Chapter 5:
Linkers

Mrs. Sunita M Dol (Aher),
Assistant Professor,
Computer Science and Engineering Department,
Walchand Institute of Technology, Solapur, Maharashtra

5. Linkers

- Relocation and Linking Concepts
- Design of a Linker
- Self-Relocating Programs
- Linking for Overlays

5. Linkers

- Relocation and Linking Concepts
- Design of a Linker
- Self-Relocating Programs
- Linking for Overlays

Relocation and linking concepts

- Execution of a program written in a language L involves following steps:
  - Translation of the program by translator of language L
  - Linking of the program with other program needed for its execution by a linker
  - Relocation of the program to execute from the specific memory area allocated to it by a linker
  - Loading of the program in the memory for the purpose of execution by a loader

Relocation and linking concepts

- Schematic of program execution

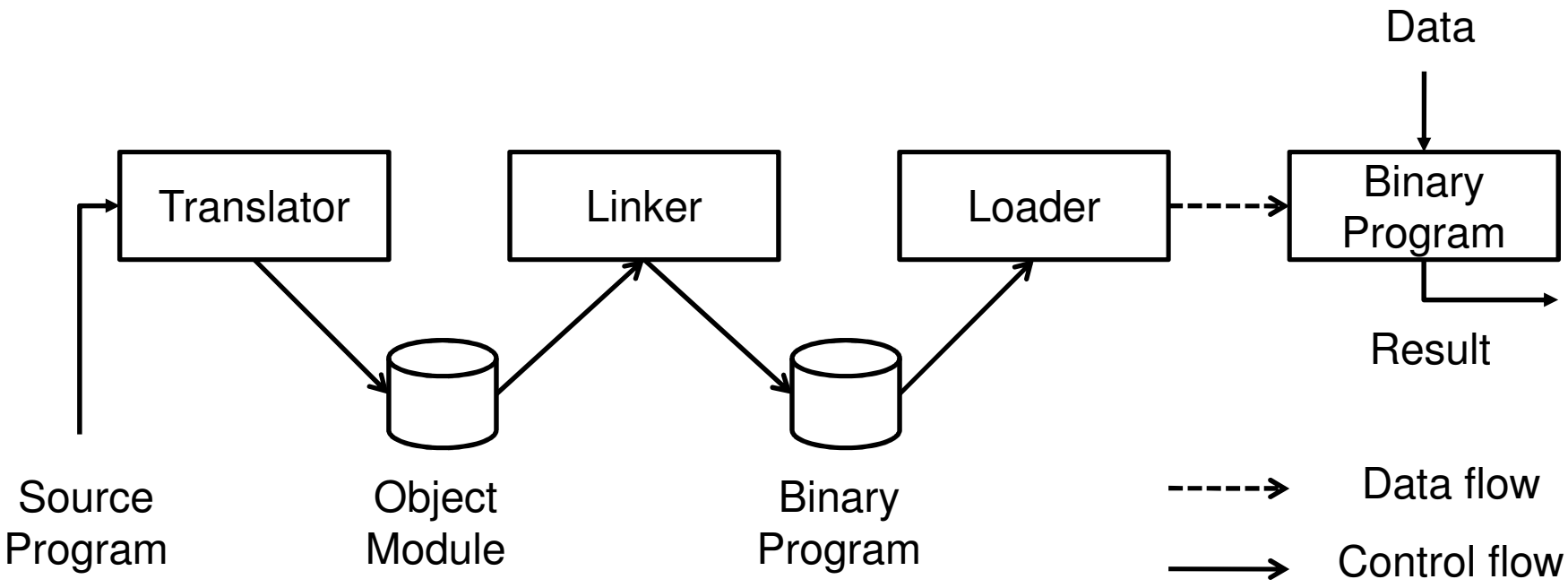


Figure: Schematic of program execution

Relocation and linking concepts

- Translated, linked and load time addresses
  - **Translation time (or translated) address**: address assigned by translator
  - **Linked address**: address assigned by the linker
  - **Load time (or load) address** : address assigned by the loader.

Relocation and linking concepts

- Translated, linked and load time addresses
  - **Translated origin**: address of the origin assumed by translator which is specified by the programmer in an ORIGIN statement.
  - **Linked origin**: address of the origin assigned by the linker while producing a binary program.
  - **Load time (or load) address** : address of the origin assigned by the loader while loading the program for execution.

Relocation and linking concepts

- Translated, linked and load time addresses example

	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		541)	
	END			

✓ Translated origin of the program = 500
✓Translation time address of LOOP = 501
✓Suppose load time origin = 900
✓Load time address of LOOP = 901

Relocation and linking concepts

- Program Relocation
  - Let AA be the set of absolute addresses- instruction or data addresses used in the instruction of a program P.
  - **Address sensitive program** : AA ≠ ϕ implies that program P assumes its instructions and data occupy memory words with specific addresses.
    - **An address sensitive instruction** : an instruction which uses an address $a_i \in AA$.
    - **An address constants** : a data word which contains an address $a_i \in AA$.

Relocation and linking concepts

- Program Relocation
  - Program relocation is the process of modifying the addresses used in the address sensitive instructions of a program such that the program can execute correctly from the designated area of memory.
  - If linked origin ≠ translated origin, relocation must be performed by the linker.
  - If load origin ≠ linked origin, relocation must be performed by the loader.
  - If load origin = linked origin, such loaders are called absolute loader.

Relocation and linking concepts

- Program Relocation

	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		541)	
	END			

- ✓ Translated origin of the program = 500
- ✓Translation time address of symbol A = 540
- ✓Suppose link origin = 900
- ✓Link time address of symbol A = 901

Relocation and linking concepts

- Performing Relocation
  - t_origin_P translated origin of program P
  - l_orgin_P linked origin of program P
  - t_{symb} translation time address
  - l_{symb} link time address
  - d_{symb} offset of symbol

Relocation and linking concepts

- Performing Relocation

  $relocation\_factor_P = l\_orgin_P - t\_origin_P$ ---------(a)

  $t_{symb} = t\_origin_P + d_{symb}$

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

  Using (a),

  $l_{symb} = t\_origin_P + relocation\_factor_P + d_{symb}$

  $= t\_origin_P + d_{symb} + relocation\_factor_P$

  $= t_{symb} + relocation\_factor_P$

Relocation and linking concepts

- Program Relocation

	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		541)	
	END			

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

Relocation and linking concepts

- Linking
  - Linking is the process of binding an external reference to the correct link time address.
  - Program consist of
    - **Public definition** – a symbol may be referenced in other program unit. The ENTRY statement list the public definition of a program unit.
    - **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.

Relocation and linking concepts

- Linking

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		541)	
	END			

Relocation and linking concepts

- Linking

Program Q

	Statement		Address	Code
	START	200		
	ENTRY	ALPHA		

ALPHA	DS	25	231)	+ 00 0 025
	END			

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

Relocation and linking concepts

- Binary Program
  - A binary program is a machine language program comprising a set of program units SP such that $\forall$ SP $\in$ P_i.
    - P_i has been relocated to the memory area starting at its link origin and
    - Linking has been performed for each external references in P_i.
  - 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>]

Relocation and linking concepts

- Object Module
  - Object module of a program contains all information necessary to relocate and link the program with other program.
  - It consist of 4 components:
    1. **Header** : contains
       - Translated origin
       - Size
       - Execution start address of program P
    2. **Program** : contains machine language program corresponding to program P.

Relocation and linking concepts

- Object Module

3. **Relocation Table (RELOCTAB)** : describes IRRp (Set of instruction requiring relocation). It contains
   - Translated address of address sensitive instruction.

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

   This table contains
   - Symbol : Symbolic name.
   - Type : PD/EXT
   - Translated Address
     - ✓ For public definition, this is the address of the first memory word allocated to the symbol.
     - ✓ For external reference, it is address of the memory word which is required to contain the address of the symbol.

Relocation and linking concepts

- Object Module

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		541)	
	END			

Relocation and linking concepts

- Object Module
  - for program P
    1. Header : translated origin = 500, size = 42, execution start address = 500
    2. Machine language instruction shown in figure.
    3. Relocation table

1. Linking table

Translated address
500
538

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

Design of a Linkers

- Relocation and Linking Concepts
- Design of a Linker
- Self-Relocating Programs
- Linking for Overlays

Design of a Linkers

- Design of a Linker

Algorithm (Program Relocation)

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 the object module.

Design of a Linkers

- Design of a Linker

Algorithm (Program Relocation)

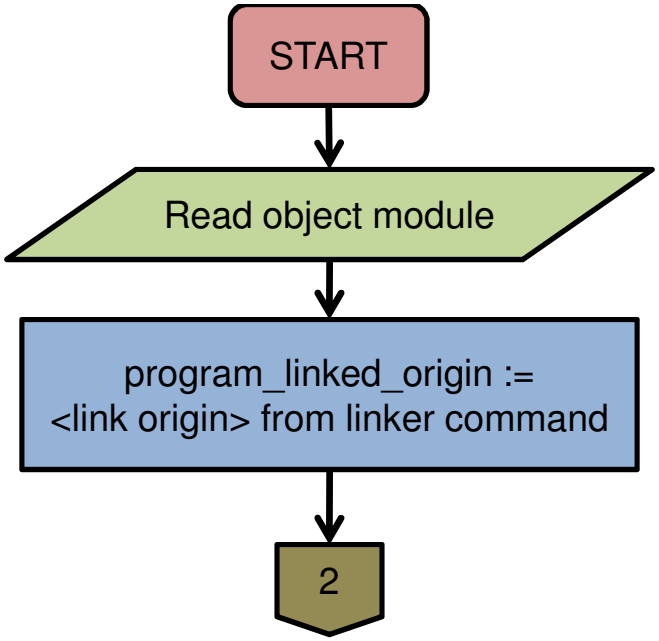
e) For each entry in RELOCTAB

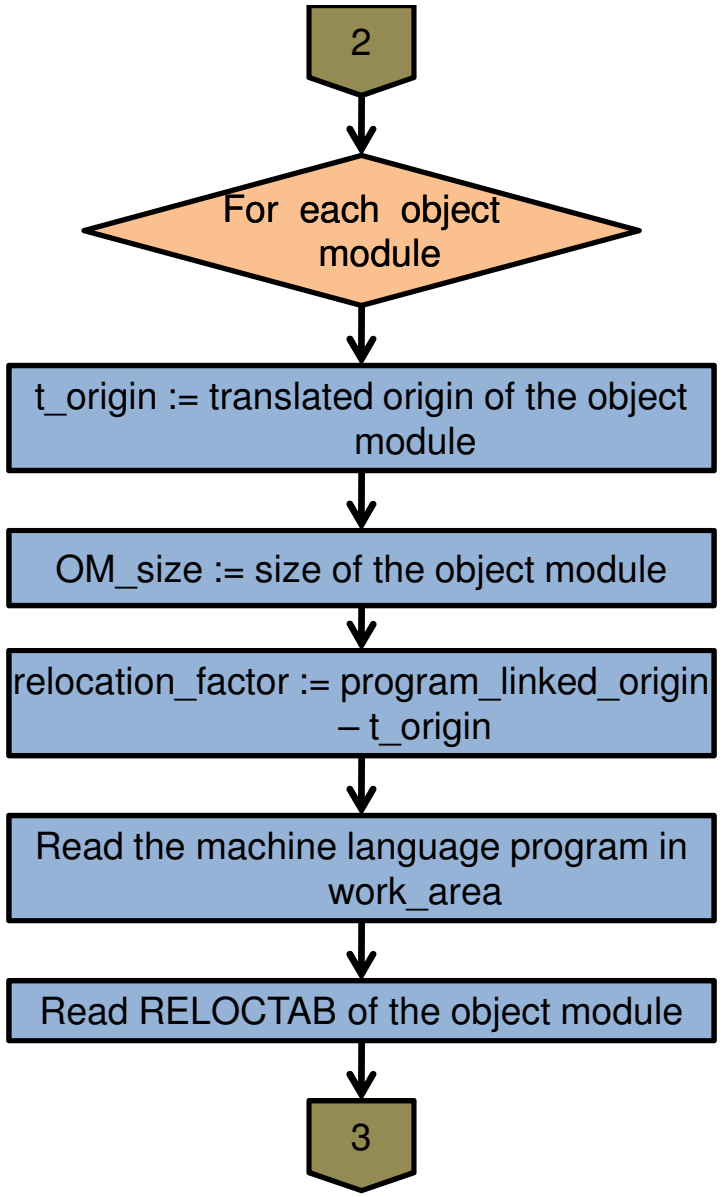
i) translated_address := address in the RELOCTAB entry;

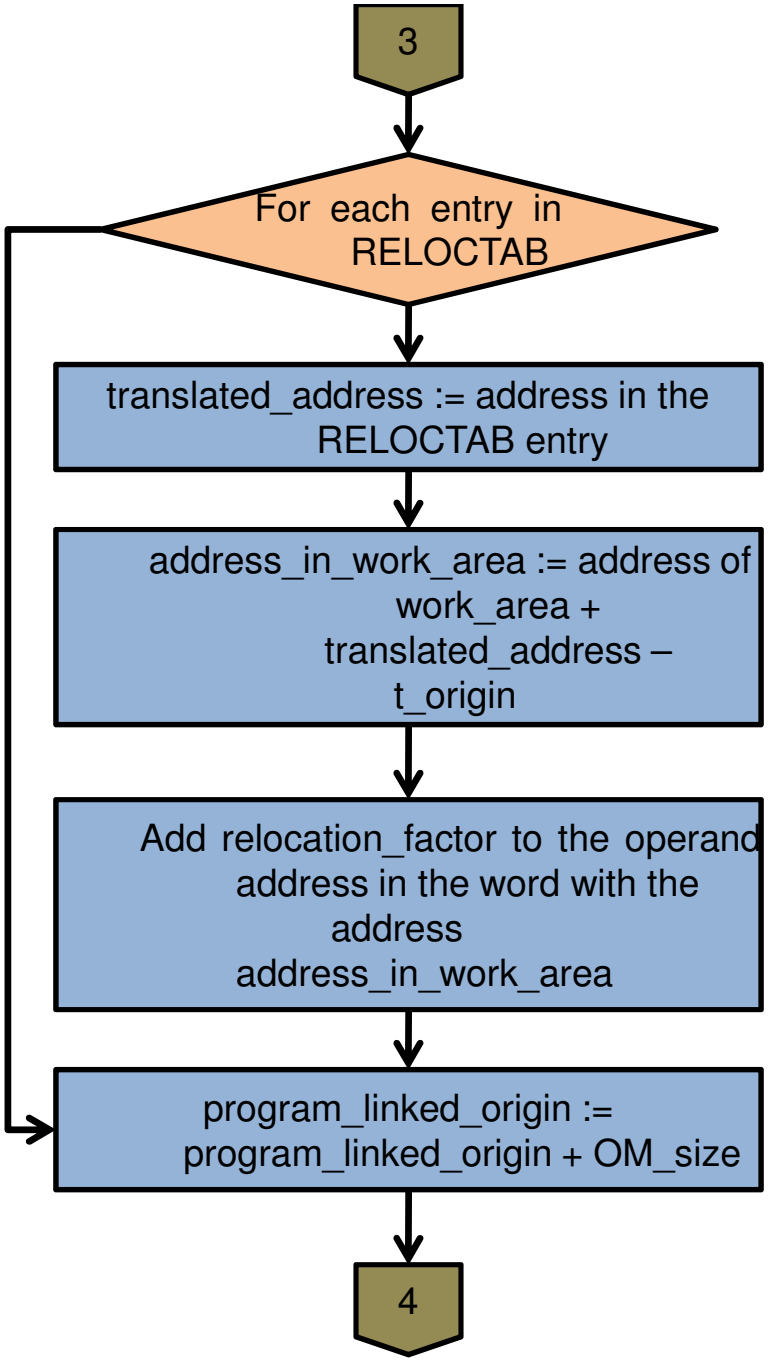
ii) address_in_work_area := address of work_area + translated_address – t_origin;

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;


START
Read object module
program_linked_origin :=
<link origin> from linker command
2


2
For each object module
t_origin := translated origin of the object module
OM_size := size of the object module
relocation_factor := program_linked_origin – t_origin
Read the machine language program in work_area
Read RELOCTAB of the object module
3


3
For each entry in RELOCTAB
translated_address := address in the RELOCTAB entry
address_in_work_area := address of work_area + translated_address – t_origin
Add relocation_factor to the operand address in the word with the address address_in_work_area
program_linked_origin := program_linked_origin + OM_size
4

Design of a Linkers

- Design of Linker

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		541)	
	END			

Design of a Linkers

- Design of Linker
  - let the address of work_area be 300
  - relocation factor = 400
  - For first RELOCTAB entry,

    address_in_work_area = 300 + 500 – 500 = 300
  - For second RELOCTAB entry

    address_in_work_area = 300 + 538 – 500 = 338

Design of a Linkers

- Linking Requirements

Algorithm (Program Linking)

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 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.

Design of a Linkers

- Linking Requirements

Algorithm (Program Linking)

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

g) program_linked_origin := program_lined_origin + OM_size;

3. For each object module

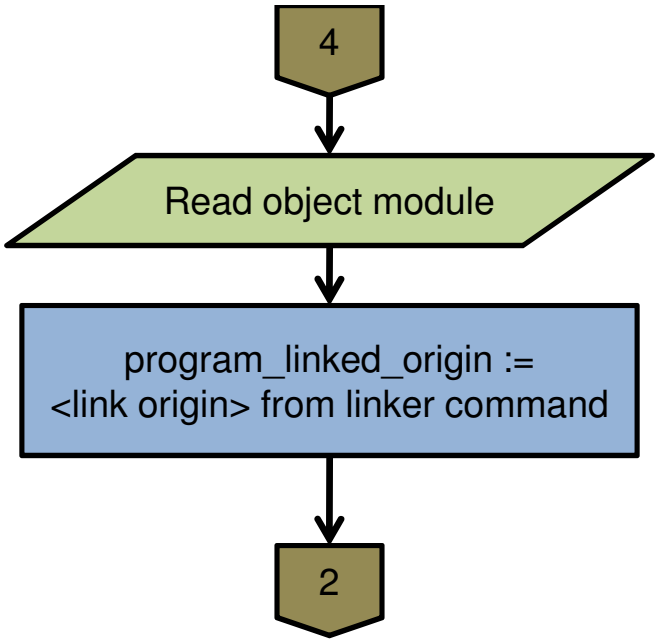
a) t_origin := translated origin of the object module;

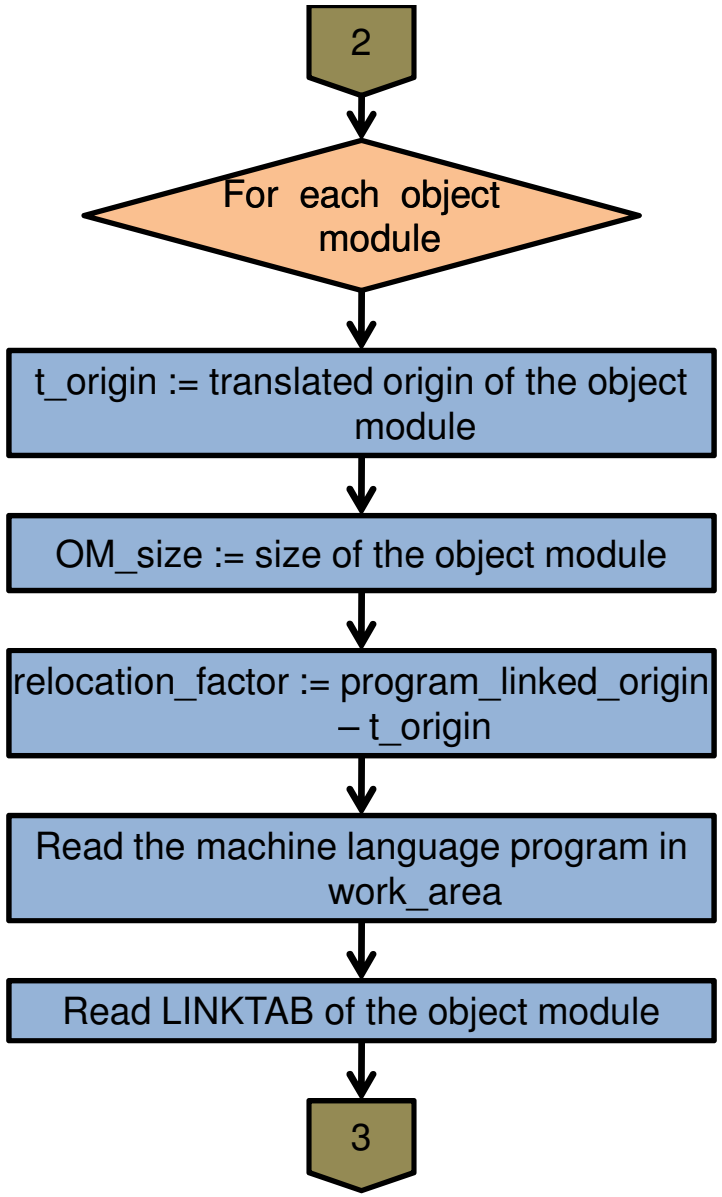
program_linked_origin := load_address from NTAB;

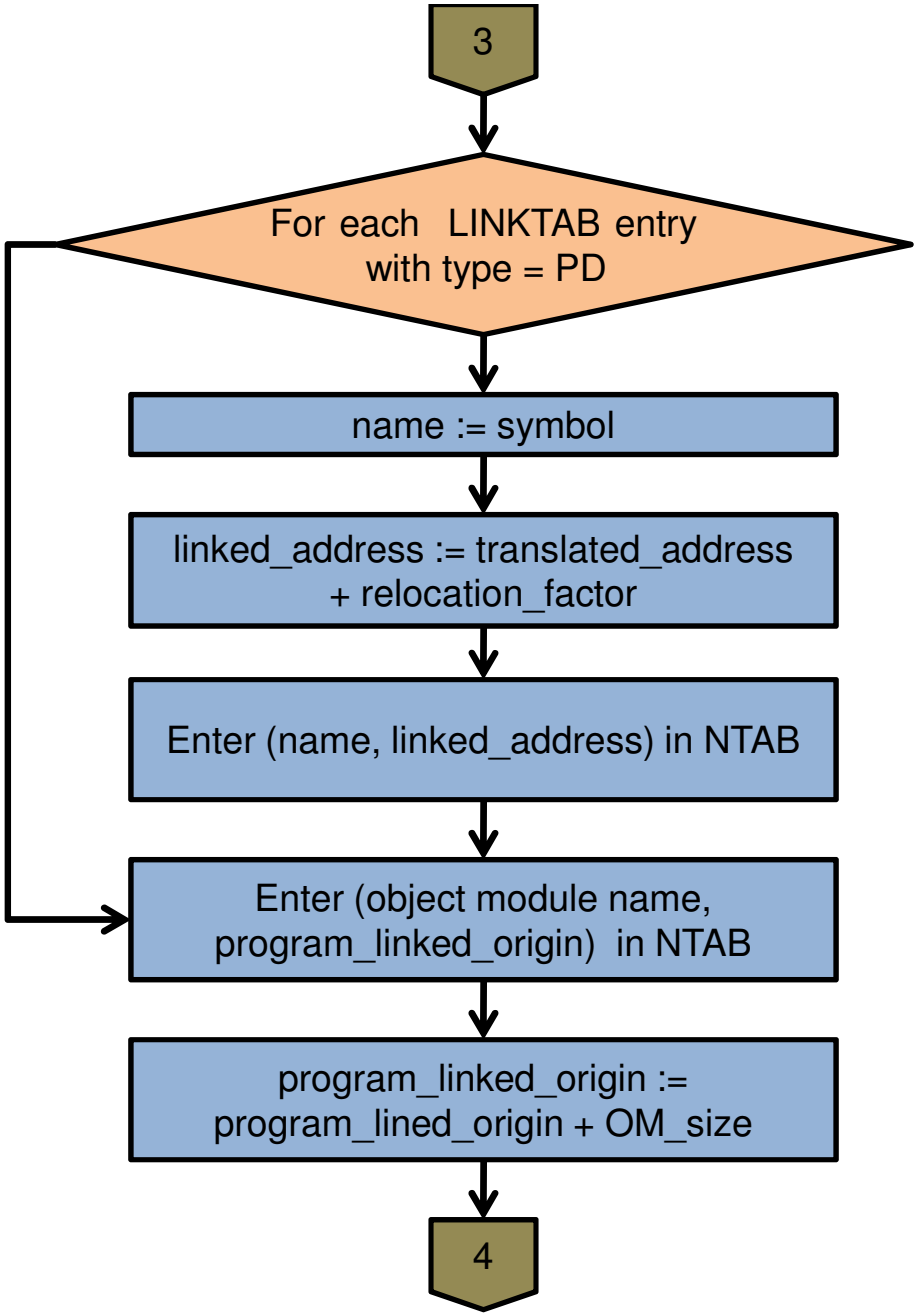
b) For each LINKTAB entry with type = EXT

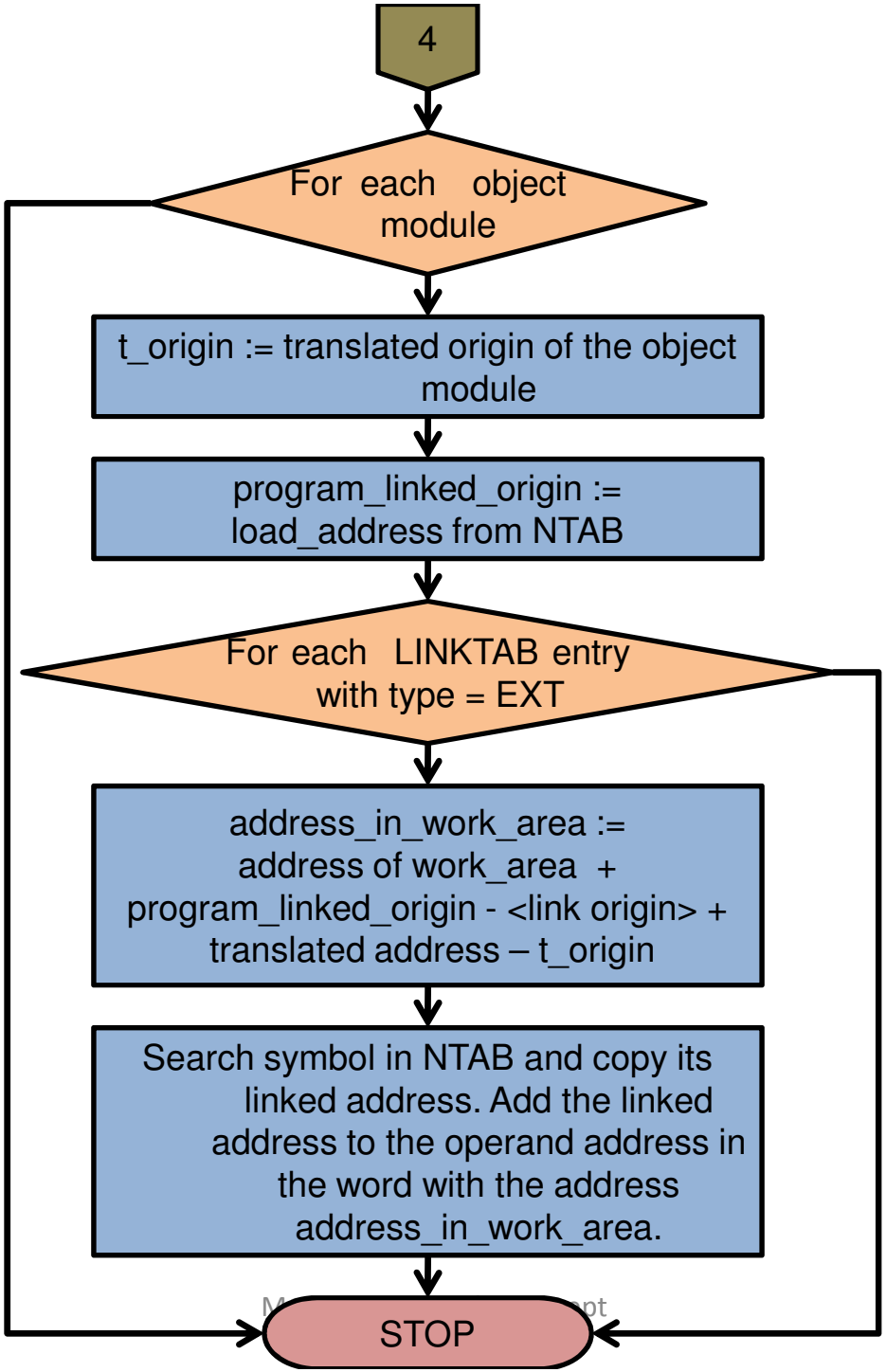
i) address_in_work_area := address of work_area + program_linked_origin - <link origin> + translated address – t_origin;

ii) Search symbol in NTAB and copy its linked address. Add the linked address to the operand address in the word with the address address_in_work_area.


4
Read object module
program_linked_origin :=
<link origin> from linker command
2


2
For each object module
t_origin := translated origin of the object module
OM_size := size of the object module
relocation_factor := program_linked_origin – t_origin
Read the machine language program in work_area
Read LINKTAB of the object module
3


3
For each LINKTAB entry
with type = PD
name := symbol
linked_address := translated_address
+ relocation_factor
Enter (name, linked_address) in NTAB
Enter (object module name,
program_linked_origin) in NTAB
program_linked_origin :=
program_lined_origin + OM_size
4


4
For each object module
t_origin := translated origin of the object module
program_linked_origin := load_address from NTAB
For each LINKTAB entry with type = EXT
address_in_work_area := address of work_area + program_linked_origin - <link origin> + translated address – t_origin
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.
STOP

Design of a Linkers

- Linking Requirements

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		541)	
	END			

Design of a Linkers

- Linking requirements

Program Q

	Statement		Address	Code
	START	200		
	ENTRY	ALPHA		

ALPHA	DS	25	231)	+ 00 0 025
	END			

  - linked_origin = 900

Design of a Linkers

- Linking requirements
  - linked_origin = 900
  - NTAB

Symbol	Liked address
P	900
TOTAL	941
Q	942
ALPHA	973

  - work_area = 300
  - For ALPHA entry of LINKTAB
    address_in_work_area := 300 + 900 – 900 + 518 – 500
    := 318
  - Linked address of ALPHA 973 is copied from NTAB entry of ALPHA and added to the word in address 318.

5. Linkers

- Relocation and Linking Concepts
- Design of a Linker
- Self-Relocating Programs
- Linking for Overlays

Self-Relocating Program

- Programs can be classify into
  - **Non relocatable program**
  - **Relocatable program**
  - **Self- relocating program**

Self-Relocating Program

- **Non relocatable program** : is a program which can not be executed in any memory area other than the area starting on its translated origin.
- **Relocatable program** : can be processed to relocate it to a desired area of memory.

Self-Relocating Program

- **Self- relocating program** : is a program which can perform the relocation of its own address sensitive instructions. It contains two provision:
  - A table of information concerning the address sensitive instructions exists as a part of the program.
  - Code to perform the relocation of address sensitive instructions also exists as a part of the program which is called the relocating logic.

5. Linkers

- Relocation and Linking Concepts
- Design of a Linker
- Self-Relocating Programs
- Linking for Overlays

Linking for Overlays

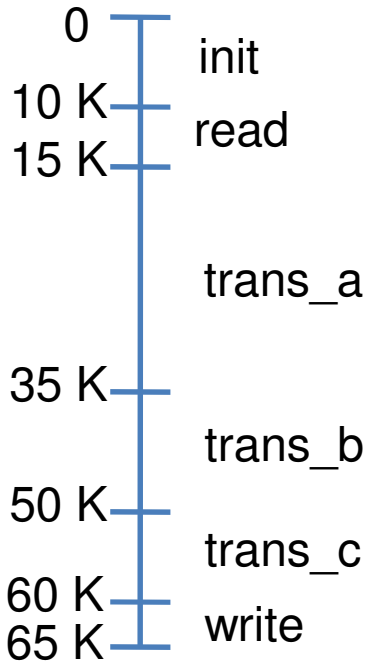
- Linking for Overlays
  - An overlay is a part of a program or software package which has the same load origin as some other part of the program.
  - Overlay structured program consist of
    - A permanently resident portion called the root
    - A set of overlays.
  - The overlay structure of a program is designed by identifying mutually exclusive modules.

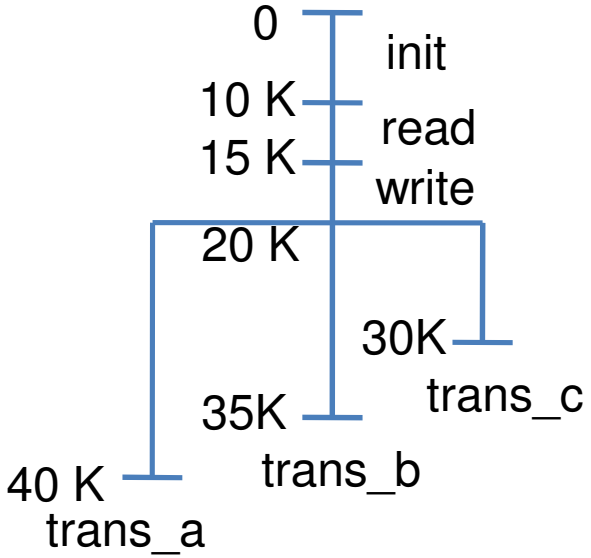
Linking for Overlays

- Linking for Overlays
  - Example: Consider a program with 6 sections named init, read, trans_a, trans_b, trans_c and print.
  - init perform some initialization and transfer control to read.
  - read reads one set of data and invokes one of trans_a, trans_b ot trans_c depending on the values of the data.
  - Print is called to print the result.
  - trans_a, trans_b and trans_c are mutually exclusive.

Linking for Overlays

- Linking for Overlays
  - Example:





Linking for Overlays

- Linking for Overlays
  - MS-DOS LINK command

    LINK init + read + write + (trans_a) + (trans_b) + (trans_c), <executable file>, <library files>

  - IBM mainframe linker command

```
Phase main :     PHASE    MAIN, +10000
                 INCLUDE  INIT
                 INCLUDE  READ
                 INCLUDE  WRITE
Phase a_trans :  PHASE    A_TRANS, *
                 INCLUDE  TRANS_A
Phase b_trans :  PHASE    B_TRANS, A_TRANS
                 INCLUDE  TRANS_B
Phase c_trans :  PHASE    C_TRANS, A_TRANS
                 INCLUDE  TRANS_C
```

Linking for Overlays

- Execution of an overlay structured program
  - For linking and execution of an overlay structured program in MSDOS
    - The linker produce a single executable file at the output which contains two provisions
      - An overlay manager module for loading the overlays when needed
      - All calls that cross overlay boundaries are replaced by an interrupt producing instruction