

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**SUBJECT NAME: LANGUAGE TRANSLATORS** **SUBJECT CODE: CST52**

**UNIT II**



**Loaders and Linkers:** Basic loader functions, machine–dependent and machine–independentloader features. Loader design – Linkage editors, dynamic linking and bootstrap loaders.

**2 MARKS**

1. **Define loader.** 
   * A loader is a system program that performs the loading function. It brings object program into memory and starts its execution. Many loaders also support relocation and linking.
   * Loader is a set of program that loads the machine language translated by the translator into the main memory and makes it ready for execution.
2. **Define linker or linkage editor.** 
   * Linker or linkage editor to perform the linking operations and a separate loader to handle relocation and loading.
3. **What are the two primary tasks of a linker? (NOV 2013)**

The two primary tasks of the linker are

* + 1. Relocating relative addresses
    2. Resolving external references

1. **What are the basic functions of loaders?** 
   * Loading – brings the object program into memory for execution.
   * Relocation – modifies the object program so that it can be loaded at an address different from the location originally specified.
   * Linking – combines two or more separate object programs and supplies the information needed to references between them.
2. **What are the types of loaders?**

The different types of loaders are

1. absolute loader
2. bootstrap loader
3. relocating loader or relative loader
4. linking loader.
5. **What is an absolute loader? State its disadvantages. (MAY 2013)** 
   * The loader, which is used only for loading, is known as absolute loader.
   * The operation of absolute loader is very simple. The object code is loaded to specified locations in the memory. At the end the loader jumps to the specified address to begin execution of the loaded program.
   * e.g. Bootstrap loader

**Disadvantages:**

The disadvantages are,

* 1. It is need for programmer to specify the actual address at which it will be loaded into memory.
  2. It is difficult to use subroutine libraries efficiently.

1. **What is the design of absolute loader?**

This loader does not perform such functions as linking and program relocation. Its operation is very similar to all functions are accomplished in a single pass.

* + The Header record is checked to verify that the correct program has been presented for loading.
  + Each Text record is read, the object code contains is moved to the indicated address in memory.
  + When the End record is encountered, the loader jumps to the specified address to begin execution of the loaded program.

1. **Write the 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 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**

1. **What is bootstrap loader? (MAY 2012)** 
   * When a computer is first tuned 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.

* + It then jumps to the just loaded program to execute it .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.

1. **What is program relocation?** 
   * Program relocation is the method of loading the object code in a different location specified in the object program. Assembler passes this information about the program relocation to the loader through modification record as a part of object program.
   * 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.
2. **What is address sensitive program? (NOV 2011)** 
   * The direct addressing mode is called address sensitive program.
   * When operands are specified in memory addressing mode, direct access to main memory, usually to the data segment, is required.
   * Direct addressing ***E.g. LDA ZERO***
3. **What are relocating loaders or relative loaders?**

Loaders that allow for program relocation are called relocating loaders or relative loaders.

1. **List out the functions performed by relocating loaders? (NOV 2011)**

The two functions performed by relocation loaders are

* + 1. Modification record and
    2. Relocation bit.
  + In modification record, a modification record M is used in the object program to specify any relocation.
  + In relocation bit, each instruction is associated with one relocation bit and, these relocation bits in a Text record is gathered into bit masks.

1. **What is the use of modification record?**

Modification record is used for program relocation. Each modification record specifies the starting address and the length of the field whose value is to be altered and also describes the modification to be performed.

1. **Define Relocation bit.** 
   * Relocation bit associated with each word of object code. In SIC instructions occupy one word; this means that there is one relocation bit for each possible instruction.
   * The relocation bit corresponding to a word of object code is set to 1, the program’s starting address is to be added to this word when the program is relocated.
   * Bit value 0 indicates no modification is required.

1. **Define bit mask.** 
   * The relocation bits are gathered together following the length indicator in each text record and which is called as bit mask.
   * For e.g. the bit mask FFC (111111111100) specifies that the first 10 words of object code are to be modified during relocation.
2. **What is the need of ESTAB?** 
   * External Symbol Table (ESTAB) is used to store the name and address of the each external symbol.
   * It also indicates in which control section the symbol is defined.
3. **What is the use of PROGADDR?** 
   * Program load 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.

**19**. **What is the use of CSADDR?**

* + Control Section Address (CSADDR) contains the starting address assigned to the control section currently being scanned by the loader.
  + Its value is added to all relative addresses within the control section to convert them to actual addresses.

1. **Write the two passes of a linking loader.**

The two passes of linking loaders are

* + - Pass1: assigns address to all external symbols.
    - Pass2: it performs the actual loading, relocation and linking.

1. **How are duplicate literal operands handled in machine independent loader? (NOV 2013)** 
   * The duplicate literal used more than once in the program. Only one copy of the specified value needs to be stored.

For how to recognize the duplicate literals

* + Compare the character strings defining them. Easier to implement, but has potential problem
  + ***E.g., =X’05’***

1. **Define automatic library search.**

In many linking loaders the subroutines called by the program being loaded are automatically fetched from the library, linked with the main program and loaded. This feature is referred to as automatic library search. Linking loaders that support automatic library search must keep track of external symbols that are referred to, but not defined, in the primary input to the loader.

**23. List the loader options INCLUDE &DELETE.**

The general format of INCLUDE is

**INCLUDE program\_name (library name)**

* This command directs the loader to read the designated object program from a library and treat it as the primary loader input.

The general format of DELETE command is

**DELETE csect-name**

* + It instructs the loader to delete the named control sections from the sets of programs loaded.

1. **What are the primary loader commands?**

The primary loader input with the loader commands are

* + INCLUDE READ (UTLIB)
  + INCLUDE WRITE (UTLIB)
  + DELETE RDREC, WRREC
  + CHANGE RDREC, READ
  + CHANGE WRREC, WRITE

1. **List the loader design options. 1.** Linkage Editors

**2.** Dynamic linking

**3.** Bootstrap loaders

1. **Give the functions of the linking loader.**

The linking loader performs all linking and relocation operations, including automatic library search and loads the linked programs directly loaded into the memory for execution.

1. **Define link editor? (MAY 2013)** 
   * A linkage editor produces a linked version of the program (often called a **load module** or an **executable image**), which is written to a file or library for later execution.
   * It performs relocation of all control sections relative to the start of the linked program.
2. **Why is linking required after a program is translated? (MAY 2013)** 
   * Linking which combines two or more separate object programs and supplies the information needed to allow references between them
   * The routines are automatically retrieved from a library as they are needed during linking.

**29. The difference between Linkage Editor and Linking Loader?**

|  |  |
| --- | --- |
| **Linking Loader** | **Linkage Editor** |
|  |  |
| A linking loader performs all linking and | A linkage editor produces a linked version of the |
| relocation operations, including automatic | program, which is normally written to a file for |
| library search, and loads the linked program | later execution. |
| into memory for execution. |  |
|  |  |
| Linkage editors perform linking operations | Linking loaders perform these same operations |
| before the program is loaded for execution | at load time. |
|  |  |

1. **Define dynamic linking.** 
   * Dynamic linking postpones the linking function until execution time; a subroutine is loaded and is linked to the rest of the program when it is first called(run time).This type of function is usually called

**dynamic loading** or **dynamic linking** or **load on call.**

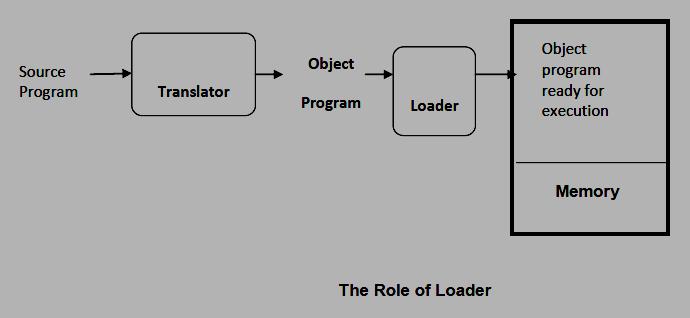
1. **Write the advantage of dynamic linking.** 
   * + Dynamic linking is used to allow several executing programs to share one copy of a subroutine or library.
     + It is often used for references to software object.
     + It has the ability to load the routine only when they are needed.
     + The dynamic linking avoids the loading of entire library for each execution.

* 1. **MARKS**

1. **Explain with neat block diagram the role of loader and linker. (11 marks)(MAY 2013)**

**Role of Loader**

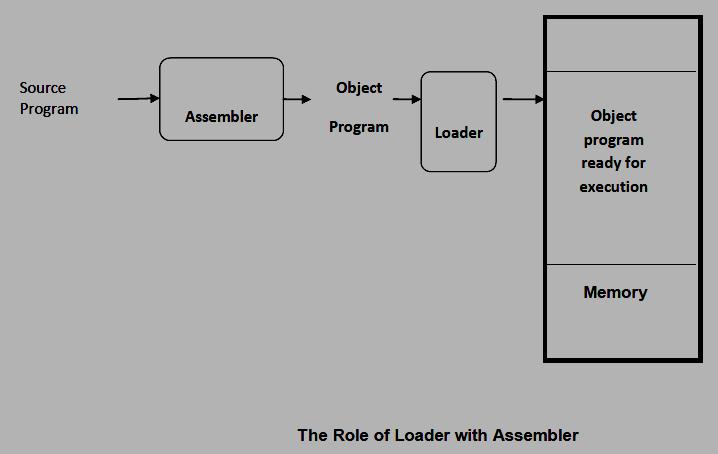
* A *loader* is a system program that performs the loading function. It brings object program into memory and starts its execution. Many loaders also support relocation and linking.
* Loader is a set of program that loads the machine language translated by the translator into the main memory and makes it ready for execution.
* An object program contains translated instructions and data values from the source program and specifies addresses in memory where these items to be loaded.
* The loader is a program which accepts the object program checks, prepares these programs for execution by the computer and initiates the execution.



The loader must perform the following functions:

* 1. *Loading*: This brings the object program into memory for execution.
  2. *Relocation:* This modifies the object program so that it can be loaded at an address from thelocation originally specified.
  3. *Linking:* This combines two or more separate object programs and supplies the informationneeded to allow references between them.
* A loader is a system program that performs the loading function. Many loaders also support relocation and linking. Some systems have a *linker or linkage editor* to perform the linking and a separate loader to handle relocation and loading.
* In most cases all the program translators (*assemblers and compilers*) on a particular system produce object programs.

**Role of loader with assembler:**



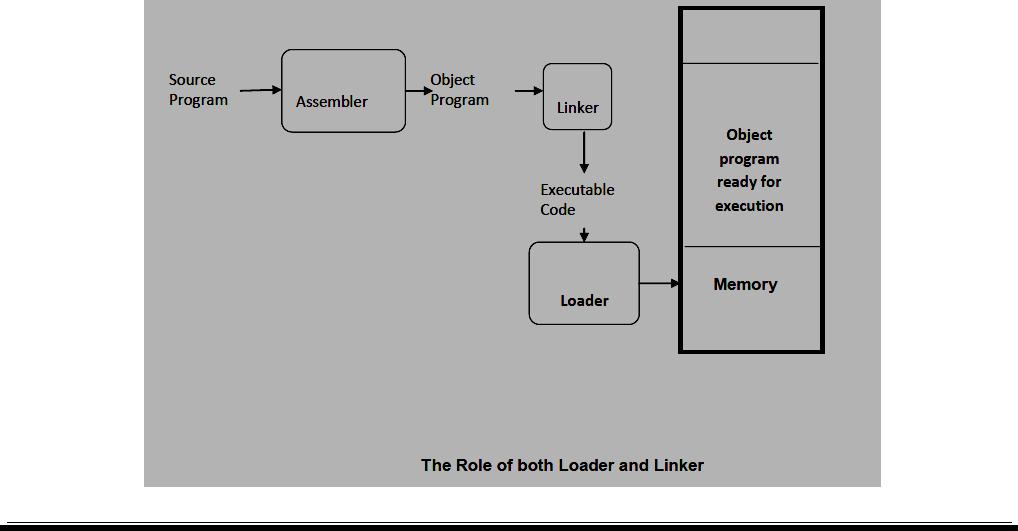
**Type of Loaders**

The different types of loaders are,

1. absolute loader
2. bootstrap loader
3. relocating loader or relative loader
4. linking - loader

**Role of loader and linker:**

* Linker combines two or more separate object programs and supplies the information needed to allow references between them
* Linker or linkage editor to perform the linking operations and a separate loader to handle relocation and loading.



**2. Discuss about Basic Loader Functions. (MAY 2012) or Discuss the functions and design of an absolute loader. (NOV 2011)**

**Basic loader functions:**

* The most fundamental function of the loader is bringing the object program into memory and starts its execution.
  1. Design of an absolute Loader
  2. A simple Bootstrap Loader

**Design of an Absolute Loader:**

This loader does not perform such functions as linking and program relocation, its operation is very simple. All functions are accomplished in a single pass.

* The Header record is checked to verify that the correct program has been presented for loading.
* As each Text record is read, the object code contains is moved to the indicated address in memory.
* When the End record is encountered, the loader jumps to the specified address to begin execution of the loaded program.

**Algorithm for an Absolute Loader:**

**begin**

read Header record

verify program name and length read first text record

**while**record type≠ ‘E’ **do begin**

{ if object code is in character form, convert into internal representation }

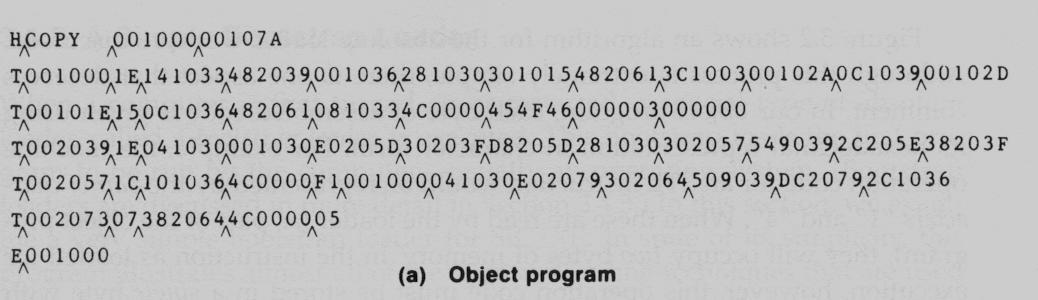
move object code to specified location in memory read next object program record

**end**

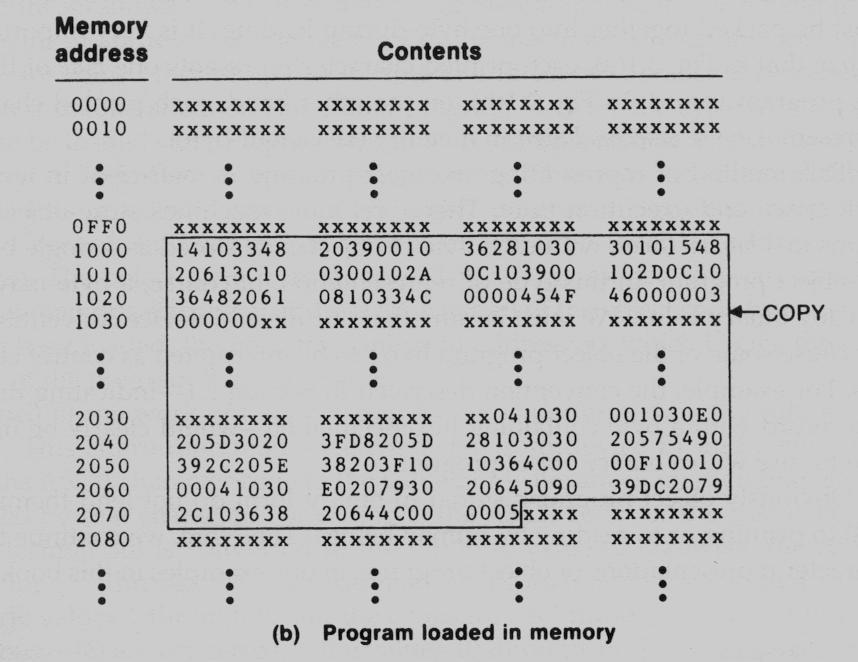
jump to address specified in End record

**end**

**Representation of object program:**



The contents of memory locations for which there is no text record as shown as **xxxx**



* In the object program, each byte of assembled code is given using its hexadecimal representation in character form.
* The operation code for an **STL** instruction would be represented by the pair of characters **“1”** and **“4”**.
* When these are read by the loader, it will occupy two bytes of memory.
* In the instruction as loaded for execution, however, this operation code must be stored in a single byte with hexadecimal value 14.
* Thus each pair of bytes from the object program record must be packed together into one byte during loading.

**Disadvantages of Absolute Loader**

* + The program must specify to the assembler the address in memory where the program is to be loaded.
  + If there are multiple subroutines, the programmer must remember the address of each and use that absolute address explicitly in his other subroutines to perform subroutine linkage.
  + The programmer must be careful not to assign two subroutines to the same or overlapping.

1. **Define bootstrap loader and state the algorithm. (NOV 2011) (MAY 2012, 2013)** 
   * When a computer is first tuned on or restarted, a special type of absolute loader, called *bootstraploader* is executed. This bootstrap loads the first program to be run by the computer–usually anoperating system.
   * The bootstrap itself begins at address 0 in the memory of the machine.
   * It loads the operating system starting at address 80.
   * No header record or control information, the object code is consecutive bytes of memory.

**Algorithm for bootstrap loader**

The algorithm for the bootstrap loader is as follows

**begin**

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

**Loop**

AGETC (and convert it from the ASCII character code to the value of the hexadecimal digit) save the value in the high-order 4 bits of S

AGETC

combine the value to form one byte A (A+S) store the value (in A) to the address in register X XX+1

**End**

It uses a subroutine GETC, which is **GETC** Aread one character

if A=0x04 then jump to 0x80 if A<48 then GETC

A  A-48 (0x30) if A<10 then return A  A-7

return



Each byte of object code to be loaded is represented on device F1 as two hexadecimal digits.

* + However, there is no Header record, End record or control information (such as addresses or lengths).
  + The object code from device F1 is always loaded into consecutive bytes of memory, starting at address 80.
  + After all of the object code from device F1 has been loaded, the bootstrap jumps to address 80, which begins the execution of the program that was loaded.

1. **Explain Machine Dependent Loader Features? (11 marks)**

The design and implementation of a more complex loader. This loader provides for program relocation and linking. For efficient sharing of the machines, we go for relocatable loader instead of absolute ones. This relocatable loader provides for program relocation and linking.

It contains

* 1. Program Relocation
  2. Program linking
  3. Algorithms and data structures

1. **PROGRAM RELOCATION:**

Loaders that allow for program relocation are called *relocating loaders or relative loaders*. There are two methods for specifying relocation as part of the object program.

* + 1. Using Modification Record.
    2. Using relocation bit mask.

1. **Program relocation using modification record:** 
   * A modification record is used to describe each part of the object code that must be changed when the program is relocated.
   * Most of the instructions in this program use relative or immediate addressing.
   * The only portions of the assembled program that contain actual addresses are the extended format instructions.
   * There is one modification record for each value that must be changed during relocation.
   * Each modification record specifies the starting address and length of the field whose value is to be altered.
   * It then describes the modification to be performed.

Eg**.M^ 000007 ^ 05 + COPY.**

* In this example, modification adds the value of the symbol COPY which represents the starting address of the program.
* The modification record scheme is a convenient means for specifying the program relocation.

**SIC/XE Relocation Loader algorithm begin**

get PROGADDR from operating system **while** not end of input **do**

**begin**

read next record

**while**record type ≠ ‘E’ **do begin**

read next input record **while** record type= ‘T’ **then**

**begin**

move object code from record to location ADDR + specified address. **end**

**while**record type = ‘M’

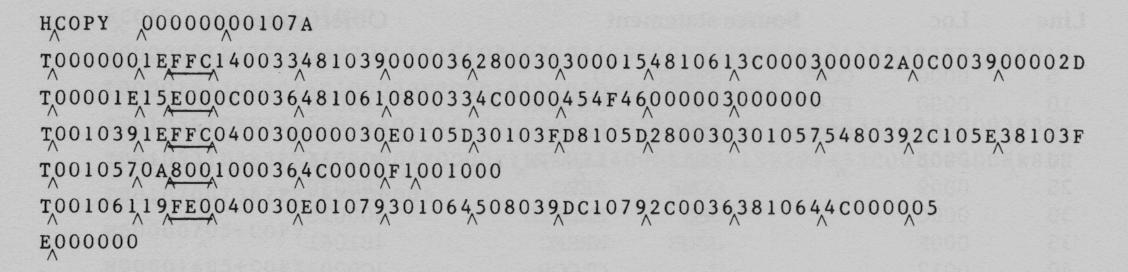
add PROGADDR at the location PROGADDR + specified address.

**end**

**end**

**end**

1. **Program relocation using Bit Mask** 
   * Relocation bit associated with each word of object code. In SIC instructions occupy one word; this means that there is one relocation bit for each possible instruction.
   * The relocation bit corresponding to a word of object code is set to 1, the program’s starting address is to be added to this word when the program is relocated.
   * Bit value 0 indicates no modification is required.
   * The relocation bits are gathered together following the length indicator in each text record and which is called as *bit mask.*



If a text record contains fewer than 12 words of object code, the bits corresponding to unused words are set to 0. Thus the bit mask FFC (111111111100) in the first text record specifies that all 10 words of the object code are to be modified during relocation. The mask E00 in the second text record specifies that the first 3 words are to be modified.

**SIC Relocation Loader Algorithm**

**begin**

get PROGADDR from operating system **while** not end of input **do**

**begin**

read next record **while** record type ≠ ‘E’ **do while** record type = ‘T’

**begin**

get length = second data mask bits(M) as third data

**for**( i=0 ; i< length; i++) **if** Mi= 1 **then**

add PROGADDR at the location PROGADDR + specified address

**else**

move object code from record to location PROGADDR + specified address.

read next record

**end**

**end**

**end**

1. **PROGRAM LINKING:** 
   * A program may be composed of many control sections.
   * These control sections may be assembled separately.
   * These control sections may be loaded at different addresses in memory.
   * External references to symbol defined in other control sections can only be resolved after these control sections are loaded into memory.

**Example**

* Let us consider three separately assembled programs, each of which consists of a single control section.
* Each program contains a list of items (LISTA, LISTB, LISTC).
* The ends of these lists are marked by the labels ENDA, ENDB, ENDC.

The labels on the beginnings and ends of the lists are external symbols. That is they are available for use in linking. Each program contains exactly the same set of references to these external symbols.

**External Symbols**

* ***EXTDEF (external definition)*** - The EXTDEF statement in a control section names symbols, calledexternal symbols, that are defined in this (present) control section and may be used by other sections. **ex:** EXTDEF LISTA, ENDA

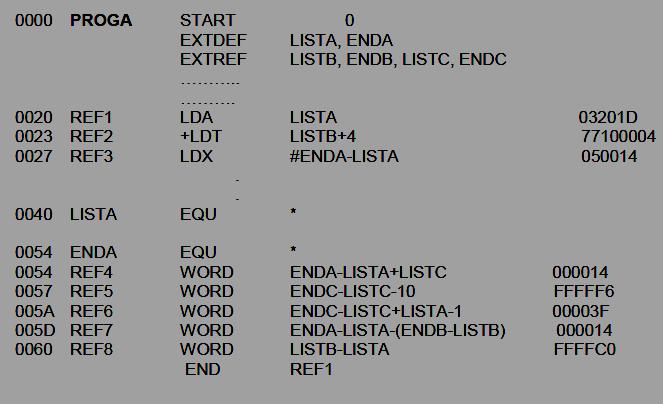
EXTDEF BUFFER, BUFFEND, LENGTH

* ***EXTREF (external reference)*** - The EXTREF statement names symbols used in this (present) controlsection and are defined elsewhere.

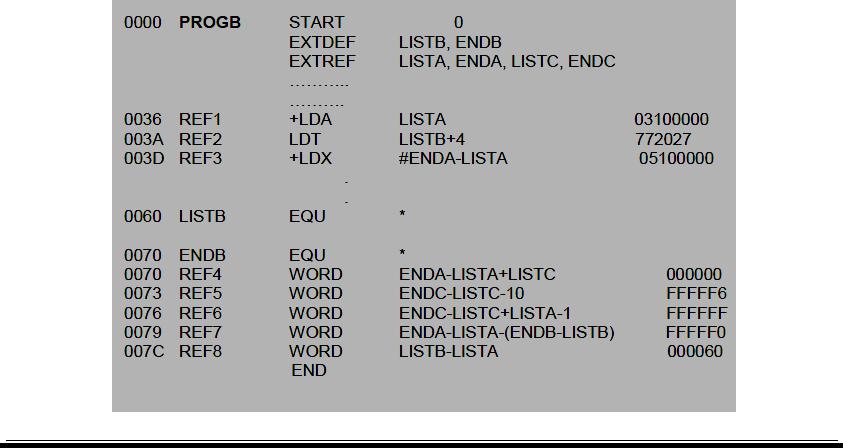
**ex:** EXTREF LISTB, ENDB, LISTC, ENDCEXTREF RDREC, WRREC

**Example in program linking**

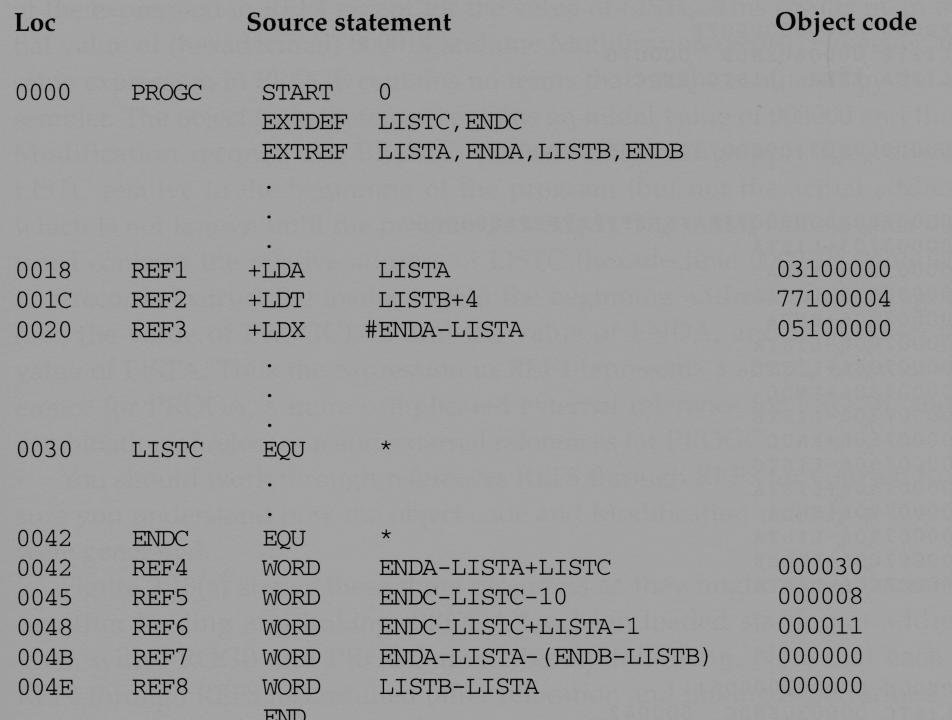
**Program A**



**Program B**

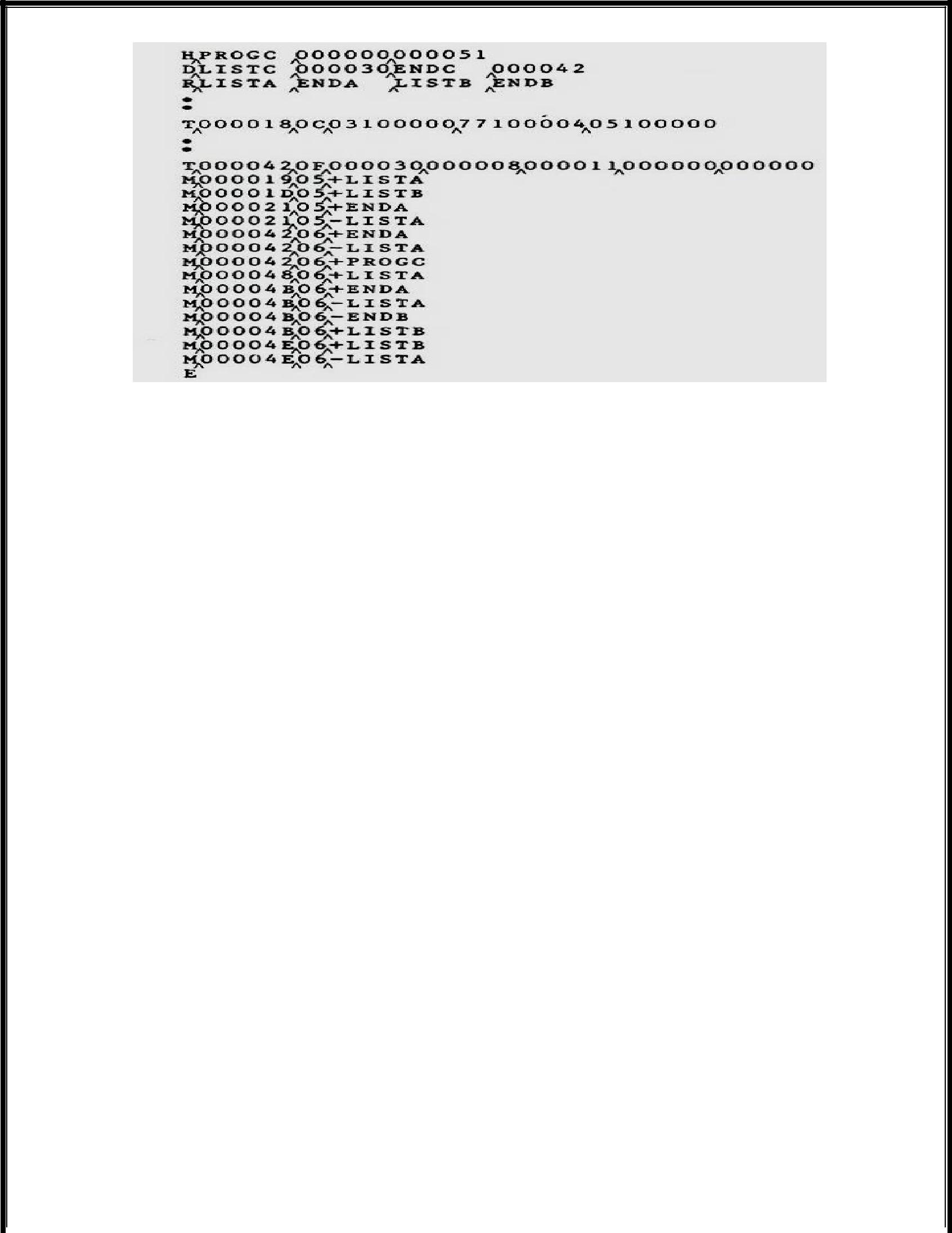


**Program C**



**Object Program Example**





* Notice that program A defines LISTA and ENDA, program B defines LISTB and ENDB, and program defines LISTC and ENDC.
* Notice that the definitions of REF1, REF2, .., to REF7 in all of these three control sections are the same.
* Therefore, after these three control sections are loaded, no matter where they are loaded, the values of REF1 to REF7 in all of these programs should be the same.

**REF 1 LISTA**

* Program A

– LISTA is defined in its own program and its address is immediately available. Therefore, wecan simply use program counter-relative addressing

* Program B

– Because LISTA is an external reference, its address is not available now. Therefore anextended-format instruction with address field set to 00000 is used. A modification record in inserted into the object code so that once LISTA’s address is known, it can be added to this field.

* Program C

– The same as that processed in Program B.

**REF 2 LISTB + 4**

* Program A

– Because LISTB is an external reference, its address is not available now. Therefore anextended-format instruction with address field set to 00004 is used. A modification record is inserted into the object code so that once LISTB’s address is available; it can be added to this field.

* Program B

– LISTB is defined in its own program and its address is immediately available. Therefore, wecan simply use program counter-relative addressing

* Program C

– The same as that processed in Program A.

**REF 3 # ENDA - LISTA**

* Program A

– The difference between ENDA and LISTA (14) is immediately available during assembly.

* Program B

– Because the values of ENDA and LISTA are unknown during assembly, we need to use anextended-format instruction with its address field set to 0.

– Two modification records are inserted to the object program–one for +ENDA and the otherfor –LISTA.

* Program C

– The same as that processed in Program B.

**REF4 ENDA – LISTA + LISTC**

* Program A

– The difference between ENDA and LISTA can be known now. Only the value of LISTC isunknown. Therefore, an initial value of 000014 is stored with one modification record for LISTC.

* Program B

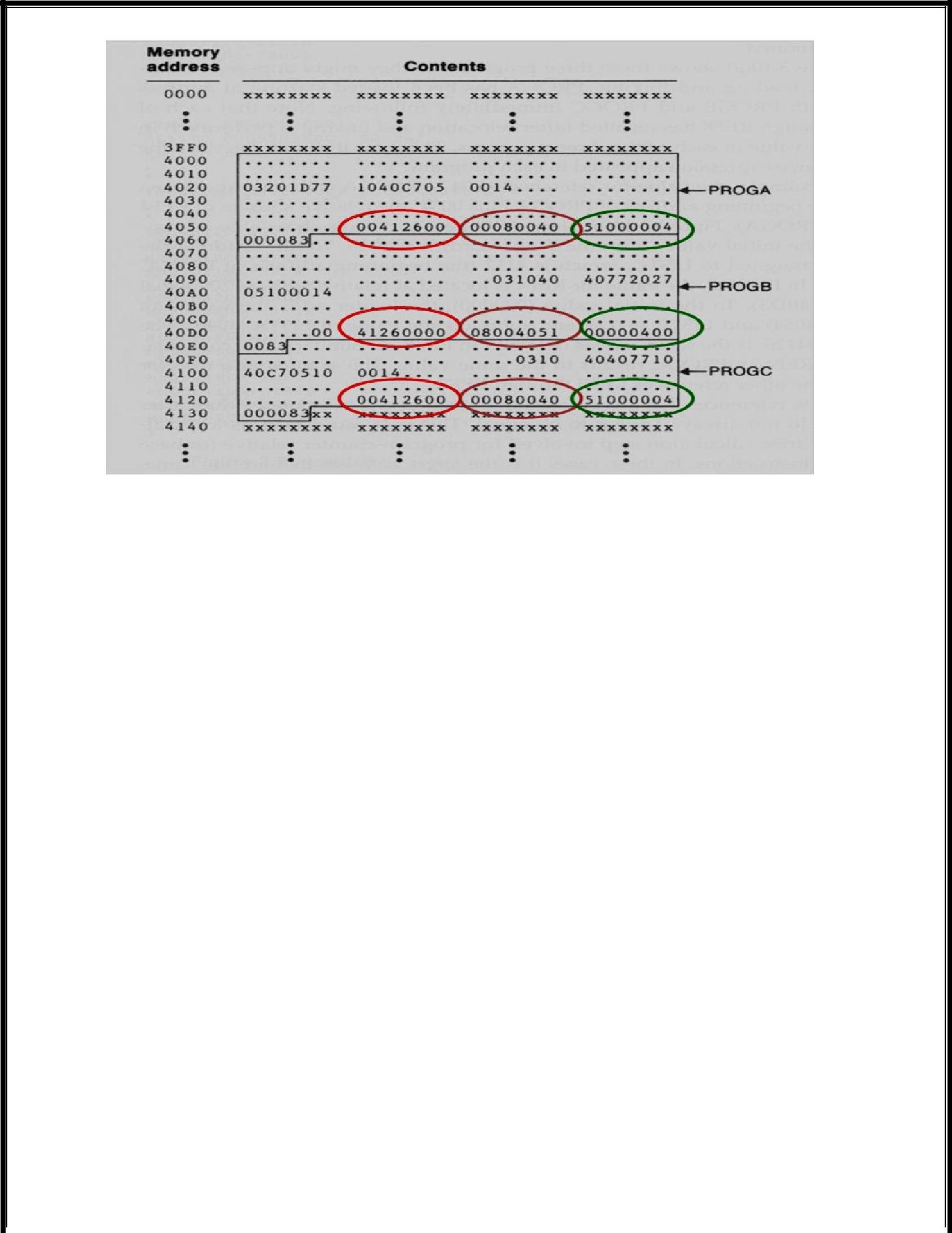
– Because none of ENDA, LISTA, and LISTC’s values can be knownnow, an initial value of000000 is stored with three modification records for all of them.

* Program C

– The value of LISTC is known now. However, the values for ENDA and LISTA are unknown. Aninitial value of 000030 is stored with two modification records for ENDA and LISTA.

**After Loading into Memory**

* Suppose that program A is loaded at 004000, program B at 004063, and program C at 0040E2.
* The REF4, REF5, REF6, and REF7 in all of these three programs have the same values.



**REF4 AFTER LINKING**

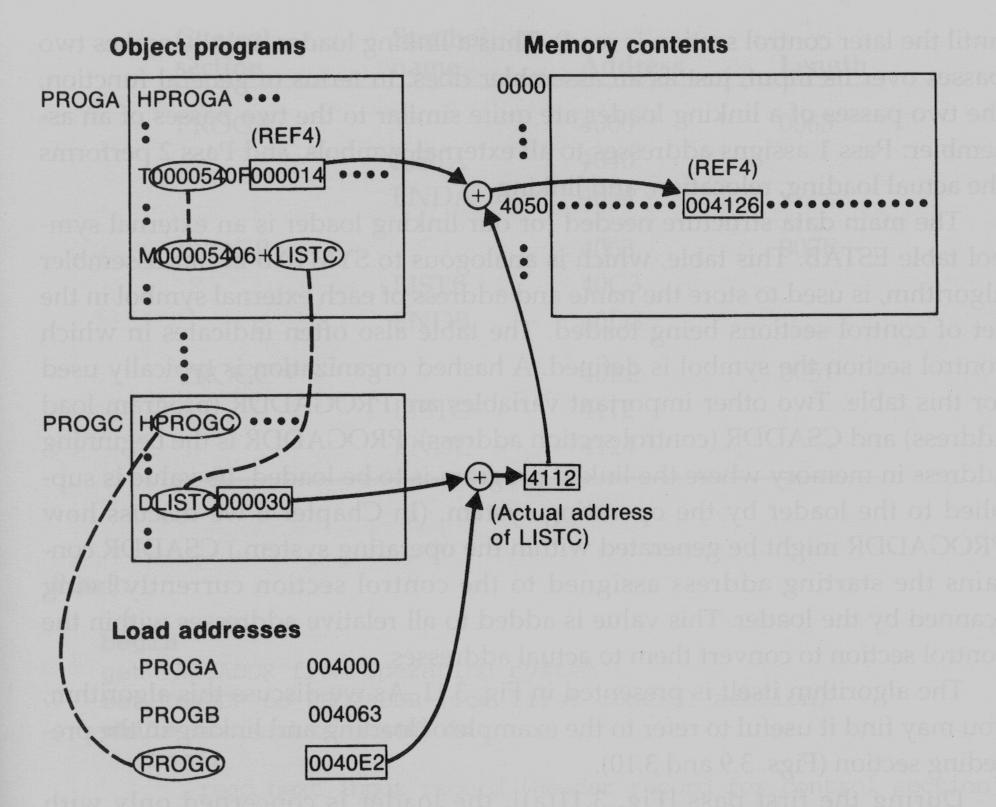
* **In Program A**

– The address of REF4 is 4054 (4000 + 54) because program A is loaded at 4000 and the relativeaddress of REF4 within program A is 54.

– The value of REF4 is 004126 because

The address of LISTC is 0040E2 (the loaded address of program C) + 000030 (the relative address of LISTC in program C)

0040E2 + 000014 (constant already calculated) = 004126.



* + **In Program B**

– The address of REF4 is 40D3 (4063 + 70) because program B is loaded at 4063 and the relativeaddress of REF4 within program A is 70.

– The value of REF4 is 004126 because

* + - * The address of LISTC is 004112
      * The address of ENDA is 004054
      * The address of LISTA is 004040
      * 004054 + 004112 – 004040 = 004126

1. **ALGORITHM AND DATASTRUCTURES OF A LINKING LOADER:**

**Implementation of Loader and Linker**

A linking loader makes two passes over its input

* In pass 1: assign addresses to external references
* In pass 2: perform the actually loading, relocation, and linking

**Data Structures**

* **External symbol table (ESTAB)**

– Like SYMTAB, store the name and address of each external symbol in the set of control sectionsbeing loaded.

– It needs to indicate in which control section the symbol is defined.

* **Program load Address (PROGADDR)**

– The beginning address in memory where the linked program is to be loaded.

– Its value is supplied to the loader by the operating system.

* **Control Section Address (CSADDR)**

– It contains the starting address assigned to the control section currently being scanned by theloader.

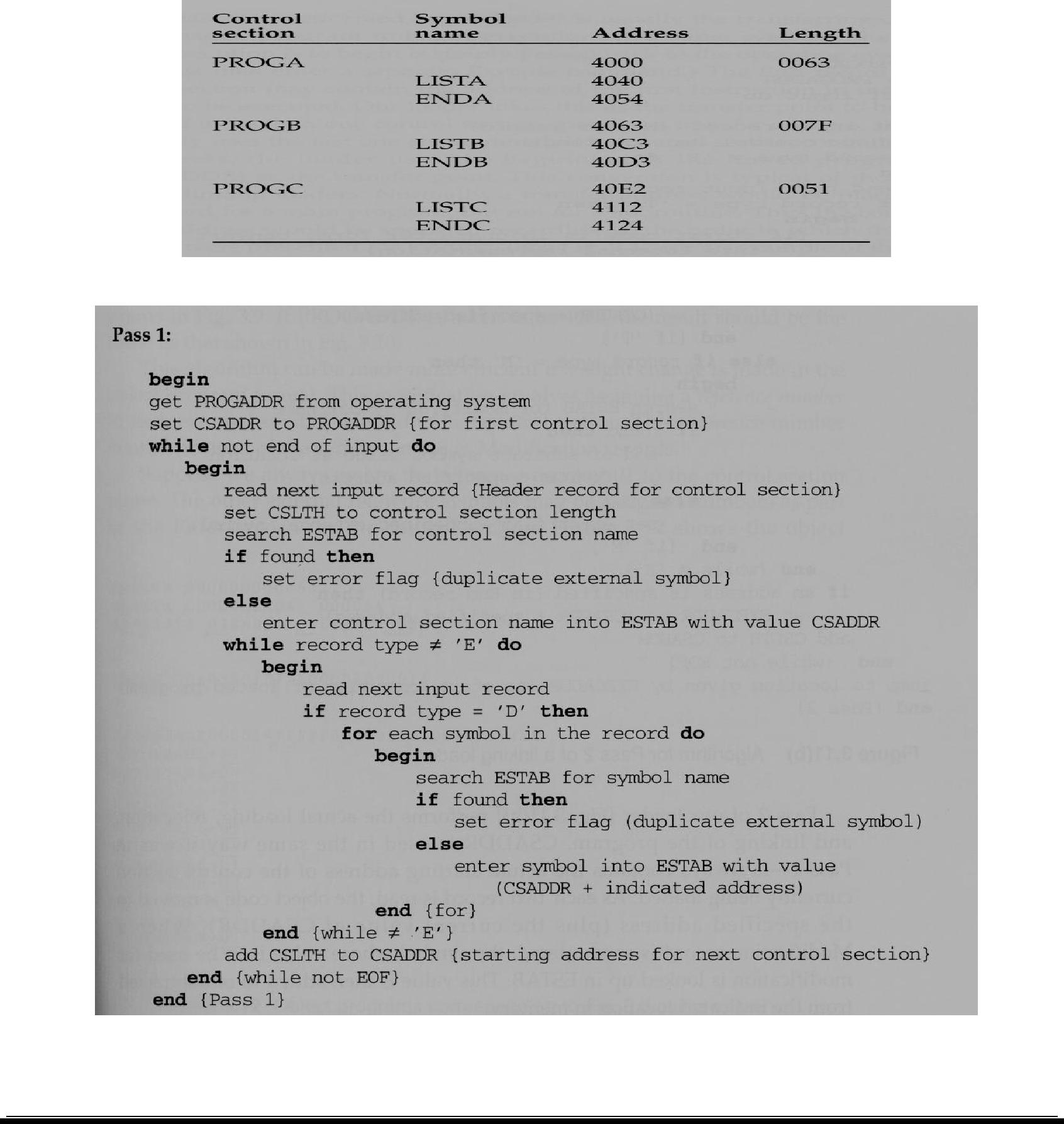
– This value is added to all relative addresses within the control sections.

**Algorithm – PASS1**

* During pass 1, the loaders are concerned only with HEADER and DEFINE record types in the control sections to build ESTAB.
* PROGADDR is obtained from OS.
* This becomes the starting address (CSADDR) for the first control section.
* The control section name from the header record is entered into ESTAB, with value given by CSADDR.
* All external symbols appearing in the DEFINE records for the current control section are also entered into ESTAB.

* Their addresses are obtained by adding the value (offset) specified in the DEFINE to CSADDR.
* At the end, ESTAB contains all external symbols defined in the set of control sections together with the addresses assigned to each.
* A Load Map can be generated to show these symbols and their addresses.

**A Load Map**

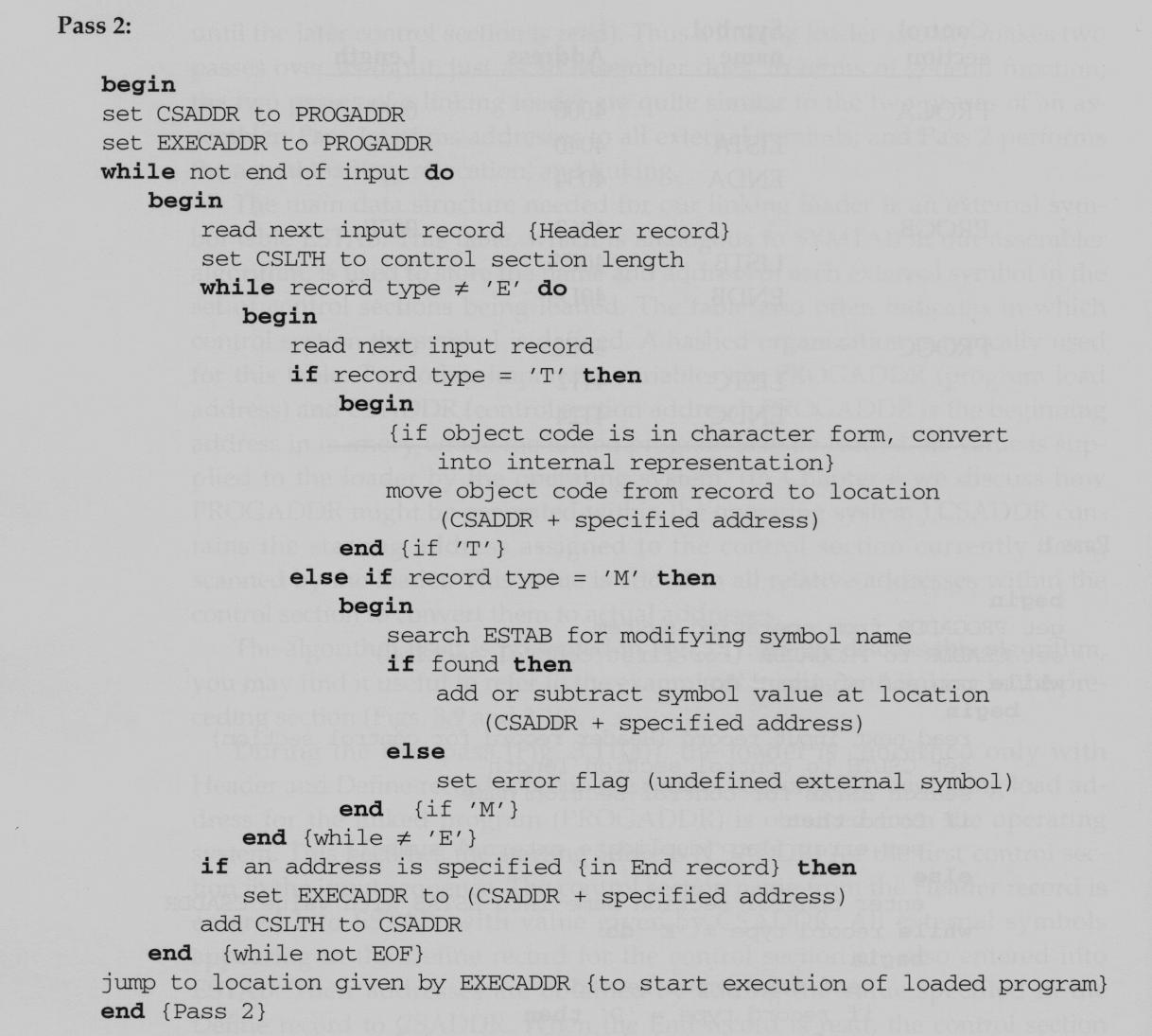


**Algorithm - PASS 2**

* During pass 2, the loader performs the actual loading, relocation, and linking.
* CSADDR is used in the same way as it was used in pass 1

– It always contains the actual starting address of the control section being loaded.

* As each text record is read, the object code is moved to the specified address (plus CSADDR).
* When a modification record is encountered, the symbol whose value is to be used for modification is looked up in ESTAB.
* This value is then added to or subtracted from the indicated location in memory.

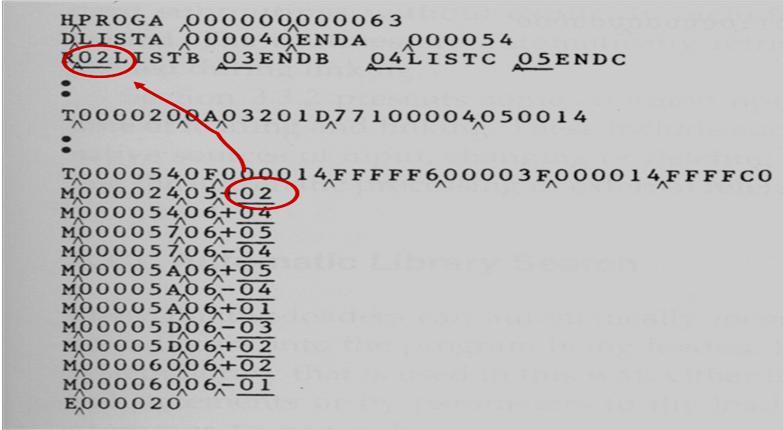


**Reference Number**

* The linking loader algorithm can be made more efficient if we assign a reference number to each external symbol referred to in a control section.
* This reference number is used (instead of the symbol name) in modification record.
* These simple techniques avoid multiple searches of ESTAB for the same symbol during the loading of a control section.

– After the first search for a symbol (the REFER records), we put the found entries into an array.

– Later in the same control section, we can just use the reference number as an index into thearray to quickly fetch a symbol’s value.



1. **Explain the machine independent loader features. (11 marks) (MAY, NOV 2012)(NOV 2013)**

Machine-independent Loader Features are

* + Automatic Library Search
  + Loader Options

**Automatic Library Search:**

* The use of automatic library search process for handling external references. This feature allows a programmer to use standard subroutines without explicitly including them in the program to be loaded. The routines are automatically retrieved from a library as they are needed during linking.
* Many linking loaders can automatically incorporate routines from a subprogram library into the program being loaded.
* In most cases there is a standard system library. (E.g., the standard C library)
* Other libraries may be specified by control statements or by parameters to the loader.

* This feature allows the programmer to use subroutines from one or more libraries( mathematical or statistical routines) are a part of the programming language.
* The subroutines called by the program are automatically fetched from the library, linked with the main program, and loaded.
* The programmer does not need to take any action beyond mentioning the subroutine names as external references in the source program. On some systems, this feature is referred to as automatic library call.
* Linking loader that 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 searches the library for routines that contain the definitions of these symbols, and processes the subroutines found by this search process exactly as if they had been part of the primary

input stream.

* The subroutines fetched from a library in this way may themselves contain external references. It is necessary to repeat the library search process until all references are resolved.
* If unresolved references remain after the library search is completed, they are treated as errors.
* If a symbol (or a subroutine name) is defined both in the source program and in the library, the one in the source program is used first.
* A programmer can make his own library easily on UNIX by using the “ar” command.

**Loader Options:**

* Many loaders allow the user to specify options that modify the standard processing.
* It include such capabilities as a specifying alternative sources of input, changing or deleting external references and controlling the automatic processing of external references.
* The loader option allows the selection of alternative sources of input.

For example, the command

**– INCLUDE program-name (library name)**

* might direct the loader to read the designated object program from a library program from a library.

**– DELETEcsect-name**

* might instruct the loader to delete the named control sections from the set of programs being loaded.

**– CHANGE name1, name2**

* might cause the external symbol name1 to be changed to name2 wherever it appears in the object program.

The primary loader input with the loader commands

**LIBRARY UTLIB**

**INCLUDE** **READ (UTLIB)**

**INCLUDE** **WRITE (UTLIB)**

**DELETE** **RDREC, WRREC**

**CHANGE** **RDREC, READ**

**CHANGE** **WRREC, WRITE**

**NOCALL** **STDDEV, PLOT, CORREL**

* + The commands are, use UTLIB (utility library), include READ and WRITE control sections from the library, delete the control sections RDREC and WRREC from the load, the change command causes all external references to the symbol RDREC to be changed to the symbol READ.
  + Similarly a reference to WRREC is changed to WRITE.
  + Finally, no call to the functions SQRT, PLOT, if they are used in the program.

1. **Explain the different options of loader design. (11 marks) (NOV 2012, 2013)**

The different loader design options are

* + 1. Linkage Editors
    2. Dynamic Linking
    3. Bootstrap Loaders

1. **LINKAGE EDITORS** 
   * A linking loader performs all linking and relocation operations including automatic library search and loads the linked program directly into memory for execution.
   * A ***linkage editor*** produces a linked version of the program (often called a ***load module*** or an ***executable image***), which is normally written to a file for later execution.
   * 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 an actual address to relative values within the program.
   * The linkage editor performs relocation of all control sections relative to the start of the linked program.
   * All items that need to be modified at load time have values that are relative to the start of the linked program. This means that the loading can be accomplished in one pass with no external symbol table required.

* If a program is to be executed many times without being reassembled, the use of a linkage editor substantially reduces the overhead required.
* Resolution of external references and library searching are only performed once.
* A linking loader searches libraries and resolves external references every time the program is executed.

**Advantages**

* Linkage editors can perform many useful functions besides simply preparing an object program for execution.
* For example a program (PLANNER) that uses a large number of subroutines.
* Suppose that one sub-routine (PROJECT) used by the program (PLANNER) is changed to correct an error or to improve efficiency.
* After the new version of PROJECT is assembled or compiled, the linkage editor can be used to replace this subroutine in the linked version of the PLANNER.

The linkage editor commands.

|  |  |  |
| --- | --- | --- |
| **INCLUDE** | **PLANNER (PROGLIB)** |  |
| **DELETE** | **PROJECT** | {DELETE from existing PLANNER} |
| **INCLUDE** | **PROJECT (NEWLIB)** {INCLUDE new version} | |
| **REPLACE** | **PLANNER (PROGLIB)** |  |

* Linkage editors can be used to build packages of subroutines or other control sections that are generally used together.
* This can be useful when dealing with subroutines libraries that support high level programming languages.

The linkage editor also used to combine the appropriate subroutines into a package with a FORTRAN command sequence like

**INCLUDE READR (FTNLIB)**



***Data blocks***

**INCLUDE WRITER (FTNLIB)**



**INCLUDE BLOCK (FTNLIB)**

***Records***

**INCLUDE DEBLOCK (FTNLIB)**

**INCLUDE ENCODE (FTNLIB)**



***Data items***

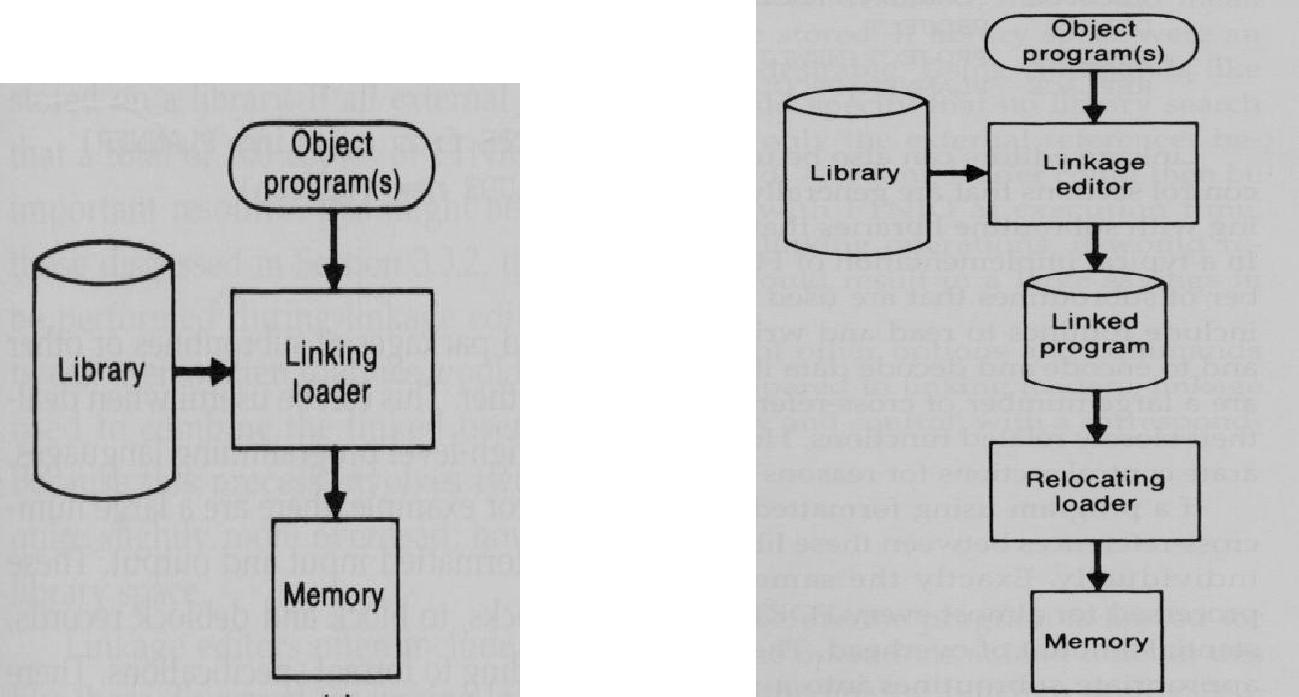
**INCLUDE DECODE (FTNLIB)**

**.**

**.**

**SAVE FTNIO (SUBLIB)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | | |
|  |  | | | |  |
|  | **The difference between a Linkage Editor and a Linking Loader** | | | |  |
|  |  |  |  |  |  |
|  |  | **Linking Loader** | **Linkage Editor** |  |  |
|  |  |  |  |  |  |
|  |  | A linking loader performs all linking and | A linkage editor produces a linked version of the |  |  |
|  |  | relocation operations, including automatic | program, which is normally written to a file for |  |  |
|  |  | library search, and loads the linked program | later execution. |  |  |
|  |  | into memory for execution. |  |  |  |
|  |  |  |  |  |  |
|  |  | Linkage editors perform linking operations | Linking loaders perform these same operations |  |  |
|  |  | before the program is loaded for execution. | at load time. |  |  |
|  |  |  |  |  |  |
|  | **Processing of an object program using** | |  |  |  |
|  | **a) Linking Loader** | | **b) Linkage Editor** | |  |
|  |  |  |  |  |  |



**(ii) DYNAMIC LINKING**

Dynamic linking postpones the linking function until execution time; a subroutine is loaded and linked to the test of the program when it is first called. This type of function is usually called ***dynamic linking,dynamic loading or load on call.***

**Dynamic linking applications**

* Dynamic linking is often used to allow several executing programs to share one copy of a subroutine or library.
* For example, run-time support routines for a high level language like C could be stored in ***a dynamiclink library.***
* A single copy of the routines in this library could be loaded into memory.
* In an object-oriented system, dynamic linking is often used for references to software objects.
* This allows the implementation of the object and its method to be determined at the time the program is run. (e.g., C++)
* This implementation can be changed at any time, without affecting the program that makes use of the object.
* Dynamic linking also makes it possible for one object to be shared by several programs.

**Dynamic Linking Advantage**

* The program contains subroutines that correct or diagnose errors in the input data during execution.
* However, the program were completely linked before execution, these subroutines must be loaded and linked every time the program is run.
* Dynamic linking provides the ability to load the routines only when they are needed. If the subroutines involved large or many external references, it can save both time and memory space.

**Dynamic Linking Implementation**

* The number of different mechanisms that can be used to accomplish the actual loading and linking called subroutine.
* A subroutine that is to be dynamically loaded must be called via an operating system service request.
* This method can also be thought of as a request to a part of the loader that is kept in memory during execution
* n of the program.

**(a).**Instead of executing a JSUB instruction to an external symbol, the program makes a load-and-call servicerequest to the OS.The parameter of this request is the symbolic name of the routine to be called.

**(b).**The OS examines its internal tables to determine whether the subroutine is already loaded. If needed, thesubroutine from the specified user or system libraries.

**(c).**Control is passed from the OS to the subroutine being called.

**(d).**When the called subroutine completes its processing, it returns to its caller (operating system).The OSthen returns control to the program that issues the request.

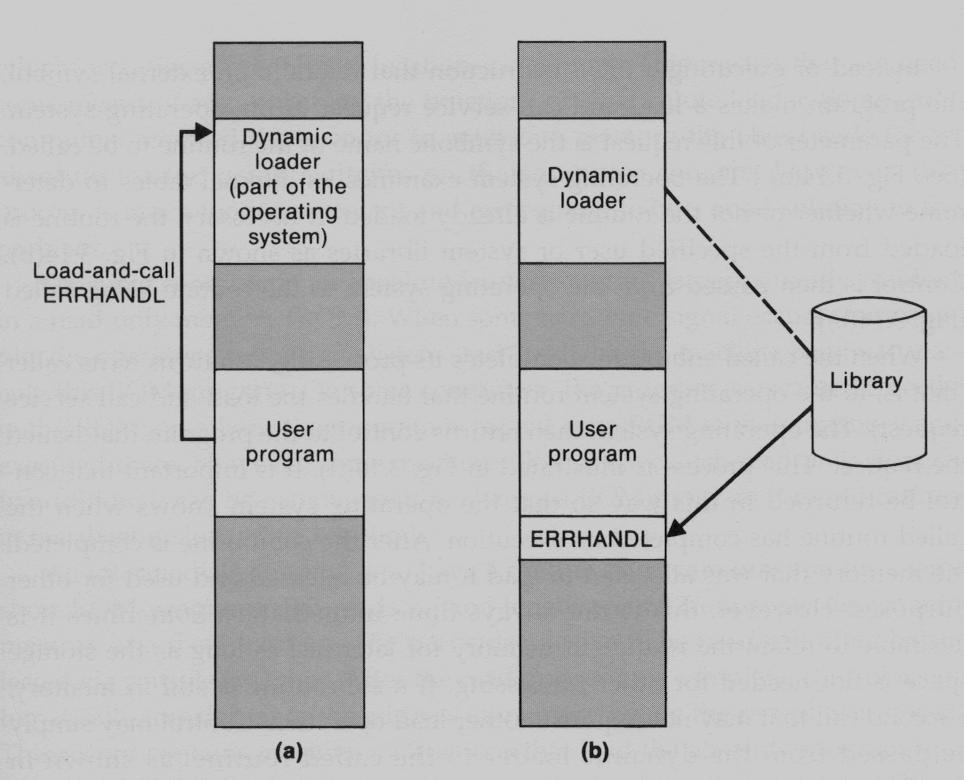
* After the subroutine is completed, the memory that was allocated to it may be released.
* However, often this is not always done immediately. If the subroutine is retained in memory, it can be used by later calls to the same subroutine without loading the same subroutine multiple times.

**(e).**Control can simply be passed from the dynamic loader to the called routine directly.

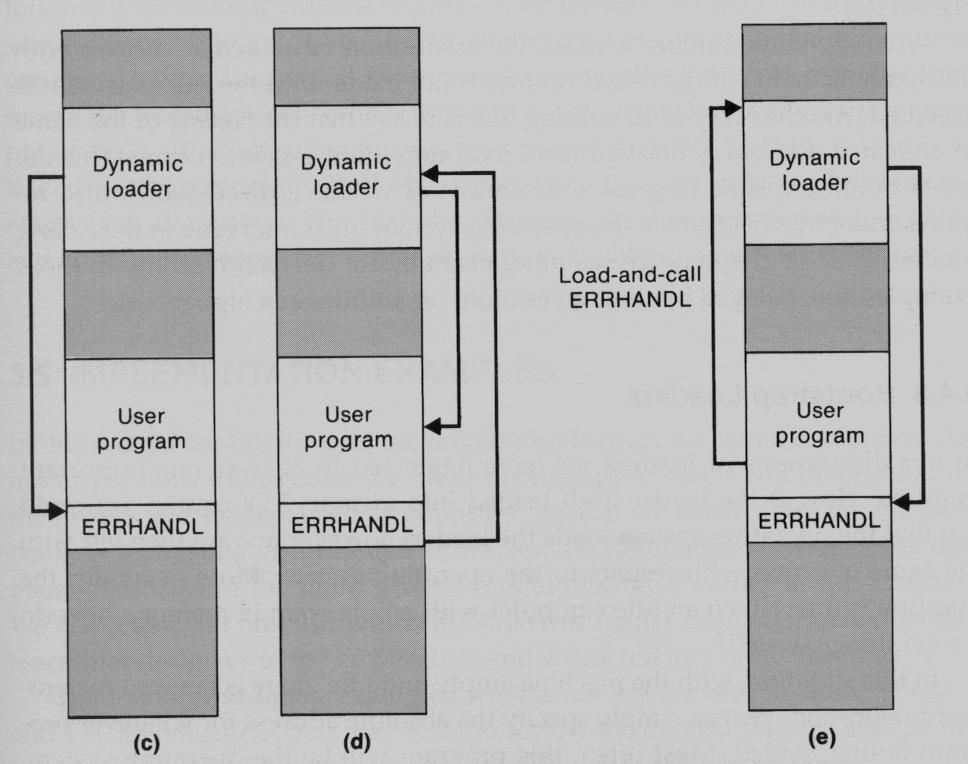
**Implementation Example:**

**Issue a Load and Call** **Load the called subroutine**

**service request** **into memory**



|  |  |  |
| --- | --- | --- |
| **Control is passed** | **Control is returned to** | **The called subroutine** |
| **to the loaded** | **the loader and later** | **this time is already loaded** |
| **Subroutine** | **returned to the user** | **program** |



1. **Bootstrap loaders** 
   * The operating system loads the loader. When computer is started – with no program in memory, a program present in ROM (absolute address) can be made executed – may be OS itself or a Bootstrap loader, which in turn loads OS and prepares it for execution.
   * The first record (or records) is generally referred to as a ***bootstrap loader*** – makes the OS to be loaded.
   * Such a loader is added to the beginning of all object programs that are to be loaded into an empty and idle system.

**PONDICHERRY UNIVERSITY QUESTIONS**

* 1. **MARKS**

1. What is address sensitive program? **(NOV 2011) (Ref.Qn.No.11, Pg.no.4)**
2. List out the functions performed by relocating loaders? **(NOV 2011) (Ref.Qn.No.13, Pg.no.4)**
3. What is Bootstrap Loader? **(MAY 2012) (Ref.Qn.No.9, Pg.no.3)**
4. What is an absolute loader? State its disadvantages. **(MAY 2013) (Ref.Qn.No.6, Pg.no.2)**
5. Why is linking required after a program is translated? Define link editor. **(MAY 2013)**

**(Ref.Qn.No.27,28 Pg.no.6)**

1. What are the two primary tasks of a linker? **(NOV 2013) (Ref.Qn.No.3, Pg.no.2)**
2. How are duplicate literal operands handled in machine independent loader? **(NOV 2013)**

**(Ref.Qn.No.21, Pg.no.5)**

**11 MARKS**

**NOV 2011(REGULAR)**

**1.** Discuss the functions and design of an absolute loader. **(Ref.Qn.No.2, Pg.no.10)**

**(OR)**

1. (a) What is dynamic linking? Explain (6) **(Ref.Qn.No.6, Pg.no.29)**
   1. Write about bootstrap loaders. (5) **(Ref.Qn.No.3, Pg.no.12)**

**MAY 2012(ARREAR)**

1. Discuss about Basic Loader Functions. **(Ref.Qn.No.2, Pg.no.10)**

**(OR)**

**2.** Explain the machine independent loader features. **(Ref.Qn.No.5, Pg.no.24)**

**NOV 2012(REGULAR)**

**1.** Discuss about Machine independents loader features. **(Ref.Qn.No.5, Pg.no.24)**

**(OR)**

**2.** Explain the different options of loader design. **(Ref.Qn.No.6, Pg.no.26)**

**MAY 2013(ARREAR)**

**1.** Define bootstrap loader and state the algorithm for the same. **(Ref.Qn.No.3, Pg.no.12)**

**(OR)**

**2.** Explain with neat block diagram the role of loader and linker. **(Ref.Qn.No.1, Pg.no.8)**

**NOV 2013 (REGULAR)**

**1.** Explain the loader design options. **(Ref.Qn.No.6, Pg.no.26)**

**(OR)**

1. (a) Discuss the machine independent loader features in detail.(6) **(Ref.Qn.No.5, Pg.no.24)**

(b) State and describe the significance of dynamic linking. (5) **(Ref.Qn.No.6, Pg.no.29)**