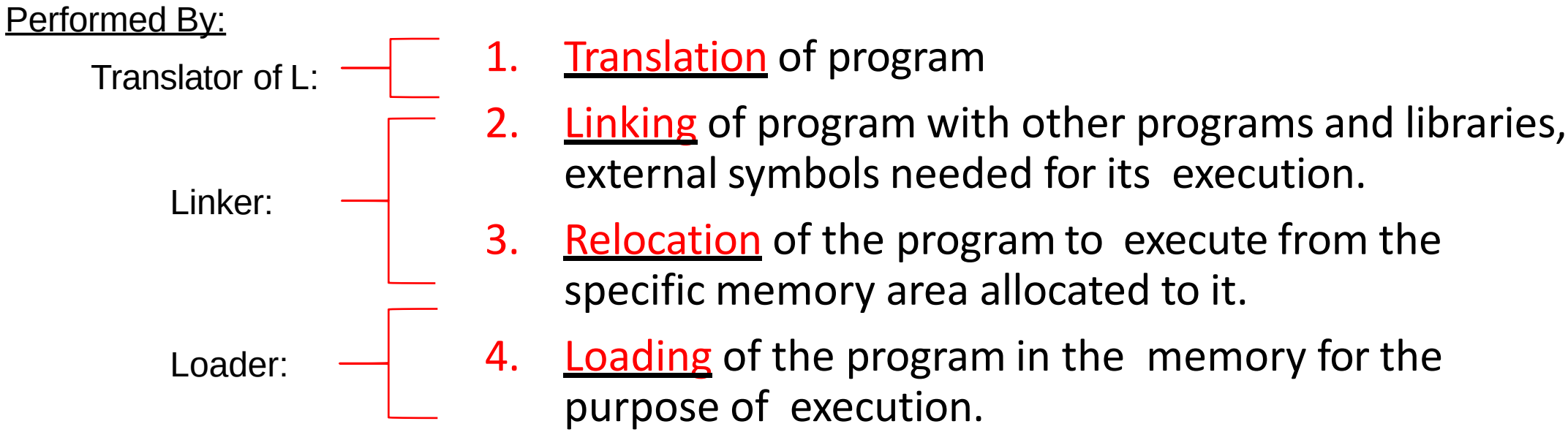
2. Design language translators like Macro processor and Assembler.

3. Develop approaches and methods for implementing compiler, linker and loader.
4. Use LEX tool for lexical analysis.
5. Interpret the techniques of implementing utility software.

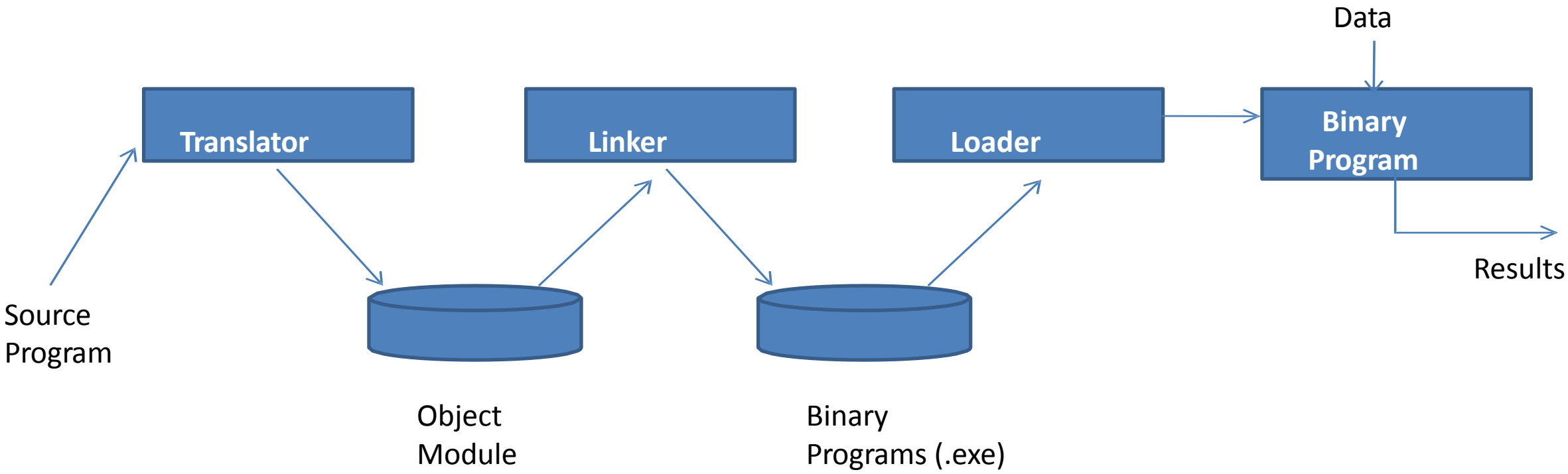
Outline

- Linkers: Relocation and linking concepts, static and dynamic linker, Subroutine linkages.
- Self-relocating programs.
- Loaders: Loader schemes: Compile and Go, General loader scheme, Absolute loaders,
- Relocating loaders, Direct linking loaders, Overlay Structure.
- Dynamic linking, API compatibility, Dynamically Linked libraries.

Schematic of Program Execution:



Schematic of Program Execution:



Terminologies:

- Translation Time (or Translated) Address: Address assigned by the translator.
  As specified in ORG / START or default value
- Linked Address: Address assigned by the linker.
- Load Time (or Load) Address: Address assigned by the loader.
- Translated Origin: Address of the origin assumed by the translator. This is the address specified by the programmer in an ORIGIN statement.
- Linked Origin: Address of the origin assigned by the linker while producing a binary program.
- Load Origin: Address of the origin assigned by the loader while loading the program for execution.

Change of origin

- The linked and load origins may differ from translated origin.
- Reason?
  - Same set of translated addresses may have been used by different object modules of the program. This results in conflict during memory allocation.
  - OS may require that a program should execute from specific area of memory. This may require change in its origin, thus changing execution start address and symbol addresses.
- Thus, origin may be changed by linker and loader.

Example: P

	Statement	Address	Code
	START 500		
	ENTRY TOTAL	The translated origin	
	EXTRN MAX, ALPHA		
	READ A	500)	+ 09 0 140
LOOP		501)	
	.		The translated time address of LOOP
	.		
	MOVER AREG, ALPHA	518)	+ 04 1 000
	BC ANY, MAX	519)	+ 06 6 000
	.		
	.		
	BC LT, LOOP	538)	+ 06 1 101
	STOP	539)	+ 00 0 000
A	DS 1	540)	
TOTAL	DS 1	541)	
	END		

RELOCATION & LINKING CONCEPTS

(a) Program Relocation

- Performing Relocation

(b) Linking

- EXTERN & ENTRY statement
- Resolving external references
- Binary Programs

(c) Object Module

Program Relocation:

Address Sensitive Program:

- AA – Set of Absolute Addresses, may be instruction address or data address.

- A Address sensitive program contains:
  1. Address Sensitive Instruction: an instruction which contains address $a_i \in AA$.
  2. Address Constant: a data word which contains an address $a_i \in AA$.
- Address Sensitive Program P can execute correctly only if the start address of the memory area allocated to it is the same as its translated origin.

  Start Address = Translation Address
- Thus, to execute correctly from any memory area, address used in each address sensitive instruction of P must be '**corrected**'.

Program Relocation : Definition

- The process of modifying the address used in the address sensitive instructions such that the program can execute correctly from the designated area of memory.
- Linker performs relocation if
  - Linked Origin ≠ Translated Origin
- Loader performs relocation if
  - Load Origin ≠ Linked Origin
- In general: Linker always performs relocation, whereas some loaders do not.
- Absolute loaders do not perform relocation, then
  - Load Origin = Linked Origin
  - Thus, Load Origin and Linked Origin are used interchangeably.

Performing Relocation:

- Relocation Factor (RF) of P is defined as

  $relocation\_factor_p = l\_origin_p - t\_origin_p$

  RF can +ve or –ve or 0.

- Consider a statement in which symbol is an operand

  $t_{symb} = t\_origin_p + d_{symb}$ where, d_{symb} = offset

  t_{symb} = translation time address l_{symb} = link time address

  t_origin_p = translation origin of Program ‘ P ‘

  After program P is relocated to linked origin l_origin_p, we have

  $l_{symb} = l\_origin_p + d_{symb}$

  $l_{symb} = t_{origin} + relocation\_factor_p + d_{symb}$

  $= t_{symb} + relocation\_factor_p$

IRR: Instruction Requiring Relocation

A set of instructions that require relocation in program p.

- Steps:
  - Calculate Relocation Factor(RF)
  - Add it to Translation Time Address(es) for every instruction which is member of IRR.

Eg : In previous program, if linked origin is 300

relocation factor = 300 - 100 = 200

IRR contains translation addresses 100 & 138

(instruction : read A) Operand addresses 140 and 101

Operand address is changed to 140 + 200 = 340 and

101 + 200 = 301.

Linking:

Application Program consists of a Set of Program units.

- A program unit Pi interacts with another program unit Pj using instructions & addresses of Pj.
- For this it must have:
  - Public Definition: a symbol pub_symb defined in a program unit which may be referenced in other program.
  - External Reference: a reference to a symbol ext_symb which is not defined in program unit containing the reference.

EXTRN & ENTRY statements:

- The ENTRY statement lists the public definition of program unit.
  - i.e it list those symbols defined in program unit which may be referenced in other program units.
- The EXTRN statement list the symbols to which external references are made in the program unit.
- Refer Example in slide Number 8
  - TOTAL is ENTRY statement. (public defination)
  - MAX and ALPHA are EXTRN statements (external reference)
  - Assembler don't know address of EXTRN symb and so it puts 0's in address field of instruction wherever these symbols are used.
  - It is required to resolve external references.

Resolving External Reference:

- Every external reference should be bound to correct link time address , before application program executes.
- Who will do this binding?
  – Here comes Linker into the picture.
- **Linking**:

  Linking is the process of binding an external reference to correct link time address.
- External reference is said to be unresolved until linking is performed and resolved once linking is completed.

Example: Linking

Consider Program Unit Q linked with P (Slide No. 8)

	Statement	Address	Code
	START 200		
	ENTRY ALPHA		
	- -		
ALPHA	DS 25	231)	+ 000 025
	END		

ALPHA is external reference in example ‘P’ which is linked with above given unit.

Binary Programs:

Machine language program comprising of set of program units SP such that for all $P_i \in SP$,

- P_i relocated at linked origin.
- Linking is performed for each external reference in P_i.

Linker Command:

Linker <link origin> <object module names> [, <execution start address>]

- Linker converts object module into-> the set of program units (SP) into-> a binary program.

- If link address = load address, loader simply loads binary program into appropriate memory area for execution.

Object Module:

- Object Module(OM) contains all the information necessary to relocate and link the program with other programs.
- OM consist of four components:

  1. Header: Contains translated origin, size and execution start address of P.

  2. Program: Contains machine language program corresponding P.

  3. Relocation Table: (RELOCTAB) describing IRRp (Instruction Requiring Relocation). Eachentry contains field: 'Translated Address' - of address sensitive instruction.

  4. Linking Table: (LINKTAB) contains information concerning public definition and external reference. Has three fields:

  - Symbol: symbolic name
  - Type: PD/EXT i.e public definition or external reference
  - Translated Address: If PD, address of first memory word allocated. If EXT, address of memory word containing reference of symbol.

Example:

- For previously given assembly program, object module contains following information: Consider first statement START 500

  1. Header :Translated Origin: 500, Size:42, Execution Start Add= 500

  2. Machine Language Instructions

  3. Relocation table:

500
538

  4. Linking table:

ALPHA	EXT	518
MAX	EXT	519
TOTAL	PD	541

  LOOP is not here as it is not PD.

DESIGN OF LINKER:

- What influence relocation requirement of a program?

  Addressing Structure of Computer System.
- How to reduce relocation requirement of a program?

  By using segmented addressing structure.
- JUMP avoids use of absolute address, hence instruction is no more address sensitive. Thus, no relocation is needed.

- Effective Operand Address would be calculated considering starting address as 2000 would be

  <CS> + 0196 = 2196 (which is corrected address)

Linking in Segmented Addressing

Sr. No	Statement	Offset
0001	DATA_HERE SEGMENT	
0002	ABC DW 25	0000
0003	B DW ?	0002
.	.	
.	.	
0012	SAMPLE SEGMENT	
0013	SEGMENT CS:SAMPLE,	
	DS:DATA_HERE	
0014	MOV AX, DATA_HERE	0000
0015	MOV DS, AX	0003
0016	JMP A	0005
0017	MOV AL,B	0008
.		
0027	A MOV AX,BX	0196
.		
0043	SAMPLE ENDS	
0044	END	

Linking in Segmented Addressing

- Code written in assembly language for Intel 8088.
- **ASSUME** statements declares segment register CS and DS for memory addressing.
  So, all memory addressing is performed using suitable displacement of their contents.
- Translation time address of A is 0196.

- In statement 16, reference of A due to JMP statement makes displacement of 196 from the content of CS register.
  Hence avoids usage of absolute addressing. Thus, instruction is not address sensitive.
  Displacement ->avoids usage of absolute address->not address sensitive instruction.
- As DS is loaded with execution time address of DATA_HERE, reference to B would be automatically relocated to correct address.
- Thus, Use of segment register reduces relocation requirement but doesn't eliminate it.
- Inter Segment Calls & Jumps are handled in similar way.

- Relocation is more involved in case of intra segment jumps assembled in FAR format.
- Linker computes both:
  - Segment Base Address
  - Offset of External Symbol
- No reduction in linking requirements.

Quiz

1.) _____ is the process binding a program with other programs and libraries, external symbols.

2.) Output of Linker is _____.

3) Relocation corrects the address used in _____ instructions such that the program can execute correctly from the designated area of memory.

4) Relocation may be performed by : a) Linker b.) Loader c.) Both

5) Relocation Factor (RF) of P is defined as_____

Quiz : Solution

1.) _____is the process binding a program with other programs and libraries, external symbols.

Linking

2.) Output of Linker is _____ .

Binary Program

3) Relocation corrects the address used in _____ instructions such that the program can execute correctly from the designated area of memory.

address sensitive instructions

4) Relocation may be performed by a) Linker b.) Loader c.) Both

c) Both

5) Relocation Factor (RF) of P is defined as _____

$relocation\_factor_p = l\_origin_p - t\_origin_p$

Quiz

6.) _____ indicates a symbol defined in a program unit which may be referenced in other program.

7.) The _____ statement lists the public definitions of program unit whereas _____ statement lists the symbols to which external references are made in the program unit.

8.) The four components of OM are : _____

9.) The three fields of LINKTAB are _____, _____, _____

10.) _____ table contains translated address of address sensitive instructions.

11) _____ Reduces relocation requirement of a program

Quiz : Solution

6.) _____ indicates a symbol defined in a program unit which may be referenced in other program. Public Definition

7.) The _____ statement lists the public definitions of program unit whereas _____ statement lists the symbols to which external references are made in the program unit. ENTRY, EXTRN

8.) The four components of OM are : _____ Header, Program, RELOCTAB and LINKTAB

9.) The three fields of LINKTAB are Symbol, type and Translated address

10.) RELOCTAB table contains translated address of address sensitive instructions.

11) Segmented addressing Reduces relocation requirement of a program

Assignments

1) Briefly explain the concept of relocation with an example.
2) Briefly explain the concept of Linking with an example.
3) Describe how the relocation is performed.
4) Elaborate the four components in Object Module.
5) Explain the purpose and structure of LINKTAB and RELOCTAB.
6) Justify :

   Segmented addressing reduces the relocation requirements with valid explanation and example.

Relocation Algorithm:

1. program_linked_origin:=<link origin> from Linker command;
2. For each object module
   (a) t_origin := translated origin of object module; OM_size := size of 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_addr := address in RELOCTAB entry;
      (ii) address_in_work_area := address of work_area + translated_address – t_origin;
      (iii) Add relocation_factor to the operand address in the word with the address 'address_in_work_area'.

   (f) program_linked_origin := program_linked_origin + OM_Size.

Example:

- Let address of work_area = 300.
- While relocating OM, relocation factor = 400.
- For the first RELOCTAB entry, address_in_work_area = 300+500-500=300.
- This word contains the instruction for READ A.
- It is reallocated by adding 400 to operand address in it.
- For the second RELOCTAB entry, address_in_work_area=300+538-500=338.
- The instruction in this word is similarly relocated by adding 400 to the operand address in it.

Linking Requirements:

Linker processes all object modules being linked & builds a table of all public definition & load time address.

Name Table (NTAB)

- Symbol: Symbol Name for external reference or object module.
- Linked Address: For public definition contains linked_address.
  For object module contains linked_origin.

- Most information in NTAB is derived from LINKTAB entries with type=PD.

Algorithm: Program Linking

1. program_linked_origin := <link origin> from linker command.
2. For each object module

(a) t_origin := translated origin of 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 LINKTAB of the object module.

(e) For each LINKTAB entry with type = PD ,
name:= symbol;
linked_address := translated_address + relocation_factor;
Enter(name,linked_address) in NTAB.

(f) Enter (object module name, program_linked_origin) in NTAB.

(g) program_linked_origin := program_linked_origin + OM_size;

Algorithm: Program Linking

3. For each object module mentioned in Linker Command,

(a) t_origin := translated origin of the object module.

program_linked_origin = linked_address from NTAB

(a) 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 and copy its linked address. Add the linked address

to the operand address in the word with the address address_in_work_area.

Example: Program Linking

- Linker Command be :

  > Linker 900 , P , Q

  linked_origin=900, NTAB contains information shown in fig:

  e.g: work_area = 300.

- When LINKTAB entry of alpha is processed during linking, address_in_work_area := 300 + 900 - 900 + 518 – 500.

  = 318.

Linked address of ALPHA is 973, is copied from NTAB entry of ALPHA and added to word in address 318.

300+900-900+519-500

= 319

NTAB : Name Table Global

Symbol	Linked Address
P	900
TOTAL	941
Q	942
ALPHA	973

SELF RELOCATING PROGRAMS

- Types of programs:
  - Non Re-locatable Programs
  - Re-locatable Programs
  - Self Re-locating Programs

1. Non Re-locatable Programs:

  - It cannot be executed in any memory area other than area starting on its translated origin.
  - Has address sensitive programs.
  - Lack information concerning address sensitive instruction
  - Eg. Hand coded machine language program.

SELF RELOCATING PROGRAMS

2. Re-locatable Programs:

- It can be processed to relocate it into desired data area of memory.
- Eg: Object Module.

3. Self Relocating Programs:

- It can perform relocation of its own address sensitive instruction.
- Consist of:
  - A table of Information (for address sensitive prg)
  - Relocating logic
- Self re-locating program can execute in any area of memory.
- As load address is different for each execution, good for time sharing systems.

A LINKER FOR MS DOS

A linker for Intel 8088/80/x86 processors which resembles LINK of MS DOS.

Object Module Format:

- An Intel 8088 object module is a sequence of object records.
- Each object record describes specific aspect of program in object module.
- There are 5 categories comprising total 14 types of object records.
- Five Categories:
  1. Binary image(ie code generated by translator)
  2. External reference
  3. Public definition
  4. Debugging information (e.g. Line numbers)
  5. Miscellaneous information (e.g. comments)

- Each object record contains variable length information and may refer to contents of previous object records.

Types of Object Records (order is important)

THEADR	80	Translator header record (First record)
LNAMES	96	List of names record
SEGDEF	98	Segment definition record
EXTDEF	8C	External names definition record.
PUBDEF	90	Public names definition record
LEDATA	A0	Enumerated data (binary image)
FIXUPP	9C	Fix up record
MODEND	8A	Module end record

Object Record Formats

THEADR record: Translator Header Record

80H	Length	T-module name	Check Sum

- Typically derived by the translator from the source file name.
- This file name is used by the linker to report errors.
- An assembly programmer can specify the module name in the NAME directive.

Object Record Formats:

LNAMES: record lists the names for use by SEGDEF records.

96H	Length	Name List	Check Sum

SEGDEF: Designates a segment name

98H	Length	Attributes (1-4)	Segment Length (2)	Name Index (1)	Check Sum

- Uses index into in to the list
- Attribute field indicates whether segment is re-locatable or absolute.
- Also indicates the manner in which it is combined with other segments.
- Also indicates alignment requirement of its base address.
- Attribute field also contains origin specification for absolute segment.

Object Record Formats:

EXTDEF record : The EXTDEF record contains a list of external references used by module

8CH	Length	External Reference List	Check Sum

PUBDEF record : Contains list of public names declared.

90H	Length	base (2-4)	Name	offset (2)	…	Check Sum

- Base specification identifies the segment.
- (name, offset) pair defines one public name.

Object Record Formats:

FIXUPP record : Contains information necessary for performing Relocation and Linking

9CH	Length	Locat (1)	Fix dat (2)	Frame datam (1)	Target datum(1)	Target offset(2)	…	Check Sum

- Designates external symbol name
- Uses index to the list

- Locat contains offset of fix up location in previous LEDATA.
- Frame datum refers to SEGDEF record.
- Target datum and target offset specify relocation and linking information.

Target Datum contains segment index or an external index.

- Target offset contains offset from name indicated in target datum.
- Fix data field indicates manner in which target datum and fix datum fields are to be interpreted.

Loc Code	Meaning:
0	Low order byte to be fixed
1	Offset is to be fixed
2	Segment is to be fixed
3	Pointer (segment offset) to be fixed.

Codes in Fixdat field

Fix dat	Meaning:
0	Segment index & displacement
2	External index &target displcmt
4	Segment index (without offset)
6	External index (without offset)

Object Record Formats:

MODEND record : Indicates the end of module

8AH	Length	Type (1)	Start addr (5)	Check sum

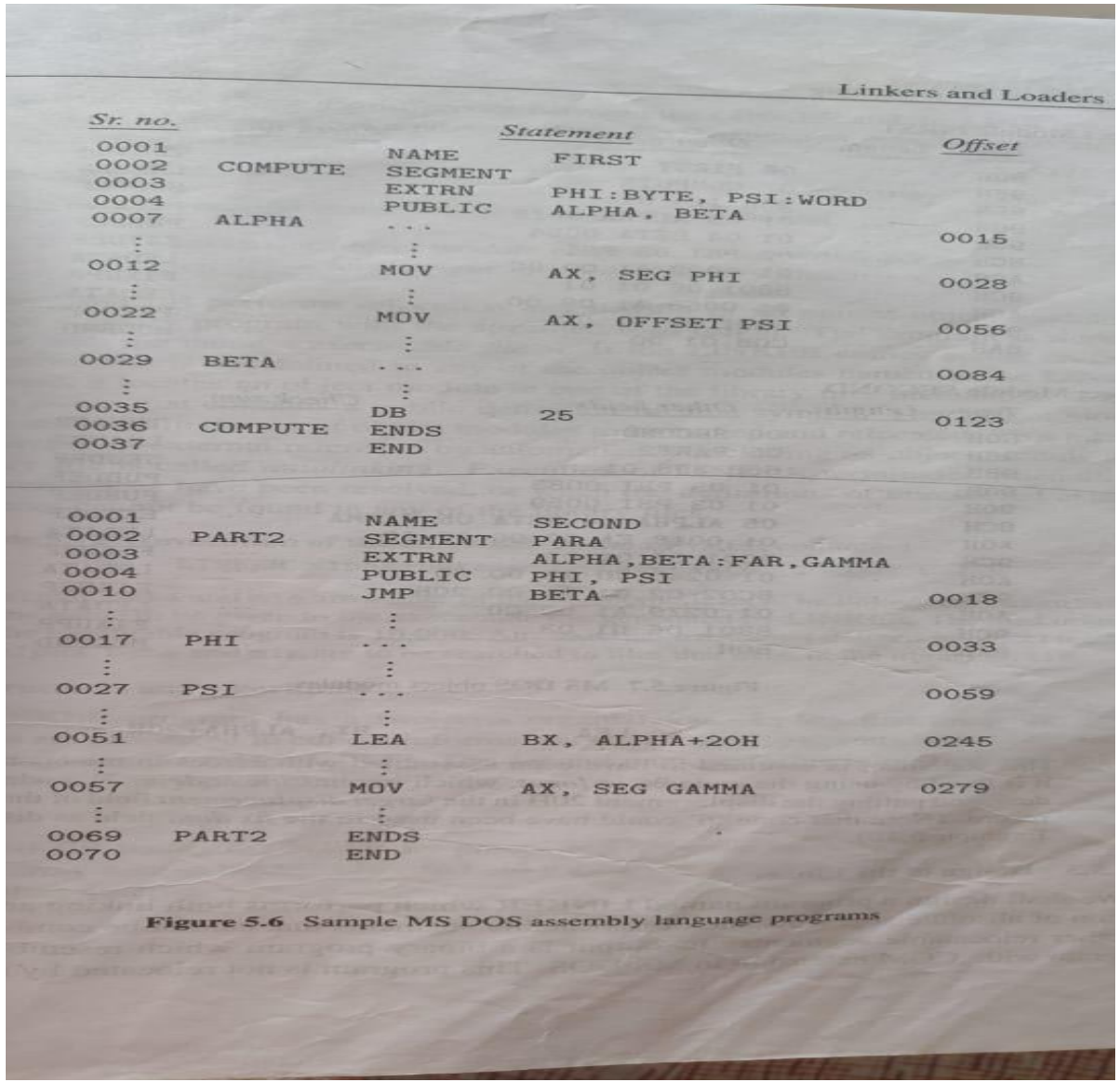
- Type field indicates whether it is the main program.
- Has 2 components: (a) segment, (b) offset

LEDATA records: Contains binary image of code generated by language translator.

- Segment index identifies the segment to which the code belongs.
- Offset specifies the location of the code within the segment.

AOH	Length	Segment Index (1-2)	Data Offset (2)	data	Check sum

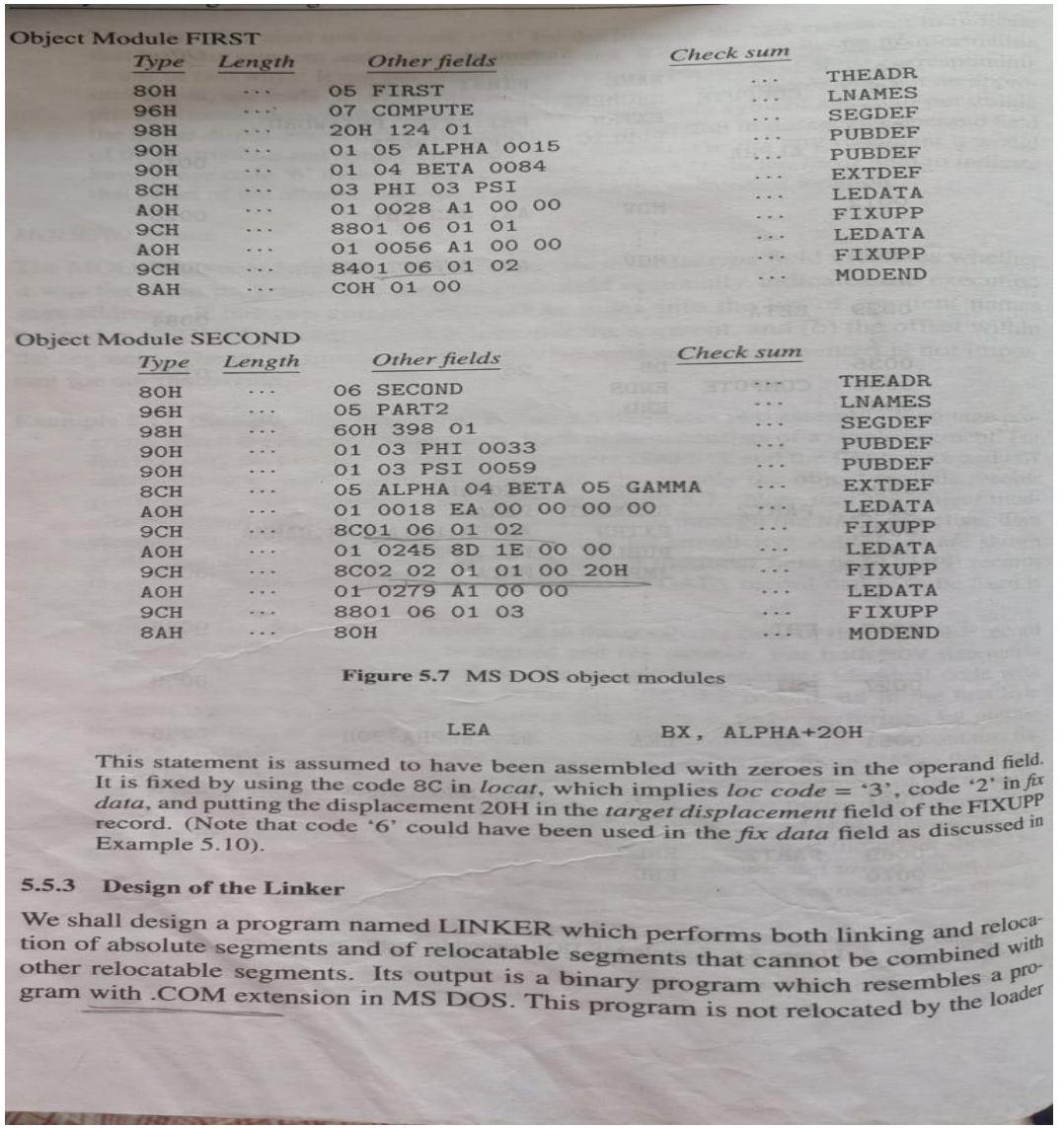
Example MS DOS Assembly Program and Object Module



Sr. no.		Statement		Offset
0001		NAME	FIRST	
0002	COMPUTE	SEGMENT		
0003		EXTRN	PHI:BYTE, PSI:WORD	
0004		PUBLIC	ALPHA, BETA	
0007	ALPHA	...		
⋮		⋮		0015
0012		MOV	AX, SEG PHI	
⋮		⋮		0028
0022		MOV	AX, OFFSET PSI	
⋮		⋮		0056
0029	BETA	...		
⋮		⋮		0084
0035		DB	25	
0036	COMPUTE	ENDS		0123
0037		END		

0001		NAME	SECOND	
0002	PART2	SEGMENT	PARA	
0003		EXTRN	ALPHA,BETA:FAR,GAMMA	
0004		PUBLIC	PHI, PSI	
0010		JMP	BETA	0018
⋮		⋮		
0017	PHI	...		0033
⋮		⋮		
0027	PSI	...		0059
⋮		⋮		
0051		LEA	BX, ALPHA+20H	0245
⋮		⋮		
0057		MOV	AX, SEG GAMMA	0279
⋮		⋮		
0069	PART2	ENDS		
0070		END		

Figure 5.6 Sample MS DOS assembly language programs



Object Module FIRST

Type	Length	Other fields	Check sum	
80H	...	05 FIRST	...	THEADR
96H	...	07 COMPUTE	...	LNAMES
98H	...	20H 124 01	...	SEGDEF
90H	...	01 05 ALPHA 0015	...	PUBDEF
90H	...	01 04 BETA 0084	...	PUBDEF
8CH	...	03 PHI 03 PSI	...	EXTDEF
A0H	...	01 0028 A1 00 00	...	LEDATA
9CH	...	8801 06 01 01	...	FIXUPP
A0H	...	01 0056 A1 00 00	...	LEDATA
9CH	...	8401 06 01 02	...	FIXUPP
8AH	...	C0H 01 00	...	MODEND

Object Module SECOND

Type	Length	Other fields	Check sum	
80H	...	06 SECOND	...	THEADR
96H	...	05 PART2	...	LNAMES
98H	...	60H 398 01	...	SEGDEF
90H	...	01 03 PHI 0033	...	PUBDEF
90H	...	01 03 PSI 0059	...	PUBDEF
8CH	...	05 ALPHA 04 BETA 05 GAMMA	...	EXTDEF
A0H	...	01 0018 EA 00 00 00 00	...	LEDATA
9CH	...	8C01 06 01 02	...	FIXUPP
A0H	...	01 0245 8D 1E 00 00	...	LEDATA
9CH	...	8C02 02 01 01 00 20H	...	FIXUPP
A0H	...	01 0279 A1 00 00	...	LEDATA
9CH	...	8801 06 01 03	...	FIXUPP
8AH	...	80H	...	MODEND

Figure 5.7 MS DOS object modules

```
LEA          BX, ALPHA+20H
```

This statement is assumed to have been assembled with zeroes in the operand field. It is fixed by using the code 8C in *locat*, which implies *loc code* = '3', code '2' in *fix data*, and putting the displacement 20H in the *target displacement* field of the FIXUPP record. (Note that code '6' could have been used in the *fix data* field as discussed in Example 5.10).

5.5.3 Design of the Linker

We shall design a program named LINKER which performs both linking and relocation of absolute segments and of relocatable segments that cannot be combined with other relocatable segments. Its output is a binary program which resembles a program with .COM extension in MS DOS. This program is not relocated by the loader

Source : " Systems programming ", D M Dhamdhere. McGraw-Hill Education (India) Pvt Limited

Algorithm for First Pass of LINKER

1. program_linked_origin := <load origin>;
2. Repeat step 3 for each object module to be linked.
3. Select an object module and process its object records.

(a) If an LNAMES record, enter the names in NAMELIST.

(b) If a SEGDEF record

(i) i := name index; segment_name := NAMELIST[i];
segment_addr := start address in attributes;

(ii) If an absolute segment, enter (segment_name, segment_addr) in NTAB.

(iii) If the segment is re-locatable and cannot be combined with other segments

- Align the address containing in program_linked_origin on the next word or paragraph as indicated in attribute list.
- Enter (segment_name,program_linked_origin) in NTAB.
- program_linked_origin := program_linked_ origin + segment length;

Algorithm for First Pass of LINKER Contd.

(c) For each PUBDEF record

(i) i := base; segment_name := NAMELIST[i];

(ii)segment_addr := load address of segment_name in NTAB;

(iii) sym_addr := segment_addr + offset;

(iv) Enter (symbol,sym_addr) in NTAB.

In first pass, linker only processes the object records relevant for building NTAB.

Second pass performs relocation and linking.

Algorithm for Second Pass of LINKER

1. list_of_object_module := object modules named in LINKER command;
2. Repeat step 3 until list_of_object_modules is empty.
3. Select an object module and process its object records.
   (a) if an LNAMES record
   Enter the names in NAMELIST.
   (b) if a SEGDEF record
   i := name index; segment_name := NAMELIST[i];
   (c) if an EXTDEF record
   (i) external_name := name from EXTDEF record;
   (ii) if external_name is not found in NTAB, then
   - Locate object module in library which contains external_name as a segment or public definition.
   - Add name of object module to list_of_object_module.
   - Perform first pass of LINKER, for new object module.

   (iii) Enter (external_name, load address from NTAB) in EXTTAB.

Algorithm for Second Pass of LINKER (Contd.)

(d) if an LEDATA record

(i) i:= segment index; d := data offset;

(ii) program_linked_origin := SEGTAB[i].linked address;

(iii) address_in_work_area := address of work_area + program_linked_origin - <load origin> + d ;

(iv) move data from LEDATA into the memory area starting at the address address_in_work_area.

(e) if a FIXUPP record, for each FIXUPP specification

(i) f := offset from locat field;

(ii) fix_up_address := address of work_area + program_linked_origin - <load origin> + f;

(iii) Perform required fix up using a load address from SEGTAB or EXTTAB and the value of code in locat and fix dat.

(f) if MODEND record

if start address is specified, compute the corresponding load address and record it in the executable file being generated.

Linking for OVERLAYS:

Definition: (Overlay) An overlay is a part of a program (or software package) which has the same load origin as some other parts of the program.

Overlays are used to reduce the main memory requirement of a program.

Overlay structured programs consist of:

1. A permanently resident portion, called the root.
2. A set of overlays.

Execution of overlay structured program:

1. root is loaded in the memory
2. other overlays are loaded as and when needed.
3. loading of an overlay overwrites a previously loaded overlay with the same load origin.

Linking for OVERLAYS:

This reduces the memory requirement of the program.

Hence, facilitate execution of those programs who's size exceeds memory size.

How to design overlay structure?

- By identifying mutually exclusive module.
- Modules that do not call each other.
- These modules do not need to reside simultaneously in the memory.
- Hence keep those modules in different overlays with same load origin.

Linking for OVERLAYS:

- Execution of an overlay structured program:
  - **Overlay manager module** is included in executable file which is responsible for overlay load as and when required.
  - **Interrupt producing instruction** replaces all cross overlay boundary calls.
- Changes in Linker Algorithm?
  - Assignment of load address to segments after execution of root.
  - Mutually exclusive modules are assigned same program_load_origin.
  - Also algorithm has to identify inter-overlay call and determine destination overlay.

Quiz

1.) Linker processes all object modules being linked & builds _____ table.

2.) Information in NTAB is derived from LINKTAB entries with type _____ .

3.) LNAMES record lists the names of _____ .

4.)_____ object records contain information necessary for performing Relocation and Linking.

5.) _____ program can perform relocation of its own address sensitive instruction.

6.) _____structure is built by identifying mutually exclusive modules.

Quiz : Solution

1.) Linker processes all object modules being linked & builds Name Table (NTAB) table.

2.) Information in NTAB is derived from LINKTAB entries with type __ PD.

3.) LNAMES record lists the names of _____ Segments

4.)_____ object records contain information necessary for performing Relocation and Linking. FIXUPP

5.) _____ program can perform relocation of its own address sensitive instruction. Self Relocating Program

6.) _____structure is built by identifying mutually exclusive modules. Overlay

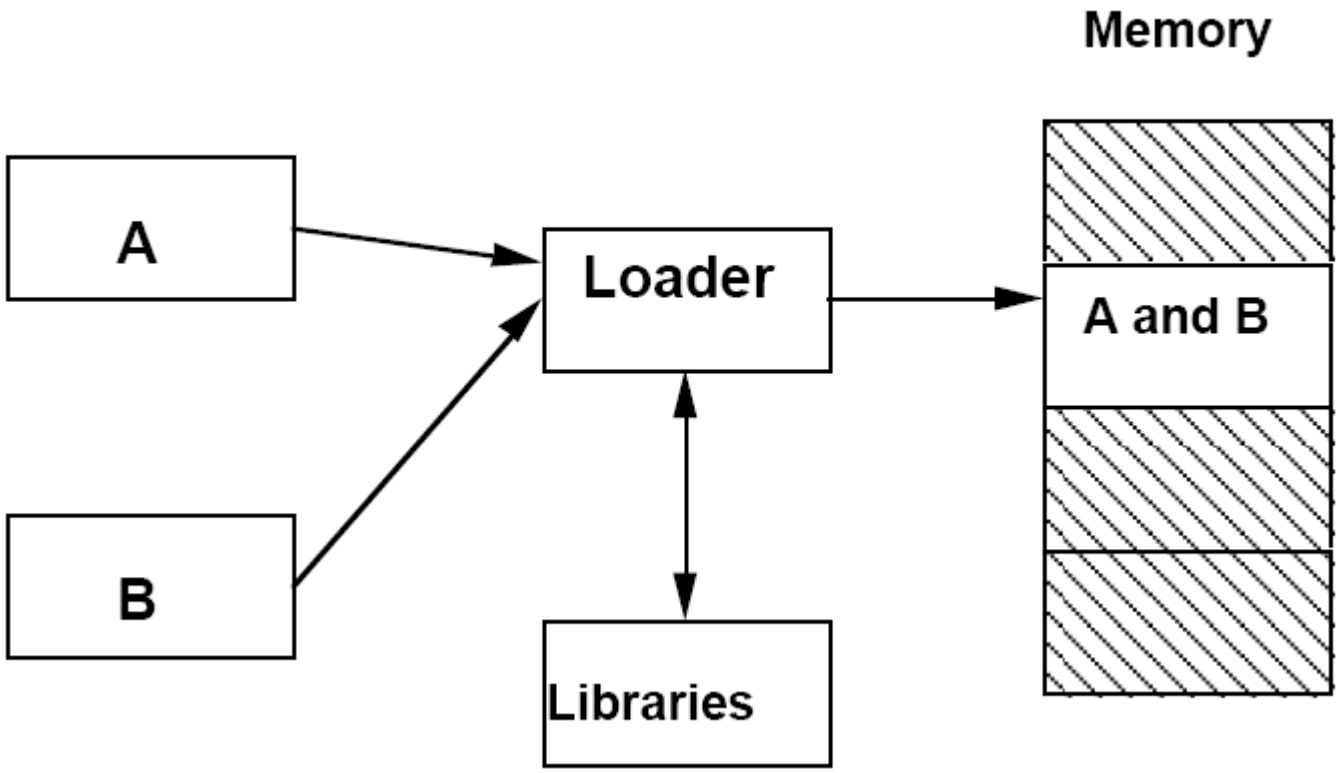
Assignments

1) Write an algorithm for Relocation.
2) Explain the purpose of NTAB in Linking? How is it built?
3) Write an algorithm to perform Linking on a set of Oms.
4) What is Overlay structure? What is the advantage of it?.
5) Explain the format and purpose of following Object record formats:
   a.) FIXUPP b) LEDATA c) EXTDEF d) SEGDEF
6) Write an algorithm for Pass 1 and Pass 2 MSDOS Linker.

7) Explain the purpose behind building EXTTAB. Why is it required?

Loaders

A loader is a system program that performs the loading of object program into memory. Loaders also support relocation & linking. Others have a separate linker and loader,



Loader : Basic Functions

Allocation: allocate space in memory for the programs

Linking: Resolve symbolic references between object files

- combines two or more separate object programs
- supplies the information needed to allow references between them

Relocation: Adjust all address dependent locations, such as address constants, to correspond to the allocated space

- modifies the object program so that it can be loaded at an address different from the location originally specified

Loading: Physically place the machine instructions and data into memory

Loaders

Two Types:

1. Absolute Loader:
   - Can load only programs with load origin = linked origin.
   - This is inconvenient when load address differ from the one specified for execution.
2. Relocating loader:
   - Performs relocation while loading a program for execution.
   - Permits program to be executed in different parts of memory.

MS DOS support 2 object program forms:

1. .COM:
   - Contains non re-locatable object program.
   - Absolute loaders are invoked.
2. .EXE:
   - Contains re-locatable object program.
   - Relocating loaders are invoked.

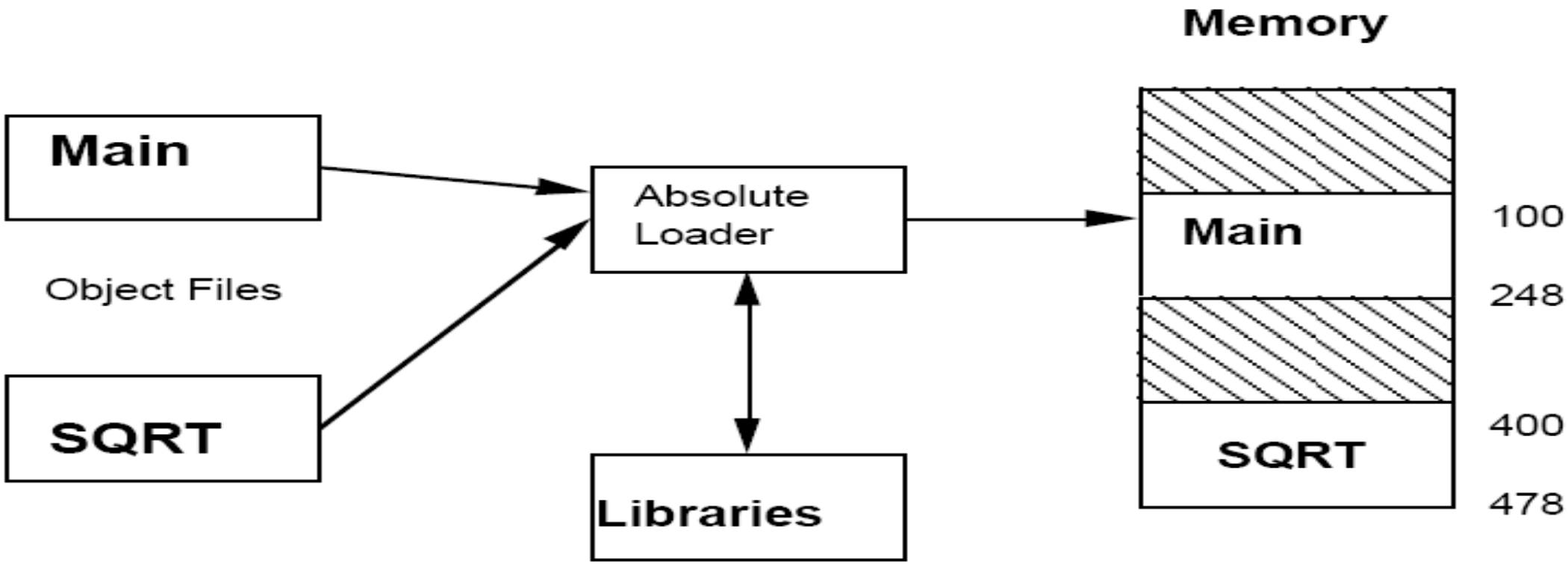
At the end of loading, execution starts.

Loader Schemes

Compile and Go

- The assembler runs in one part of memory
- Place the assembled machine instructions and data, as they are assembled, directly into their assigned memory locations
- When the assembly is completed, the assembler causes a transfer to the starting instruction of the program.

Absolute Loader



Disadvantages

A portion of memory is wasted because the memory occupied by the assembler is unavailable to the object program.

It is necessary to re-translate (assemble) the user's program file every time it is run.

It is very difficult to handle multiple segments, especially if the source programs are in different.

Disadvantages

If changes were made to MAIN that increased its length to more than 300 bytes

- the end of MAIN (at 100 + 300 = 400) would overlap the start of SQRT (at 400)
- It would then be necessary to assign SQRT to a new location
  - changing its START and re-assembling it?!

Furthermore, it would also be necessarily to modify all other subroutines that referred to the address of SQRT.

Disadvantages of Absolute Loaders

Actual load address must be specified

The programmer must be careful not to assign two subroutines to the same or overlapping locations

Difficult to use subroutine libraries (scientific and mathematical) efficiently

- important to be able to select and load exactly those routines that are needed

Disadvantages of Absolute Loaders

Allocation - by programmer

Linking - by programmer

Relocation - None required-loaded where assembler assigned

Loading - by loader

A Simple Bootstrap Loader

Automatically executed when the computer is first turned on

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

General Loader Scheme

- Linking
- Relocation
- Loading

Dynamic Loading

If the total amount of memory required by all subroutines exceeds the amount available

The module loader loads the only the procedures as they are needed.

- Allocating an overlay structure

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

Example

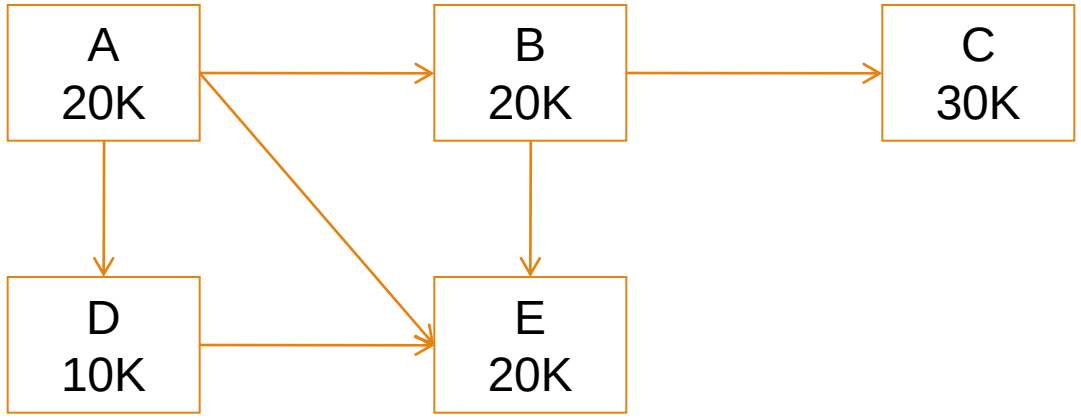
Suppose a program consisting of five subprograms (A{20k},B{20k}, C{30k}, D{10k}, and E{20k}) that require 100K bytes of core.

- Subprogram A only calls B, D and E;
- subprogram B only calls C and E;
- subprogram D only calls E
- subprogram C and E do not call any other routines

Note that procedures B and D are never in used the same time; neither are C and E.

Longest Path Overlay Structure

100k Vs. 70k needed



Dynamic Linking

The loading and linking of external references are postponed until execution time.

The loader loads only the main program

If the main program should

- execute a branch to an external address,
- reference an external variable

The loader is called

- Only then has the segment containing the external reference loaded.

Quiz (Practice)

1) A linker is given object module for a set of programs that were compiled separately. What information need not be included in an object module?_____.
2) The process of assigning load addresses to the various parts of the program and adjusting the code and data in the program to reflect the assigned addresses is called _____.
3) Object modules generated by assembler that contains unresolved external references are resolved for two or more object module by a/an_____.
4) Object code is the output of _____.
5) In an absolute loading scheme, which loader function is accomplished by a loader _____.
6) A system program that combines the separately compiled modules of a program into a form suitable for execution ?_____.

Quiz (Practice)

7) Load address for the first word of the program is called?_____.

8) Consider a program with a linked origin of 5000. Let the memory area allocated to it have the start address of 70000. Which among the following will be the value to be loaded in relocation register?_____

9) Resolution of externally defined symbols is performed by_____.

10)Static memory allocation is typically performed during _____.

11)Dynamic memory allocation is typically performed during _____ .

12)Linking is process of binding_____.

13)If load origin is not equal to linked origin then relocation is performed by_____.

14)If linked origin is not equal to translated address then relocation is performed by_____.

15)Binder performs the functions of_____.

Assignments

1. Explain Loaders. What are functions of loader?
2. List out the function performed by relocating loaders?.
3. What is an absolute loader? State its advantages.
4. Discuss the functions and design of an absolute loader.
5. What is dynamic linking? Explain. What is the advantage of Dynamic Linking?
6. Explain with neat block diagram the role of loader & linker.
7. What are Dynamic Link Libraries? State the 3 advantages of DLL.
8. Explain with a neat diagram the concept of Overlay.

Assignments

9. What steps are involved in execution of a program? Explain translated, linked and load time addresses.
10. Discuss linking for overlays in detail.
11. Discuss about ‘self-relocating programs’.
12. Compare Linker and Loader.
13. What is meant by bootstrap loader?
14. Define automatic library search.
15. Describe the structure and purpose of following tables in Linker.
    I) RELOCTAB II) LINKTAB

Reference

“ Systems programming “, D M Dhamdhere. McGraw-Hill Education (India) Pvt Limited,

THANK YOU