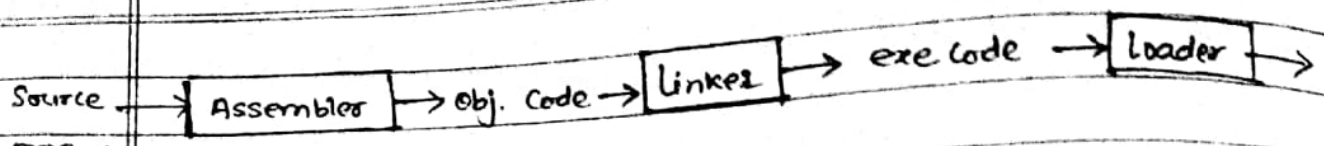
Date
Page

Chapter :- 3 Loader and Linker //



→ Loader is a system program that performs loading function.
→ many loaders also support relocation and linking.
→ Some systems have linker to perform linking operation and a separate loader to ~~have~~ handle relocation and loading

→ loading
:- brings object program into memory for execution.

→ Relocating.
:- modifies the object program so that it can be loaded at an address different ~~from~~ from location originally specified.

→ Linking.
:- Combines two or more separate object program.

Basic loader function

① bringing an object program into memory.
② Starting its execution. //

* Absolute Loader

→ An object program is loaded at the address specified on the START directive.

→ No relation or linking is needed.

→ All functions are accomplished in a single pass as follows

ⓐ The header record of object program is checked to verify that correct program has been presented for loading.

ⓑ As each text record is read, the object code it contains is moved to begin execution of loaded program.

ⓒ When the end record is encountered, loader jumps to the specified address to begin execution of loaded program.

→ Fig ① shows the representation of program (SIC simple) after loading.

Memory address	Contents			
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
0010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮				
1000	14103348	20390010	36281030	30101548
1010	20613c10	0300102A	0C103900	10200c10
1020	36482061	08103348	00004548	46000003
1030	000000xx	xxxxxxxx	xxxxxxxx	xxxxxxxx
⋮				
2030	xxxxxxxx	xxxxxxxx	xx041030	001030E0
2040				
2050				
2060				
2070	2C103638	20644C00	0005xxxx	xxxxxxxx

Date
Page

Algorithm for Absolute loader.

```
begin
    read Header Record
    verify program name and length.
    read first text record
    while record type ≠ E do
    begin
    {
        if object codes in characters form, convert
        into internal representation.
        move object code to specified location in memory
        read next object program record.
    end:
        jump to address specified in End record.
    end.
```

A ~~Bootstrap~~ simple Bootstrap loader.

- Special type of absolute loader
- loads the first program to be run by computer usually OS
- bootstrap begins at address 0 in memory of machine.
- load OS at address 80.
- each byte of object code to be loaded, is represented on device F3. as two hexadecimal digit just as in text record of SIC object program.
- the object code from device F3 is always loaded into consecutive bytes of memory starting at address 80.
- The main loop of bootstrap keeps the address of next memory location to be loaded in register X.
- After all of the object code from device F3 has been loaded, the bootstrap jumps to address 80, which begins execution of program that was loaded.

→ Much of work is performed by subroutine GETC,
→ GETC is used to read and convert a pair of characters from F3 representing 1 byte of object code to be loaded.
→ The resulting byte is stored at address currently in register X, using STCH instructions that refers to location 0 using index addressing.
→ TIXR is used to add 1 to value in X.

Machine dependent loader feature.

→ One of the disadvantage of absolute loader is need for programmer to specify the actual address at which it will be loaded into memory.
→ On a large and more advanced machine, several independent programs are run together, sharing memory between them.
→ We do not know in advance where a program will loaded. Hence we write relocatable program instead of absolute ones.
→ Loaders that allow for program relocation are called relocating or relative orders.

Relocation :-

- Use of two methods for specifying relocation as part of object program.

First method

- a modification is used to describe each part of object code that must be changed when the program is relocation.

- Consider example 2 (SIC/XE)
- most of instruction used relative or immediate addressing.
- The only portion of assembled program that contains actual address are the extended format instruction on line 15, 35, 65.
- These are only items whose value is affected by relation.
- refer the corresponding object program.
- Here, each modification record specifies starting address and length of field whose value to be changed.
- it then describes modification to be performed.
- In this, all modification add the value of symbol COPY, which represents starting address of program.
- Although modification record scheme is convinient mean for specifying program relocation, it is not well suited for use with machine architectures.
- Consider the relocatable program written for SIC simple.
- here it does not use relative addressing.
- So, the address of all instruction except RSUB must be modified when program is relocated.
- this would require 3 modification record which results in an object program more than twice as large as previous one.

Date
Page

Second Method.

- There are no modification records.
- The text records are the same as before except that there is relocation bit associated with each word of object code
- Since all the SIC instruction occupy one ~~record~~ word, this means that there is one relocation bit for each possible instruction.
- The relocation bit are gathered together in a bit ~~mask~~ mask following length indicator in each text record.
- This mask is represented as three hexadecimal digits.
- If the relocation bit corresponding to a word of object code is set to 1, the programs starting address is to be added to this word when the program is relocated.
- a bit value of 0 indicates no modification is necessary.
- if a text record contains fewer than 12 words of object code, the bits corresponding to unused words are set 0 but mask = 12 bits
- Eg:- bit mask FFC (1111 1111 1100) in first text record specifies that all 10 words of object code are to be modified during relocation.

Object program with ~~the~~ relocation bit mask.

H^COPY^000000^001077

T^000000^1E^FFC^140033^---^00002D

T^00001E^15^E00^0C0036^---^000000

T^001039^1E^FFC^0[illegible]00 80^---^38103F

T^001039^0A^800^100036^---^001000

T^001061^19^FE0^040030^---^05

E^000000

→ The LDX instruction line 210 begins a new text record. - if it were placed in the preceding text record, it would not be properly aligned to correspond to a relocation bit because of (byte data value generated from line 185.

SIC Simple (Relative)

line	LOC	SYMBOL	OPCODE	OPERAND	object Code.
5	0000			0	140033
	0000				481039

Program Linking (from slide)

Machine Independent Loader features

Date | Page |

Automatic library search

→ Many linking loaders can automatically incorporate routines from a subprogram library into program being loaded.

→ In linking loaders the support automatic library search must keep track of external symbols that are referred to but not defined in the primary input to the loader.

→ At the end of pass 1, the symbols in ESTAB that remain undefined represent unresolved external references.

→ The loader seaches the library or libraries specified for routines that contain the definitions of these symbol and process the subroutine found by this search exactly if they had been part of primary input stream.

→ The subroutine fetched from a library in this way may themselves contain external references it is therefore necessary to repeat the library search process until all references are resolved.

→ If unresolved external references remain after the library search is completed, these must be treated as errors.

Loader Options.

→ Many loaders allow the user to specify options that modify standard processing.

→ Typical loader option 1: allows the selection of alternative source of input.

eg:- INCLUDE programme (library name) might direct the loader to read the designated object program from library and treat it as if it were part of the primary loader input.

→ loader option 2: allow the user to delete extern symbol or entire control sections.

eg:- DELETE CSECT – name might instruct the loader to delete the named control section from set of programs being loaded.

– CHANGE name 1, name 2 might cause the external symbol name 1 to be changed to name 2 wherever it appears in the object program.

– Loader option 3:- involves automatic inclusion of library routines to satisfy external references.
eg: LIBRARY MYLIB.
– such user specified libraries are normally searched before standard system libraries.

Loader design Options

→ Linking loaders perform all linking and relocation at load time.

→ There are two alternatives

ⓐ Linkage editors :- which perform linking prior to load time

ⓑ Dynamic linking:- in which the linking function is performed at execution time.

Date
Page

difference between linking editor and linking loades.

→ A linking loader performs all linking and relocation operations, including automatic library search, and loads the linked program into memory for execution.

→ a linkage editor procedures a linked version of the program which is normally written to a file for later execution.

Linkage Editor.

→ When the user is ready to run the linked program a simple relocating loader can be used to load the program into memory.

→ the only object code modification necessary is the addition of actual address to relative values with ~~to be~~ with in program.

→ ~~all items that need to~~ the linkage editor performs relocation of all control sections relative to start of linked program.

→ the means that the loading can be accomplished in one pass with no external symbol table required.

→ all items that need to be modified at load time have values that are relative to the start of the linked program.

→ Thus, if a program is to be executed many times without being reassemble the use of linkage editor can substantially reduces the overhead required.

→ Linkage editor can also be used to build packages of subroutines or other control sections that are generally used together.

→ Linkage editor often include variety of other options and commands like in linkage loaders. But in general

Date
Page

linkage editor tend to offer more control and flexibil

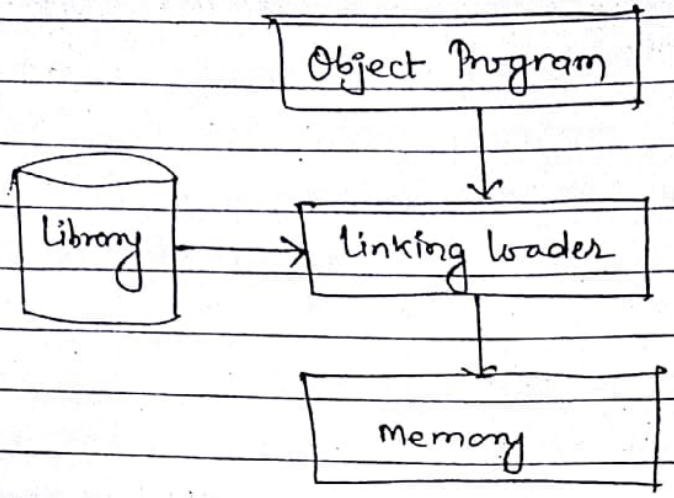


Fig:- Processing of object program in linking loade

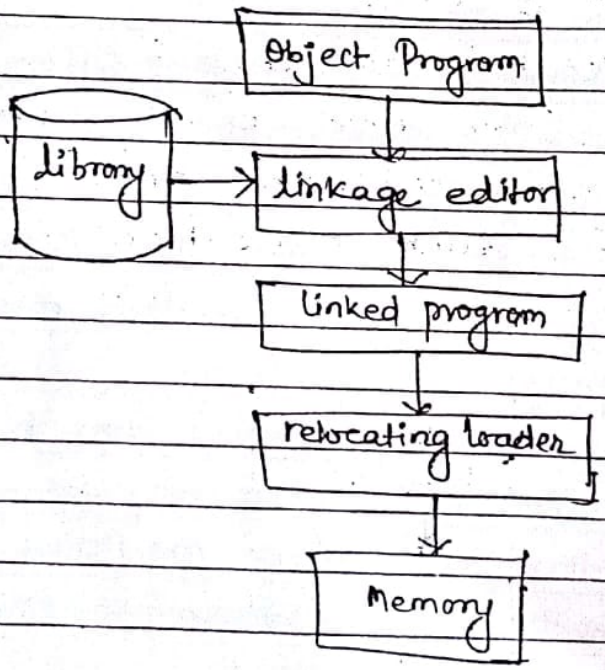


Fig:- Processing of object program in linkage editor.

Date
Page

Dynamic Linking

→ linkage editors perform linking before the program is loaded.

→ Linking loader perform linking at load time.

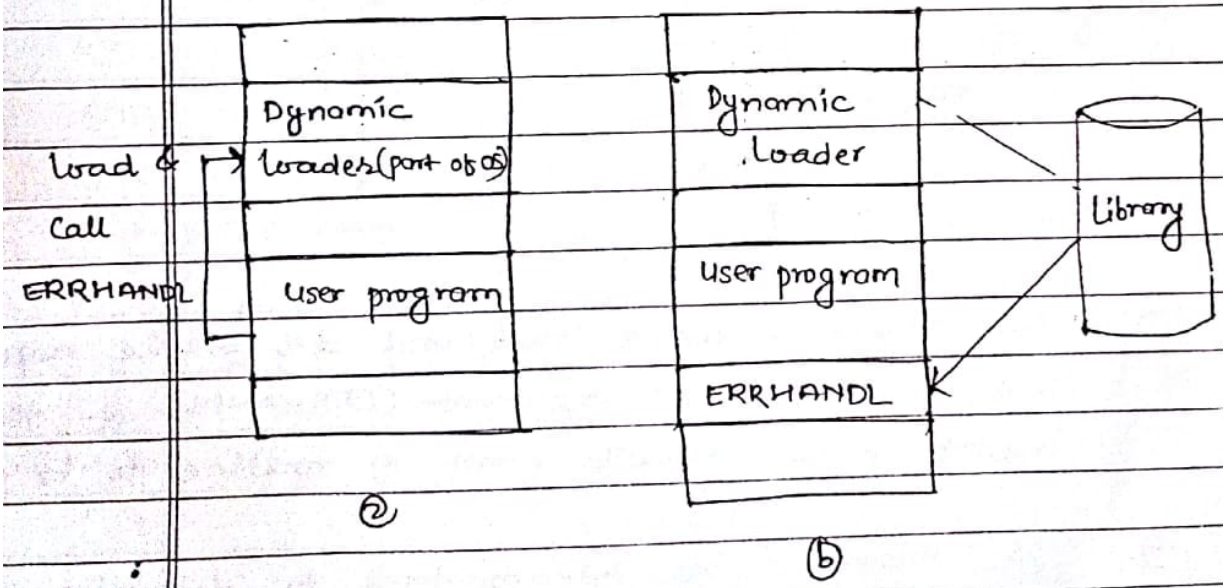
→ Dynamic linking postpones the linking function until execution time.

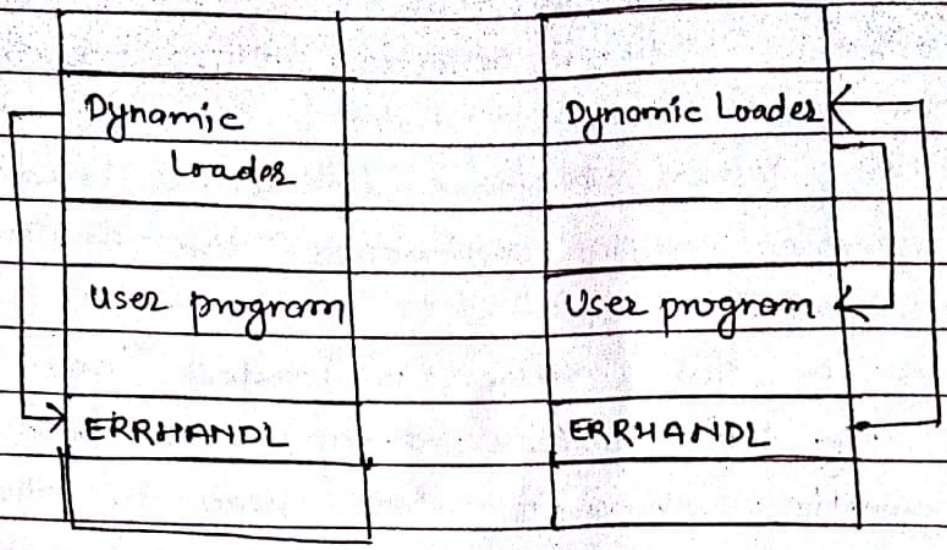
- a sub routine is loaded and linked to the rest of the program when it is first called.

→ Dynamic linking is often used to allow several executing programs to share one copy of subroutine or library.

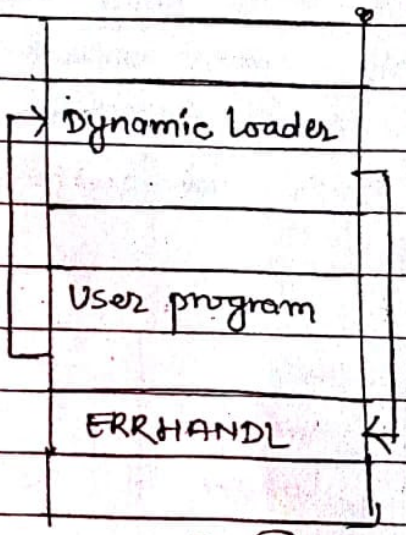
→ eg:- a single copy of standard C library can be loaded into memory. All C programs currently in execution can be linked to this one copy instead of linking a separate copy into each object program.

→ Dynamic linking can avoid the necessity of loading entire library for each execution except those necessary subroutine.





© ⓓ



ⓐ ⓔ

ⓐ → the program makes a load and call service request to OS the parameter argument (ERRHANDL) of this request in the symbolic name of routine to be called.

ⓑ → OS examines its internal task to determine whether or not the routine is already loaded. If necessary, the routine is loaded from specified user or system libraries.

⑤→ Control is then passed from OS to routine being called.

⑥→ When the called subroutine completes its processing, it returns to its caller (i.e OS). OS then returns control to the program that issued the request.

⑦→ If a subroutine is still in memory, a second call to it may not require another load operation control may simply be passed from dynamic loader to the called routine.

Line	loc	symbol	opcode	Exp	Object code
10	5000	STRCPY	START	5000	
20	5000	FIRST	LDX	ZERO	04501E
30	5003	MOVECH	LDCH	STR1,X	50D00F
40	5006		STCH	STR2,X	54D013
50	5009		TIX	ELEVEN	2C5021
60	500C		JLT	MOVECH	385003
70	500F	STR1	BYTE	C'ABCD'	41424344
80	5013	STR2	RESB	11	
90	501E	ZERO	WORD (3)	0	000000
100	5021	ELEVEN	WORD (3)	11	00000B
110	5024		END	FIRST	

LDX → 04

LDCH → 50

STCH → 54

JLT → 38

TIX → 2C

50 500F

8

0101 0000 0101 0000 0000

0

8 | X | 15

1

D

12/2 = 26

06

1

8

8

38, 19/16

Start मा zero नभए भने relocatable हुन्छ
यदि छैन भने absolute

Date
Page

5000 H^STRCOPY^005000^000024
5010 T^005000^13^04501E^ --- ^41424844
5020 T^00501E^06^000000^000011
50 E^005000
0000 xxxxxxxx xxxxxxxx

5000	04501E50	D00F54D0	132C5021	38500341
5010	42434400	xxxx xxxx	xxxx xxxx	xxxx 0000
5020	00000002	xx xx xxxx	xxxxxxxx	xxxxxxxx

Assignment

1. How forward reference is handled in one pass assembler?
2. What is relocation? How relocation is carried out in a loader?
3. What is advantage of relative addressing mode over absolute addressing mode?
4. Write about program blocks and control section?
5. What are main features of machine dependent loader when logically related parts of programming are linked then what is generated and why it is important?