Linkers

Execution of program

1. Translation of program.
2. Linking of one prog with another progs. Needed for its execution
3. Relocation of the program to execute form the specific memory area location
4. Loading of the program in memory to perform execution

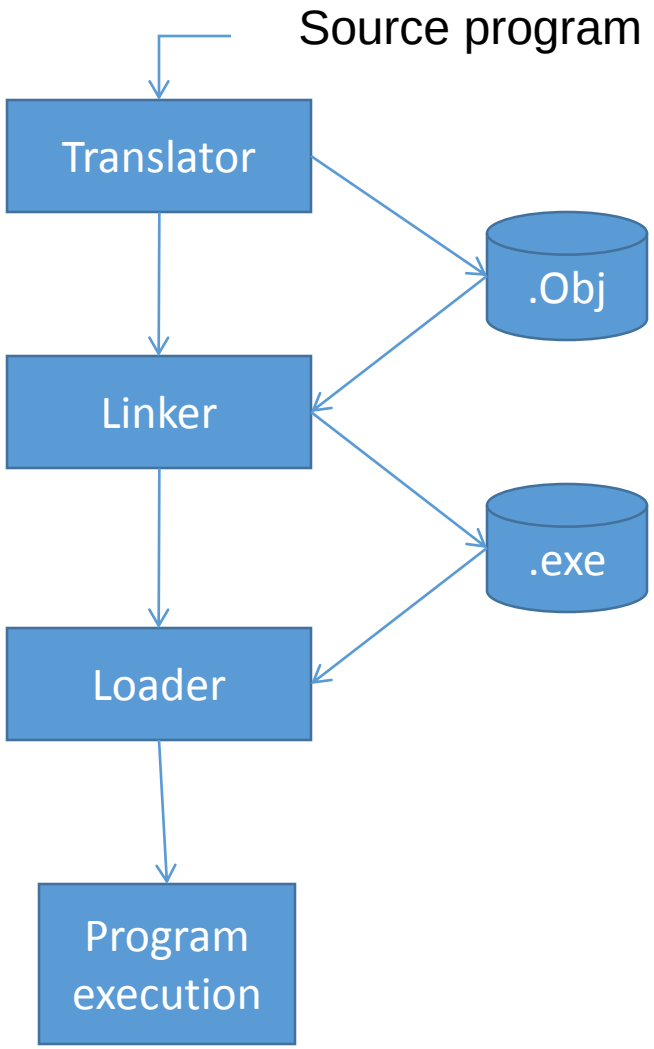
•Step 1 is perform by translator

•Steps 2 & 3 are performed by linker

•step 4 is performed by loader

Introduction:

- Translator translates source program to machine code and creates object file.
- Linker receives set of object files and links them solving external reference problems and creates ready to execute file.(which is a binary file).
- Loader loads this binary file to execution area for its execution



Some terminology used to refer to programing address

- Translation time address :address assign by translator
- Linked address : address assign by linker
- Load time address: address assign by loader

- Same terminology as assigning the origin of program
- ***Translated origin***: Address of origin assumed by the translator.
  - This address is specified by the programmer in an ORIGIN statement.
- ***Linked origin***: Address of origin assign by the linker while generating binary program
- ***Load origin***: Address of origin assign by the loader while loading program for execution.

Important terminologies :

Lang. Processors->	Translator	Linker	Loader
Types of address			
Start/Origin	Translator_Orgin (t_origin)	Linker_Orgin (l_origin)	Loader_Origin (ld_origin)
Symbol Address	Translated address/Translator time address (tsymb)	Link address / Link time address (lsymb)	Load address/Load time address (ldsymb)

- Assembly language program and its generated code

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 1	541)	
	END		

Relocation and linking Concepts

- Let AA be set of absolute addresses used in the program P.
- If AA!= Φ implies that program P assumes that instructions and data occupy specific addresses. a such program is called address sensitive program.

Address sensitive programs can execute correctly only if start address is same as translated origin. To execute correctly , addresses must be corrected.

Program relocation

- It is the process of modifying the addresses used in address sensitive instruction of a program such that the program can execute correctly from the designated area of memory
- Relocation can be done by the linker or loader
- If linked origin != translated origin then relocation is done by linker
- If load origin!= linked origin then relocation is done by loader
- In general linker performs relocation.

- Relocation factor=l_origin_p - t_origin_p (1)

- If statement uses symbol as an operand, then
- tsymb=t_origin+d_{symb}
  - d_{symb} is the offset of a program

- For link symbol
- lsymb = l_origin+dsymb

- lsymb=t_origin +relocation factor+dsymb
- =t_orgin +dsymb+relocation factor
- =tsymb+relocation factor

IRR=Instructions requiring relocation.

- Ex: Let IRR for prog P has dispalcement(dsymb)=40,t_origin=500,l_origin=900 then
- Relocation factor=900-500=400
- tsymb=500+40=540;lsymb=900+40=940
- lsymb=540+400=940

Linking

- ***Linking***: Linking is a process of binding an external reference to the correct link time address
- An Application Program AP consists of a set of program unit SP={x}
- Suppose that prog. **x** interact with other prog. **y** by using address of y's instruction and data of its own.
- This interaction contains public def. and ext ref.
- Public definition: a symbol defined in a program unit which may be referenced in other program units.
- Ext. ref : a ref to a symbol which is not defined in the program unit containing the references.

- Assembly language program and its generated code

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 1	541)	
	END		

- Ad

Statement			*Address*	*Code*
	START	200		
	ENTRY	ALPHA		
	- -			
	- -			
ALPHA	DS	25	231)	+ 00 0 025
	END			

Absolute loader

- Absolute loader doesn’t need to perform functions like linking and program relocation.
- For example take assembly language example of **Simplified Instructional Compute (SIC)** instruction as shown in figure.
- All functions are accomplished in a single pass assembler
- First the header record is checked to verify that correct program has been represented for loading.
- Input is read ,when the end record is encountered the loader jumps to the specified address to begin the execution of the loaded program.

```
  5     COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
 10     FIRST     STL     RETADR      SAVE RETURN ADDRESS
 15     CLOOP     JSUB    RDREC       READ INPUT RECORD
 20               LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
 25               COMP    ZERO
 30               JEQ     ENDFIL      EXIT IF EOF FOUND
 35               JSUB    WRREC       WRITE OUTPUT RECORD
 40               J       CLOOP       LOOP
 45     ENDFIL    LDA     EOF         INSERT END OF FILE MARKER
 50               STA     BUFFER
 55               LDA     THREE       SET LENGTH = 3
 60               STA     LENGTH
 65               JSUB    WRREC       WRITE EOF
 70               LDL     RETADR      GET RETURN ADDRESS
 75               RSUB                RETURN TO CALLER
 80     EOF       BYTE    C'EOF'
 85     THREE     WORD    3
 90     ZERO      WORD    0
 95     RETADR    RESW    1
100     LENGTH    RESW    1           LENGTH OF RECORD
105     BUFFER    RESB    4096        4096-BYTE BUFFER AREA
```

```
HCOPY  001000001070A
T001000 1E 14 10 33 48 20 39 00 10 36 28 10 30 30 10 15 48 20 61 3C 10 03 00 10 2A 0C 10 39 00 10 2D
T001010 1E 15 0C 10 36 48 20 61 08 10 33 4C 00 00 45 4F 46 00 00 03 00 00 00
T002039 1E 04 10 30 00 10 30 E0 20 5D 30 20 3F D8 20 5D 28 10 30 30 20 57 54 90 39 2C 20 5E 38 20 3F
T002057 1C 10 10 36 4C 00 00 F1 00 10 00 04 10 30 E0 20 79 30 20 64 50 90 39 DC 20 79 2C 10 36
T002073 07 38 20 64 4C 00 00 05
E001000
```

(a) Object program

Memory address	Contents			
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮
0FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
1000	14103348	20390010	36281030	30101548
1010	20613C10	0300102A	0C103900	102D0C10
1020	36482061	0810334C	0000454F	46000003
1030	000000xx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮
2030	xxxxxxxx	xxxxxxxx	xx041030	001030E0
2040	205D3020	3FD8205D	28103030	20575490
2050	392C205E	38203F10	10364C00	00F10010
2060	00041030	E0207930	20645090	39DC2079
2070	2C103638	20644C00	0005xxxx	xxxxxxxx
2080	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮	⋮	⋮	⋮	⋮

← COPY

(b) Program loaded in memory

Bootstrap loader

- Alternatively referred to as **bootstrapping**, **bootloader**, or **boot program**, a **bootstrap** loader is a program that resides in the computers EPROM, ROM, or other non-volatile memory that automatically executed by the processor when turning on the computer.
- The bootstrap loader reads the hard drives boot sector to continue the process of loading the computers operating system.

When a computer is first turned on or restarted, a special type of absolute loader, called *bootstrap loader* is executed

This bootstrap loads the first program to be run by the computer -- usually an operating system

Binary programs

- It is machine language program contains a set of program units (SP) such that for all **Pi** belongs to **SP**
- i) Pi has been relocated to the memory area starting at link origin
- ii) Linking has been performed for each external reference
- To create binary program from object modules linker invocation is required
- Linker<link origin>,<object module name>[,<execution start address>]

Design of linker:

- **Object module**: It contains all information necessary to relocate and link the program. It consists of following elements:

- 1)**Header:**Contains translated origin ,size and execution start addtress of program.
- 2)**Program**:Contains the machine code corresponding to program.
- 3)**RELOCTAB**: Descibes IRR.It contains single field:translated address.
- 4)**LINKTAB**(Symbol,Type,translated address):Contains information concerning public definitions.
- Type:PD/EXT

```
          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      1                541)
          END
```

Object module header

T_origin	size	Exe_start_address
500	42	500

LINKTAB

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

RELOCTAB

Translated Address
500
538

Design of linker:

Relocation algorithm:

1.program_linked_origin:=<link origin>
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_orgin;
c)Read the m/c lang. prog. In work area;
d)Read RELOCTAB of the object module
e)For each entry in RELOCTAB
i)Translated_addr:=Address in theRELOCTAB entry
ii)Addr_in _work_area:=Addr_of _work_area+Translated_addr-t_orgin;
iii)Add relocation_factor to the operand address in the word with the addr. in work area.

f) program_linked_origin:= program_linked_origin+OM_size;

```
        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      1                541)
        END
```

Ex: Let addr in work area=300,
Link origin=900,t_origin=500 ,size=42 then
Relocation factor=900-500=400
Addr_in _work_area=300+500-500=300
This word contains the instruction for
READ A
It is relocated by adding 400 to the operand address in it.

Design of linker:

Linking algorithm:

1.program_linked_origin:=<link origin>
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_orgin;
    c)Read the m/c lang. prog. In work area;
    d)Read LINKTAB of the object module

    e)For each entry in LINKTAB with type=PD
     i)name:=symbol;
       linked_address:=translated_address+relocation factor;
       Enter(name,linked_address)in NTAB
    f)Enter(object module name,Prog_linked_origin)in NTAB
    g)Program_linked_origin:=Program_linked_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)Addr_in_work_area:=Addr of work area+Program_linked_origin-Link_origin+translated addr. –t_orgin;

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

NTAB

Symbol	Linked address
Total	941
P	900
Q	942
Alpha	973

LINKTAB

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

SELF RELOCATION PROGRAMS

Non relocating programs

is a program which cannot be executed in any memory area other than area starting on its translated origin

Relocatable program : Can be processed to relocate it to a desired area of memory

Self- relocating programs

Program which can perform the relocation of its own address sensitive instructions.

It Contains

A table of information concerning address sensitive instructions

Code to perform relocation of ASI. This is called relocating logic

Linking for overlays

- An overlay is a part of a program which has same load origin as some other part of the program

- Used to reduce memory requirements of the program

- Program containing overlays is called overlay

- structured program.

It consists of

1. Permanently resident portion, called root
2. Set of overlays

- First root is loaded and given control for execution
- Other overlays are loaded when its needed.
- Loading new overlay overwrites old overlays coz we have same load origin for all overlays
- The structure of program is designed by identifying mutually exclusive modules i.e modules do not call each other.
- Should be avoided to reside simultaneously in memory.

- Ex
- Program with six section name as init, read ,prog_a,prog_b , prog_c and print.
- How overlay is done for this program..?