LOADER:

Is a utility program which takes object code as input,prepares it for execution and loads the executable code into the memory.Thus loader is actually responsible for initiating the execution process.

1.What are the basic functions of a loader?

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

ALLOCATION: it allocates the space for program in the memory, by calculating the size of the program.

LINKING: it resolves the symbolic references (code/data) between the object modules by assigning all the user subroutine and library subroutine addresses.

RELOCATION: There are some address dependent locations in the program such as address constants must be adusted acording to allocated space such activity done by loader.

LOADING:finally it places the machine instructions and data of corresponding programs and subroutines into the memory. Thus program now becomes ready for execution.

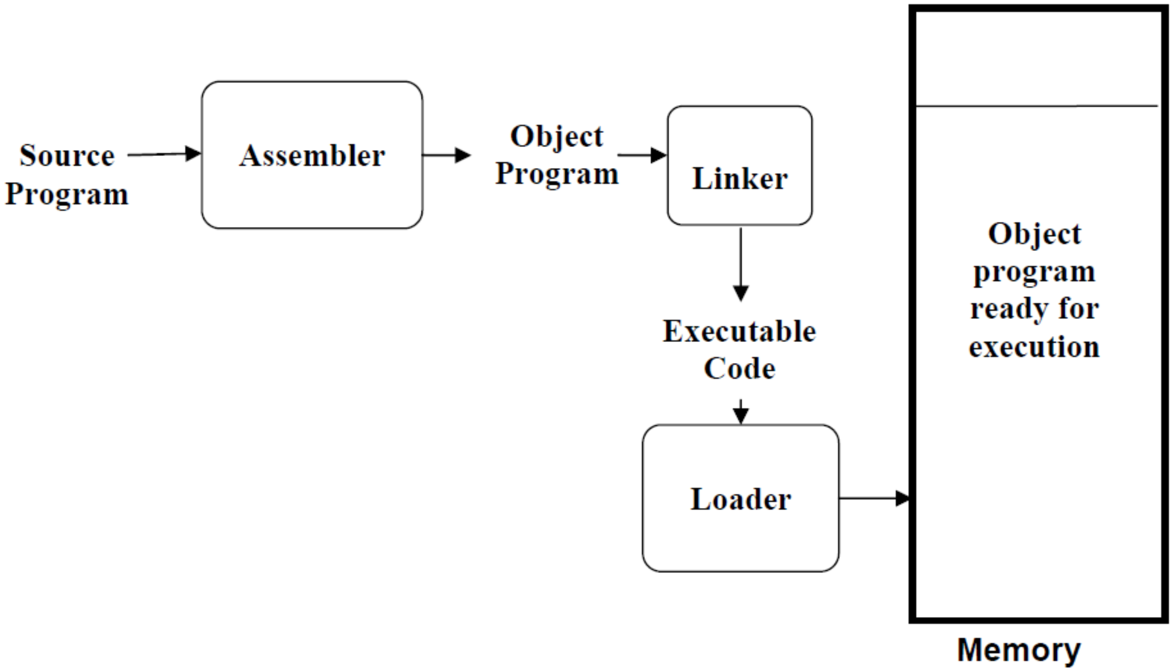


Fig: Role Of Loader and Linker

2. Explain the design of an Absolute Loader.

- Absolute Loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at specified locations in the memory. This type of loader is called absolute because no relocation information is needed rather it is obtained from the programmer or assembler.
- The starting address of every module is known to the programmer this corresponding address is stored in the object file, then task of loader becomes very simple and that is to simply place the executable form of the machine instructions at the locations mentioned in the object file.
- The programmer should take care of two things :

- First, Specification of starting address of each module to be used.
  If some modification is done in some module then the length of that module may vary. This causes a change in the starting address of immediate next modules, its then programmer duty to make necessary changes in the starting addresses of respective modules.
- Second, while branching from one segment to another the absolute starting address of respective module is to be known by the programmer so that such address can be specified at respective JMP instruction.

The record in the object code is of the following form
Header record
Text record
End record
The absolute loader accepts this object module from assembler and by reading the information about the starting address, it will place the subroutine at specified address in the memory.

Ex: H^COPY ^001000^00107A
T^001000^1E^141033^482039………
T^002039^1E^041030^001030^E0205D…..
E^001000

Memory
Locations   CONTENT

0000	**xxxxxxxx**	**xxxxxxxx**	**xxxxxxxx**	**xxxxxxxx**
0010				
:				
1000	**14103348**	**20390010**	**36281030**	**30101548**
1010				
:				
2030	**xxxxxxxx**	**xxxxxxxx**	**XX041030**	**001030E0**

The object code is divided into 1byte(hex notation) at one mem locations
For first text record:
1000-→14
1001→10
1002→33
1003→48
:
:
100F->48

ALGORITHM for Absolute loader:
INPUT: object code and starting address of the program segment
OUTPUT: an executable code corresponding to the source program.

```
Begin
read Header record
verify program name and length
read first Text record
while record type is ≠ 'E' do
begin
{if object code is in character 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
```

ADVANTAGES:

- It is simple to implement
- The task of loader becomes simpler as it simply obeys the instruction regarding where to place the object code into main memory.
- The process of execution is efficient

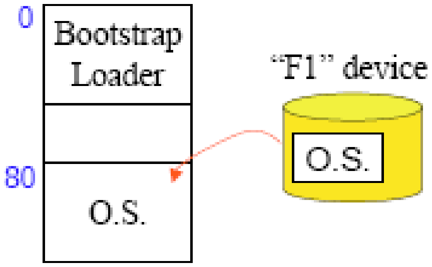
DISADVANTAGES:

- It is the programmer duty to adjust all the inter segment addresses and manually do the linking activity .For that programmer has to know the memory management.

- If at all any modification is done the some segments, the starting addresses of immediate next segment may get changed, the programmer has to take care of this issue and he needs to update the corresponding addresses on any modification in the source.

3. Explain Bootstrap Loader with the algorithm.

- 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. The bootstrap itself begins at address 0. It loads the OS starting address 0x80. No header record or control information, the object code is consecutive bytes of memory.



Various characteristics of bootstrap loader

- The bootstrap loader is a small program and it should be fitted in the ROM.
- The bootstrap loader must load the necessary portions of OS in the main memory
- The initial address at which the bootstrap loader is to be loaded is generally the lowest for example at location 0000 or at the highest location and not a intermediate location**.**

BOOTSTRAP LOADER FOR SIC/XE

Begin

X=0x80 (the address of the next memory location to be loaded)

```
BOOT        START       0
            CLEAR       A
            LDX         #128    //Initialize the reg X to hex 80
LOOP        JSUB        GETC //read hex digit from program being loaded
            RMO         A,S     //save in reg S
            SHIFTL      S,4     //move to high order 4 bits of byte
            JSUB        GETC //get next hex digit
            ADDR        S,A     //combine  digits to form one byte
            STCH        0,X     //store at address in reg X
            TIXR        X,X     //add 1 to memory address being loaded
            J           LOOP //loop until end of input is reached
```

It uses a subroutine GETC, which is

```
GETC        TD          INPUT        //test input device
            JEQ         GETC         //loop until ready
            RD          INPUT        //read character
            COMP        #4           //if char is hex 04
            JEQ         80           //jump to start of program just loaded
            COMP        #48          //compare to hex 30
            JLT         GETC        //skip characters less than ‘0’
            SUB         #48          //subtract hex 30 from ASCII code
            COMP        #10          //if result less than 10, conversion done
            JLT         RETURN
            SUB         #7      //otherwise subtract 7 more(hex digits ‘A’ Through ‘F’.
RETURN     RSUB
INPUT       BYTE        X’F1’
            END         LOOP
```

```
GETC A←read one character
if A=0x04 then jump to 0x80
if A<48 then GETC
ELSE
A ← A-48 (0x30)
if A<10 then return
ELSE
A ← A-7
return
```

MACHINE DEPENDENT LOADER FEATURES:

- **RELOCATION**
- **PROGRAM LINKING**

4. Explain the need of relocation of a program. Explain how its implemented
Or
4. What is relocating loader. What are the two methods for specifying relocation as part of object program.

Absolute loader is simple and efficient, but the scheme has potential disadvantages One of the most disadvantage is the programmer has to specify the actual starting address, from where the program to be loaded. This does not create difficulty, if one program to run, but not for several programs. Further it is difficult to use subroutine libraries efficiently.

This needs the design and implementation of a more complex loader. The loader must provide program relocation and linking, as well as simple loading functions.

Relocation

The concept of program relocation is, the execution of the object program using any part of the available and sufficient memory. The object program is loaded into memory wherever there is room for it. The actual starting address of the object program is not known until load time. Relocation provides the efficient sharing of the machine with larger memory and when several independent programs are to be run together. It also supports the use of subroutine libraries efficiently. Loaders that allow for program relocation are called relocating loaders or relative loaders.

There are two methods for specifying relocation as a part of object program and those are

1. Modification record (SIC/XE program)
2. Relocation bits(SIC program)

1. **Modification record:**
   a. For small number of relocation this method is useful. This method is used for SIC/XE program. Relative or immediate addressing is used in this method.
   b. Modification record format
   Col 1: M
   Col 2-7 Starting location of the address field to be modified relative to the beginning of the program(hexadecimal)
   Col 8-9 length of the address field to be modified, in half bytes(hexadecimal).
   Col 10:flag +or –
   Col 11: Name of the segment.

	COPY	START	0	
0000		STL	RETADR	17202D
0003		LDB	#LENGTH	69202D
0006	CLOOP	+JSUB	RDREC	4B101036
:	:	:	:	:
0013		+JSUB	WDREC	4B10105D
:	:	:	:	:
0026		+JSUB	WDREC	4B10106D

1036	RDREC	CLEAR	X	B410
:	:	:	:	:

105D	WDREC	CLEAR	X	B410
:	:	:	:	:

OBJECT PROGRAM

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D……..
M^000007^05+COPY
M^000014^05+COPY

M^000027^05+COPY
E^000000
```

Disadvantage:This method is not well suited for SIC program . because these programs will require lot of modified records and then the size of object program will get increased.

2. **RELOCATION BITS** The relocation bit method is used for simple machines. Relocation bit is 0: no modification is necessary, and is 1: modification is needed. This is specified in the columns 10-12 of text record (T),

The format of text record, along with relocation bits is as follows.

Text record

col 1: T
col 2-7: starting address
col 8-9: length (byte)
col 10-12: relocation bits
col 13-72: object code

Twelve-bit mask is used in each Text record (col:10-12 – relocation bits), since each text record contains less than 12 words, unused words are set to 0, and, any value that is to be modified during relocation must coincide with one of these 3-byte segments. For absolute loader, there are no relocation bits column 10-69 contains object code. The object program with relocation by bit mask is as shown below. Observe FFC - means all ten words are to be modified and, E00 - means first three records are to be modified.

LOCCTR	LABEL	OPCODE	OPERAND	OBJ CODE	RELOCATION BIT
	COPY	START	0000		
0000	FIRST	STL	RETADR	140033	1
0003	CLOOP	JSUB	RDREC	481039	1
0006		LDA	LENGTH	000036	1
0009		COMP	ZERO	280030	1
000C		JEQ	ENDFIL	300015	1
000F		JSUB	WRREC	481061	1
0012		J	CLOOP	3C0003	1

0015	ENDFIL	LDA	EOF	00002A	1
0018		STA	BUFFER	0C0039	1
001B		LDA	THREE	00002D	1
001E		STA	LENGTH	0C0036	1
0021		JSUB	WRREC	481061	1
0024		LDA	RETADR	080033	1
0027		RSUB		4C0000	0
002A	EOF	BYTE	C'EOF'	454F46	0
002D	THREE	WORD	3	000003	0
0030	ZERO	WORD	0	000000	0

For example :

Text record: T^000000^1E^140033^481039……………………..^00002D

1111 1111 1110 (FFE)

So new text record after adding relocation bit:

T^000000^1E^FFE^140033^481039……………………..^00002D

T^00001E^15^0C0036…………^000000

1110 0000 0000

So new text record after adding relocation bit:

T^00001E^15^E00^0C0036…………^000000

5. With the help of an example, explain how relocation and linking operations are performed.

PROG A

LOC	Label	Opcode	Operand	Object code
0000	PROGA	START	0	
		EXTDEF	LISTA,ENDA	
		EXTREF	LISTC,ENDC	
0020	REF1	LDA	LISTA	03201D
:	:	:	:	:
0040	LISTA	EQU	*	
:	:	:	:	;
0054	ENDA	EQU	*	
0054	REF4	WORD	ENDA-LISTA+**LISTC**	000014
0057	REF5	WORD	**ENDC**-LISTC-10	FFFFF6
0063		END	REF1	

PROG B

LOC	Label	Opcode	Operand	Object code
0000	PROGB	START	0	
		EXTDEF	LISTB,ENDB	
		EXTREF	LISTA,ENDA	
0060	LISTB	EQU	*	
:	:	:	:	:
0070	ENDB	EQU	*	
:	:	:	:	;
0070	REF4	WORD	**ENDA-** LISTA+LISTC	000000
007C	REF8	WORD	LISTB-**LISTA**	000060
:	:	:	:	:
007F		END		

PROG C

LOC	Label	Opcode	Operand	Object code
0000	PROGC	START	0	
		EXTDEF	LISTC,ENDC	
		EXTREF	LISTA,ENDA	
0018	REF1	+LDA	LISTA	03100000
0020	REF3	+LDX	**#ENDA-LISTA**	05100000
0030	LISTC	EQU	*	
:	:	:	:	:
0042	ENDC	EQU	*	
	:	:	:	:
0051		END		

OBJECT PROGRAM FOR PROGA

H^PROGA ^000000^000063 (header record)
D^**LISTA ^000040^ENDA ^000054**(define record)
R^**LISTC ^ENDC** (refer record)
T^000054^0F^000014^……..(text record)
M^000054^06^+LISTC(modification record)
M^000057^06^+ENDC
E^……(end record)

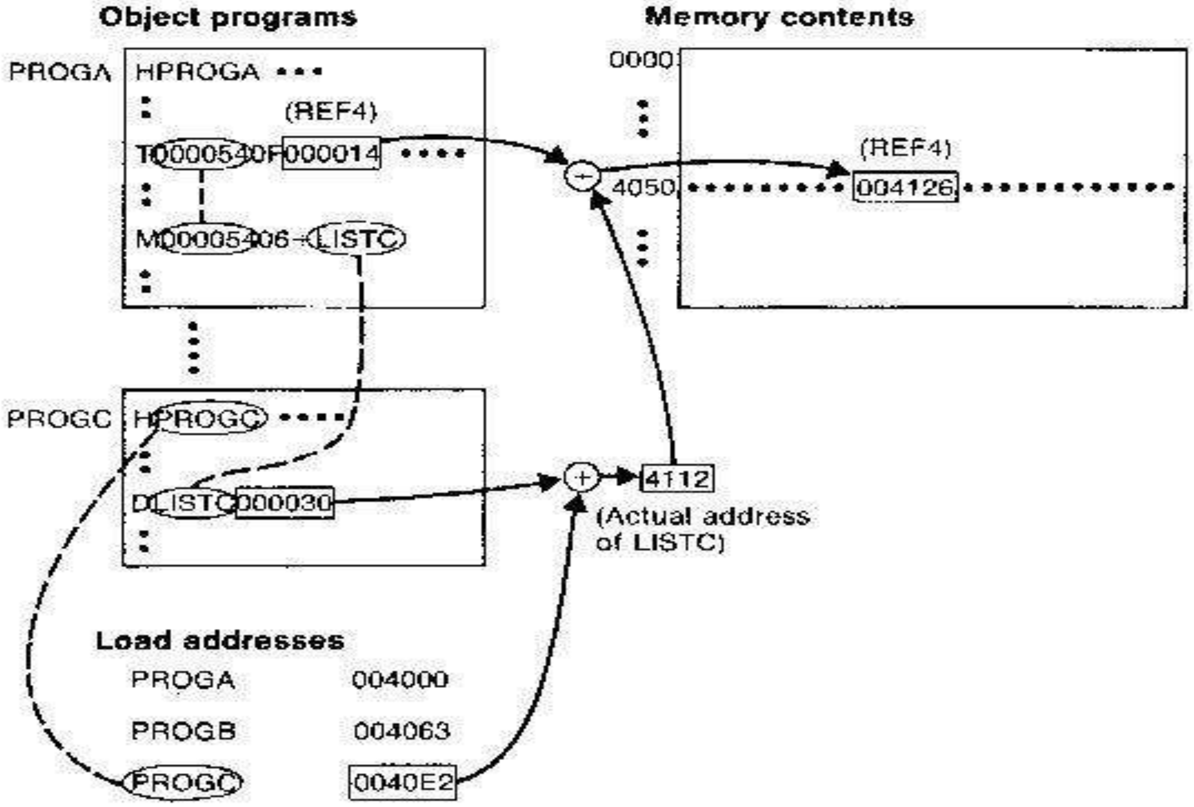
OBJECT PROGRAM FOR PROGC

H^PROGC ^000000^000051
D^**LISTC ^000030^ENDC ^000042**
R^**LISTA ^ENDA**
T^000018^0C^03100000^…………
T^000042^0F^000030^……..
M^000019^05^+LISTA
M^000021^05^+ENDA
E^….

Assume that PROGA has been loaded starting address **4000**, with PROGB,PROGC immediately following.
PROGA-→4000(Starting address)+0063(length of PROGA)-→4063
PROGB→4063(staring address)+007F(length of PROGB)→40E2
PROGC→40E2(starting address)+0051(length of PROGC)→

Control section	Symbol name	Addresss	length
PROGA		4000	0063
	LISTA	4040	
	ENDA	4054	
PROGB		4063	007F
	LISTB	40C3	
	ENDB	40D3	
PROGC		40E2	0051
	LISTC	4112	
	ENDC	4124	

For example the value of REF4 in PROGA is located at address 4054.



(PROGA starting address+ the relative address of REF4 within PROGA)=4000+0054=4054
The actual address of LISTC-→relative address of LISTC defined in PROGC + load address of PROGC-→000030+0040E2--→4112
The initial value from text record is 000014.to this is added the actual address assigned to LISTC 000014+4112----→4054.
4050→xx, 4051→xx, 4052→xx, 4053→xx, 4054→004126

6. Explain the algorithm for PASS 1 of a linking loader.

The main data structure needed for our linking loader is an **external symbol table(ESTAB).** This table is used to store name and address of each external symbol in the set of control sections being loaded. the table also often indicates in which control section the symbol is defined.

Two other important variables are PROGADDR(program address) and CSADDR(control section address) .

- PROGADDR is the beginning address in memory where the linked program is to be loaded.Its value is supplied to the loader by the operating system.
- CSADDR contains the starting address assigned to the control section currently being scanned by the loader. this value is added to all relative addresses within the control section to convert them to actual addresses.

During Pass 1 **the loader is concerned only with the header and define record.**

- The beginning load address for the linked program(PROGADDR) is obtained from the OS. This becomes the starting address(CSADDR) for the first control section in the inut sequence.
- The control section name from the header record is entered into ESTAB,with the value given by CSADDR.
- All external symbols appearing in the define record for the control section are also entered into ESTAB. Their addresses are obtained by adding value specified in the define record to CSADDR.
- When the END record is read, the control section length(CSLTH) is added to CSADDR. This gives the starting address for the next control section in the sequence.
- At the end of PASS1 the ESTAB contains all the external symbols defined in the set of control sections together with the address assigned to each.

```
PASS1
 begin
get PROGADDR from operating system
set CSADDR to program(for first control section)
while not end of input do
begin
    read next input record{header record for control section}
    set CSLTH to control section length
    search ESTAB for control section name
    if  found then
        set error flag {duplicate external symbol}
    else
    enter control section name into ESTAB with value CSADDR
    while  record type =’E  do
        begin
        read next input record
        if record type=’D’ then
        for  each symbol in the rercord do
        begin
            search ESTAB for symbol name
```

```
                    if  found then
                    set error flag{duplicate external symbol}
                    else
                    enter symbol into ESTAB with value
                    (CSADDR+indicated address)
                    end{for}

end{while≠'E'}
add CSLTH to CSADDR{starting address for next control section}
end {while not EOF}
end {pass1}
```

7. Explain the pass 2 algorithm of a linking loader.

Pass 2 :loader is concerned with TEXT and MODIFICATION record.

- Pass2 performs actual loading, relocation and linking of the program.CSADDR is used in the same way as Pass1.
- As each text record is read, object code is moved to the specified address(plus the current value of CSADDR).
- When a modification record is read, the symbol whose value is to be used for modification is looked up in ESTAB.this value is then added or subtracted from the indicated location in memory.
- The last step performed by the loader is usually transferring of control to the loaded program to begin execution.

```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
begin
    read next input record{header record for control section}
    set CSLTH to control section length
    while record type ≠'E  do
        begin
        read next input record
        if record type='T' then
        begin
            {if object code is in character form, convert into internal representation}
            Move object code from record to location
            {CSADDR + specified address}
        end {if 'T'}
    else if record type = 'M' then
    begin
        search ESTAB for modifying symbol name
        if found then
        add or subtract symbol value at location
        (CSADDR + specified address)
```

```
          else
          set error flag{undefined external symbol}
          end{if 'M'}
          end {while ≠'E'}
          if an address is specified in {in end record} then
           set EXECADDR to (CSADDR+ specified address)
          add CSLTH to CSADDR
          end{while not EOF}
jump to location given by EXECADDR{to start execution of loaded program}
end{pass2}
```