Chapter 3 Loaders and Linkers

Outline

- 3.1 Basic Loader Functions
- 3.2 Machine-Dependent Loader Features
- 3.3 Machine-Independent Loader Features
- 3.4 Loader Design Options
- 3.5 Implementation Examples

Introduction

- *Loading*
  - Brings the object program into memory for execution
- *Relocation*
  - Modify the object program so that it can be loaded at an address different from the location originally specified
- *Linking*
  - Combine two or more separate object programs and supplies the information needed to allow references between them

Absolute loader

Loader

Linking loader

Linker

Overview of Chapter 3

- Type of loaders
  - Assemble-and-go loader
  - Absolute loader (bootstrap loader)
  - Relocating loader (relative loader)
  - Direct linking loader
- Design options
  - Linkage editors
  - Dynamic linking
  - Bootstrap loaders

3.1 Basic Loader Functions

- The most fundamental functions of a loader:
  - Bringing an object program into memory and starting its execution
- Design of an Assemble-and-Go Loader
- Design of an Absolute Loader
- A Simple Bootstrap Loader

3.1.0 Assemble-and-Go Loader

- Characteristic
  - The object code is produced directly in memory for immediate execution after assembly
- Advantage
  - Useful for program development and testing
- Disadvantage
  - Whenever the assembly program is to be executed, it has to be assembled again
  - Programs consist of many control sections have to be coded in the same language

3.1.1 Design of an Absolute Loader

- Absolute Program (e.g. SIC programs)
  - Advantage
    - Simple and efficient
  - Disadvantages
    - The need for programmer to specify the actual address at which it will be loaded into memory
    - Difficult to use subroutine libraries efficiently
- Absolute loader only performs *loading* function
  - Does not need to perform *linking* and *program relocation*.
  - All functions are accomplished ***in a single pass***.

Design of an Absolute Loader (Cont.)

- In a single pass
  - Check the Header record for program name, starting address, and length
  - Bring the object program contained in the Text record to the indicated address
  - No need to perform program linking and relocation
  - Start the execution by jumping to the address specified in the End record

Loading of an Absolute Program (Fig 3.1 a)

- Object program contains
  - H record
  - T record
  - E record

```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036
T^002073^07^382064^4C0000^05
E^001000
```

(a) Object program

Loading of an Absolute Program (Fig 3.1 b)

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

Algorithm for an Absolute Loader (Fig. 3.2)

```
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
```

E.g., convert the pair of characters "14" (two bytes) in the object program to a single byte with hexadecimal value 14

Figure 3.2 Algorithm for an absolute loader.

Object Code Representation

- Figure 3.1 (a)
  - Each byte of assembled code is given using its hexadecimal representation in ***character*** form
    - For example, 14 (opcode of STL) occupies two bytes of memory
    - Easy to read by human beings
  - Each pair of bytes from the object program record must be *packed together into one byte during loading.*
    - Inefficient in terms of both space and execution time

- Thus, most machine store object programs in a *binary form*

3.1.2 A Simple Bootstrap Loader

- Bootstrap Loader
  - When a computer is first turned on or restarted, a special type of absolute loader, called a ***bootstrap loader*** is executed
    - In PC, BIOS acts as a bootstrap loader
  - This bootstrap loads the first program to be run by the computer -- usually an operating system

A Simple Bootstrap Loader (Cont.)

- Example: a simple SIC/XE bootstrap loader (Fig. 3.3)
  - The bootstrap itself begins at address 0 in the memory of the machine
  - It loads the OS (or some other program) starting address 0x80
    - No Header record, End record, or control information.
    - The object code from device F1 is always loaded into consecutive bytes of memory, starting at address 80.
  - After all the object code from device F1 has been loaded, the bootstraps jumps to address 80, which begins the execution of the program that was loaded.

Bootstrap loader for SIC/XE (Fig. 3.3)

```
BOOT     START    0        BOOTSTRAP LOADER FOR SIC/XE
.
. THIS BOOTSTRAP READS OBJECT CODE FROM DEVICE F1 AND ENTERS IT
. INTO MEMORY STARTING AT ADDRESS 80 (HEXADECIMAL). AFTER ALL OF
. THE CODE FROM DEVF1 HAS BEEN SEEN ENTERED INTO MEMORY, THE
. BOOTSTRAP EXECUTES A JUMP TO ADDRESS 80 TO BEGIN EXECUTION OF
. THE PROGRAM JUST LOADED.  REGISTER X CONTAINS THE NEXT ADDRESS
. TO BE LOADED.
.
         CLEAR    A        CLEAR REGISTER A TO ZERO
         LDX      #128     INITIALIZE REGISTER X TO HEX 80
LOOP     JSUB     GETC     READ HEX DIGIT FROM PROGRAM BEING LOADED
         RMO      A,S      SAVE IN REGISTER 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 REGISTER X
         TIXR     X,X      ADD 1 TO MEMORY ADDRESS BEING LOADED
         J        LOOP     LOOP UNTIL END OF INPUT IS REACHED
```

Bootstrap loader for SIC/XE (Fig. 3.3)

```
.  SUBROUTINE TO READ ONE CHARACTER FROM INPUT DEVICE AND
.  CONVERT IT FROM ASCII CODE TO HEXADECIMAL DIGIT VALUE. THE
.  CONVERTED DIGIT VALUE IS RETURNED IN REGISTER A. WHEN AN
.  END-OF-FILE IS READ, CONTROL IS TRANSFERRED TO THE STARTING
.  ADDRESS (HEX 80).
.
GETC    TD      INPUT    TEST INPUT DEVICE
        JEQ     GETC     LOOP UNTIL READY
        RD      INPUT    READ CHARACTER
        COMP    #4       IF CHARACTER IS HEX 04 (END OF FILE),
        JEQ     80           JUMP TO START OF PROGRAM JUST LOADED
        COMP    #48      COMPARE TO HEX 30 (CHARACTER '0')
        JLT     GETC     SKIP CHARACTERS LESS THAN '0'
        SUB     #48      SUBTRACT HEX 30 FROM ASCII CODE
        COMP    #10      IF RESULT IS LESS THAN 10, CONVERSION IS
        JLT     RETURN     COMPLETE. OTHERWISE, SUBTRACT 7 MORE
        SUB     #7         (FOR HEX DIGITS 'A' THROUGH 'F')
RETURN  RSUB             RETURN TO CALLER
INPUT   BYTE    X'F1'    CODE FOR INPUT DEVICE
        END     LOOP
```

Figure 3.3 Bootstrap loader for SIC/XE.

Bootstrap loader for SIC/XE (Fig. 3.3)

begin

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

Loop

A←GETC *; read one char. From device F1 and convert it from the*
; ASCII character code to the value of the hex digit

save the value in the high-order 4 bits of S

A←GETC

A← (A+S) *; combine the value to form one byte*

store the value (in A) to the address represented in register X

X←X+1

end

3.2 Machine-Dependent Loader Features

- Drawback of absolute loaders
  - Programmer needs to specify the actual address at which it will be loaded into memory.
  - Difficult to run several programs concurrently, sharing memory between them.
  - Difficult to use subroutine libraries.
- Solution: a more complex loader that provides
  - *Program relocation*
  - *Program linking*

Machine-Dependent Loader Features (Cont.)

- 3.2.1 Relocation
- 3.2.2 Program Linking
- 3.2.3 Algorithm and Data Structures for a Linking Loader

Review

Section 2.2.2

Program Relocation

Program Relocation

- *Relocatable* program

```
COPY   START   0
FIRST  STL     RETADR
            :
            :
```

program loading starting address is determined at load time

  - An object program that contains the information necessary to perform address modification for relocation
  - The **assembler** can identify for the **loader** those parts of object program that need modification.
  - No instruction modification is needed for
    - *immediate addressing* (not a memory address)
    - *PC-relative*, *Base-relative addressing*
  - The only parts of the program that require modification at load time are those that specify ***direct addresses***

Instruction Format vs. Relocatable Loader

- In SIC/XE
  - *Relative* and *immediate* addressing
    - Do not need to modify their object code after relocation
  - *Extended format*
    - Whose values are affected by relocation
    - Need to modify when relocation
- In SIC
  - Format 3 with address field
    - Should be modified
    - SIC does not support PC-relative and base-relative addressing

3.2.1 Relocation

- Loaders that allow for program relocation are called ***relocating loaders*** or ***relative loaders***.
- Two methods for specifying relocation as part of the object program
  - *Modification records*
    - Suitable for a ***small*** number of relocations required when relative or immediate addressing modes are extensively used
  - *Relocation bits*
    - Suitable for a ***large*** number of relocations required when only direct addressing mode can be used in a machine with fixed instruction format (e.g., the standard SIC machine)

Relocation by Modification Record

- A *Modification record* is used to describe each part of the object code that must be changed when the program is relocated.
- Fig 3.4 & 3.5
  - The only portions of the assembled program that contain addresses are the *extended format* instructions on lines 15,35,65
  - The only items whose values are affected by relocation.

Example of a SIC/XE Program (Fig 3.4,2.6)

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	

Only three addresses need to be relocated.

Example of a SIC/XE Program (Fig 3.4,2.6) (Cont.)

```
110                  .
115                  .       SUBROUTINE TO READ RECORD INTO BUFFER
120                  .
125     1036    RDREC        CLEAR    X                    B410
130     1038                 CLEAR    A                    B400
132     103A                 CLEAR    S                    B440
133     103C                +LDT     #4096                 75101000
135     1040    RLOOP        TD       INPUT                E32019
140     1043                 JEQ      RLOOP                332FFA
145     1046                 RD       INPUT                DB2013
150     1049                 COMPR    A,S                  A004
155     104B                 JEQ      EXIT                 332008
160     104E                 STCH     BUFFER,X             57C003
165     1051                 TIXR     T                    B850
170     1053                 JLT      RLOOP                3B2FEA
175     1056    EXIT         STX      LENGTH               134000
180     1059                 RSUB                          4F0000
185     105C    INPUT        BYTE     X'F1'                F1
```

Example of a SIC/XE Program (Fig 3.4,2.6) (Cont.)

195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	OUTPUT	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	OUTPUT	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Figure 2.6 Program from Fig. 2.5 with object code.

Relocatable Program

Pass the address – modification information to the **relocatable loader**

- *Modification record*
  - Col 1 M
  - Col 2-7 Starting location of the address field to be modified, relative to the beginning of the program (hex)
  - Col 8-9 length of the address field to be modified, in half-bytes
  - E.g M$_\wedge$000007$_\wedge$05

Beginning address of the program is to be added to a field that begins at addr ox000007 and is 5 bytes in length.

Object Program with Relocation by Modification Records for Fig 3.5 (Fig 2.8)

Chapter 3 Loaders and Linkers

Add the starting address of the program

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B10103… …290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200… …105D^3E2003^454F46
T^001036^1D^B410^B400^B440^751… …00^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000… …0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05+COPY
M^000014^05+COPY
M^000027^05+COPY
E^000000
```

There is one modification record for each address need to be relocated.

Figure 3.5 Object program with relocation by Modification records.

Relocation by Modification Record (Cont.)

- The Modification record scheme is a convenient means for specifying program relocation.
- However, it is not well suited for use with all machine architectures
  - See Fig. 3.6.
    - Relocatable program for a SIC machine
  - Most instructions use direct addressing
    - ***Too many modification records***

Relocatable program for a standard SIC machine (Fig. 3.6)

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	140033
15	0003	CLOOP	JSUB	RDREC	481039
20	0006		LDA	LENGTH	000036
25	0009		COMP	ZERO	280030
30	000C		JEQ	ENDFIL	300015
35	000F		JSUB	WRREC	481061
40	0012		J	CLOOP	3C0003
45	0015	ENDFIL	LDA	EOF	00002A
50	0018		STA	BUFFER	0C0039
55	001B		LDA	THREE	00002D
60	001E		STA	LENGTH	0C0036
65	0021		JSUB	WRREC	481061
70	0024		LDL	RETADR	080033
75	0027		RSUB		4C0000
80	002A	EOF	BYTE	C'EOF'	454F46
85	002D	THREE	WORD	3	000003
90	0030	ZERO	WORD	0	000000
95	0033	RETADR	RESW	1	
100	0036	LENGTH	RESW	1	
105	0039	BUFFER	RESB	4096	
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		

Relocatable program for a standard SIC machine (Fig. 3.6) (Cont.)

```
             .
1039   RDREC    LDX    ZERO                     040030
103C            LDA    ZERO                     000030
103F   RLOOP    TD     INPUT                    E0105D
1042            JEQ    RLOOP                    30103F
1045            RD     INPUT                    D8105D
1048            COMP   ZERO                     280030
104B            JEQ    EXIT                     301057
104E            STCH   BUFFER,X                 548039
1051            TIX    MAXLEN                   2C105E
1054            JLT    RLOOP                    38103F
1057   EXIT     STX    LENGTH                   100036
105A            RSUB                            4C0000
105D   INPUT    BYTE   X'F1'                    F1
105E   MAXLEN   WORD   4096                     001000
       .
       .        SUBROUTINE TO WRITE RECORD FROM BUFFER
```

Relocatable program for a Standard SIC Machine (fig. 3.6) (Cont.)

205		.			
210	1061	WRREC	LDX	ZERO	040030
215	1064	WLOOP	TD	OUTPUT	E01079
220	1067		JEQ	WLOOP	301064
225	106A		LDCH	BUFFER,X	508039
230	106D		WD	OUTPUT	DC1079
235	1070		TIX	LENGTH	2C0036
240	1073		JLT	LOOP	381064
245	1076		RSUB		4C0000
250	1079	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Figure 3.6 Relocatable program for a standard SIC machine.

This SIC program does not use relative addressing.

The addresses in all the instructions except RSUB must be modified.

This would require 31 Modification records.

Relocation by Modification Bit

- If a machine primarily uses *direct addressing* and has a *fixed instruction format*
  - There are many addresses needed to be modified
  - It is often more efficient to specify relocation using ***relocation bit***
- *Relocation bit* (Fig. 3.6, 3.7)
  - Each instruction is associated with *one relocation bit*
    - Indicate the corresponding word should be modified or not.
  - These relocation bits in a Text record is gathered into ***bit masks***

Relocation by Modification Bit (Fig. 3.7)

- Relocation bit
  - 0: no modification is needed
  - 1: modification is needed

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

```
H^COPY  ^000000^00107A
T^000000^1E^FFC^140033^481039^000036^280030^300015^481061^3C0003^00002A^0C0039^00002D
T^00001E^15^E00^0C0036^481061^080033^4C0000^454F46^000003^000000
T^001039^1E^FFC^040030^000030^E0105D^30103F^D8105D^280030^301057^548039^2C105E^38103F
T^001057^0A^800^100036^4C0000^F1^001000
T^001061^19^FE0^040030^E01079^301064^508039^DC1079^2C0036^381064^4C0000^05
E^000000
```

F1 is one-byte

Figure 3.7 Object program with relocation by bit mask.

Relocation Bits (Cont.)

- Each bit mask consists of 12 relocation bit in each Text record
  - Since each text record contains less than 12 words
  - Unused words are set to 0
    - E.g. FFC=111111111100 for line 10-55
    - However, only 10 words in the first text record

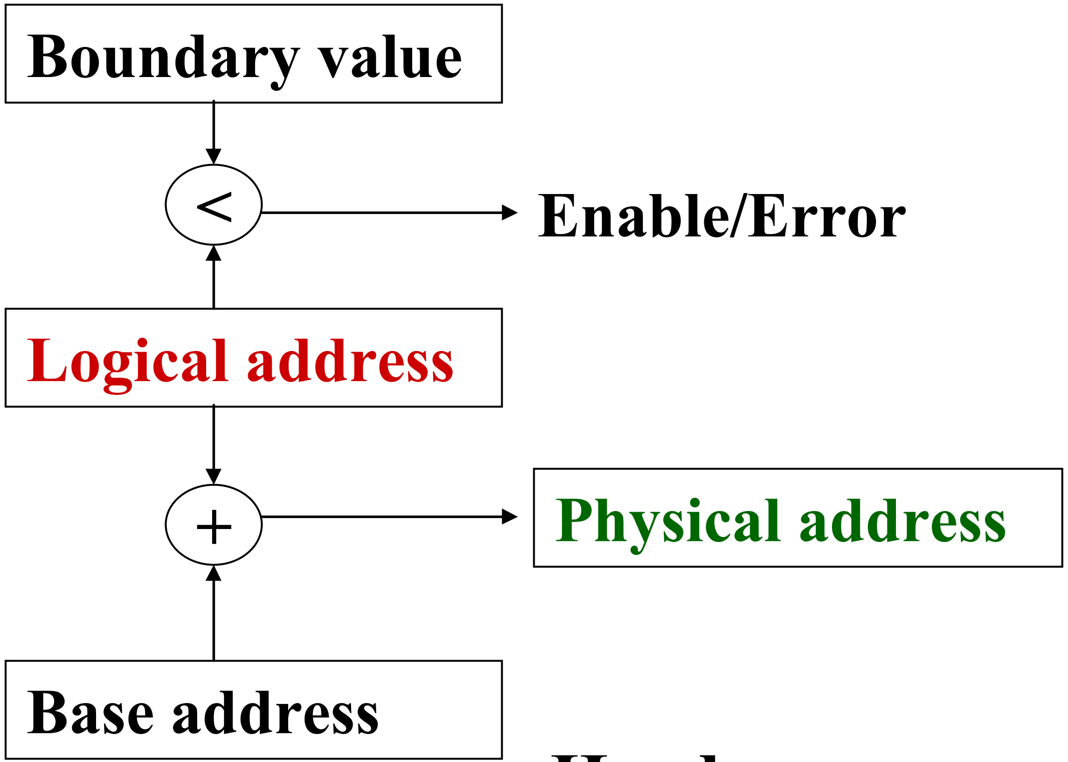
Relocation Bits (Cont.)

- Note that, any value that is to be modified during relocation must coincide with one of these 3-byte segments
  - E.g. Begin a new Text record for line 210
    - Because line 185 has only ***1-byte*** object code (F1)
    - Make the following object code does not align to 3-byte boundary

Relocation (Cont.) (Skip)

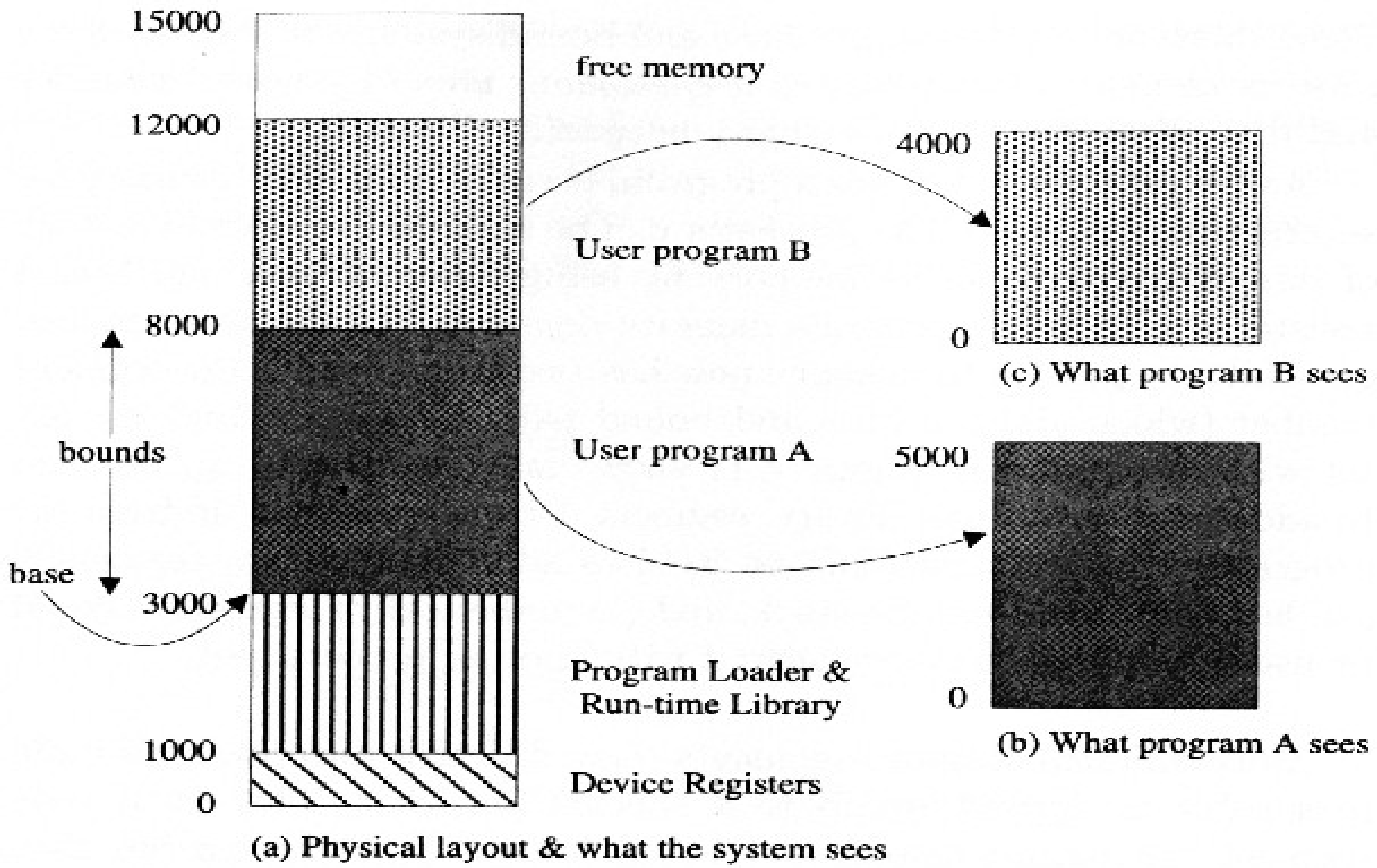
- Some computers provide a hardware relocation capability that eliminates some of the need for the loader to perform program relocation.
  - E.g. some such machine consider all memory references to be relative to the beginning of the user's assigned area of memory.
    - The conversion of these relative addresses to actual addresses is performed as the program is executed.

Mapped Memory (Skip)



Hardware memory mapping

Base & Bound (Skip)



3.2.2 Program Linking

- *Control sections*
  - Refer to segments of codes that are translated into *independent* object program units
  - These control sections could be assembled together or independently of one another
  - It is necessary to provide some means for *linking control sections together*
    - External definitions
    - External references

Illustration of Control Sections and Program Linking (Fig 2.15 & 2.16)

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
6			EXTDEF	BUFFER,BUFEND,LENGTH	
7			EXTREF	RDREC,WRREC	
10	0000	FIRST	STL	RETADR	172027
15	0003	CLOOP	+JSUB	RDREC	4B100000
20	0007		LDA	LENGTH	032023
25	000A		COMP	#0	290000
30	000D		JEQ	ENDFIL	332007
35	0010		+JSUB	WRREC	4B100000
40	0014		J	CLOOP	3F2FEC
45	0017	ENDFIL	LDA	=C'EOF'	032016
50	001A		STA	BUFFER	0F2016
55	001D		LDA	#3	010003
60	0020		STA	LENGTH	0F200A
65	0023		+JSUB	WRREC	4B100000
70	0027		J	@RETADR	3E2000
95	002A	RETADR	RESW	1	
100	002D	LENGTH	RESW	1	
103			LTORG		
	0030	*	=C'EOF'		454F46
105	0033	BUFFER	RESB	4096	
106	1033	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	

Illustration of Control Sections and Program Linking (Fig 2.15 & 2.16) (Cont.)

```
109    0000    RDREC     CSECT
110            .
115            .         SUBROUTINE TO READ RECORD INTO BUFFER
120            .
122                      EXTREF    BUFFER,LENGTH,BUFEND
125    0000              CLEAR     X                      B410
130    0002              CLEAR     A                      B400
132    0004              CLEAR     S                      B440
133    0006              LDT       MAXLEN                 77201F
135    0009    RLOOP     TD        INPUT                  E3201B
140    000C              JEQ       RLOOP                  332FFA
145    000F              RD        INPUT                  DB2015
150    0012              COMPR     A,S                    A004
155    0014              JEQ       EXIT                   332009
160    0017             +STCH      BUFFER,X               57900000
165    001B              TIXR      T                      B850
170    001D              JLT       RLOOP                  3B2FE9
175    0020    EXIT     +STX       LENGTH                 13100000
180    0024              RSUB                             4F0000
185    0027    INPUT     BYTE      X'F1'                  F1
190    0028    MAXLEN    WORD      BUFEND-BUFFER          000000
```

Illustration of Control Sections and Program Linking (Fig 2.15 & 2.16) (Cont.)

```
193    0000    WRREC    CSECT
195            .
200            .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205            .
207                     EXTREF   LENGTH,BUFFER
210    0000             CLEAR    X              B410
212    0002            +LDT      LENGTH         77100000
215    0006    WLOOP    TD       =X'05'         E32012
220    0009             JEQ      WLOOP          332FFA
225    000C            +LDCH     BUFFER,X       53900000
230    0010             WD       =X'05'         DF2008
235    0013             TIXR     T              B850
240    0015             JLT      WLOOP          3B2FEE
245    0018             RSUB                    4F0000
255                     END      FIRST
       001B    *       =X'05'                   05
```

Figure 2.16 Program from Fig. 2.15 with object code.

Control Sections and Program Linking (Cont.)

- **Assembler directive: secname** CSECT
  - Signals the start of a new control section
  - E.g. 109 RDREC CSECT
  - e.g. 193 WRREC CSECT
- *External references*
  - References between control sections
  - The assembler generates information for each external reference that will allows the loader to perform the required linking.

How the Assembler Handles Control Sections?

- **The <u>assembler</u> must include information in the object program that will cause the <u>loader</u> to insert proper values where they are required**
- *Define record*
  - Col. 1 D
  - Col. 2-7 Name of external symbol defined in this control section
  - Col. 8-13 Relative address within this control section (hex)
  - Col.14-73 Repeat information in Col. 2-13 for other external symbols
- *Refer record*
  - Col. 1 R
  - Col. 2-7 Name of external symbol referred to in this control section
  - Col. 8-73 Name of other external reference symbols

How the Assembler Handles Control Sections? (Cont.)

- *Modification record* **(revised)**
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified (hex)
  - Col. 8-9 Length of the field to be modified, in half-bytes (hex)
  - Col. 10 Modification flag (+ or -)
  - Col.11-16 External symbol whose value is to be added to or subtracted from the indicated field.
- Example (Figure 2.17)
  - M000004^05^+RDREC
  - M000011^05^+WRREC
  - M000024^05^+WRREC

  - M000028^06^+BUFEND
  - M000028^06^-BUFFER

Program Linking (Cont.)

- Goal of program linking
  - Resolve the problems with EXTREF and EXTDEF from different control sections

- Example:
  - Fig. 3.8 and Fig. 3.9

Sample Programs Illustrating Linking and Relocation (Fig. 3.8) – PROGA

Loc		Source statement		Object code
0000	PROGA	START	0	
		EXTDEF	LISTA,ENDA	
		EXTREF	LISTB,ENDB,LISTC,ENDC	
		.		
		.		
		.		
0020	REF1	LDA	LISTA	03201D
0023	REF2	+LDT	LISTB+4	77100004
0027	REF3	LDX	#ENDA-LISTA	050014
		.		
		.		
		.		
0040	LISTA	EQU	*	
		.		
		.		
0054	ENDA	EQU	*	
0054	REF4	WORD	ENDA-LISTA+LISTC	000014
0057	REF5	WORD	ENDC-LISTC-10	FFFFF6
005A	REF6	WORD	ENDC-LISTC+LISTA-1	00003F
005D	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000014
0060	REF8	WORD	LISTB-LISTA	FFFFC0
		END	REF1	

Sample Programs Illustrating Linking and Relocation (Fig. 3.8) – PROGB

Loc		Source statement		Object code
0000	PROGB	START	0	
		EXTDEF	LISTB,ENDB	
		EXTREF	LISTA,ENDA,LISTC,ENDC	
		.		
		.		
		.		
0036	REF1	+LDA	LISTA	03100000
003A	REF2	LDT	LISTB+4	772027
003D	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0060	LISTB	EQU	*	
		.		
		.		
0070	ENDB	EQU	*	
0070	REF4	WORD	ENDA-LISTA+LISTC	000000
0073	REF5	WORD	ENDC-LISTC-10	FFFFF6
0076	REF6	WORD	ENDC-LISTC+LISTA-1	FFFFFF
0079	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	FFFFF0
007C	REF8	WORD	LISTB-LISTA	000060
		END		

Figure 3.8 Sample programs illustrating linking and relocation.

Sample Programs Illustrating Linking and Relocation (Fig. 3.8) – PROGC

Loc	Source statement			Object code
0000	PROGC	START	0	
		EXTDEF	LISTC,ENDC	
		EXTREF	LISTA,ENDA,LISTB,ENDB	
		.		
		.		
		.		
0018	REF1	+LDA	LISTA	03100000
001C	REF2	+LDT	LISTB+4	77100004
0020	REF3	+LDX	#ENDA-LISTA	05100000
		.		
		.		
		.		
0030	LISTC	EQU	*	
		.		
		.		
0042	ENDC	EQU	*	
0042	REF4	WORD	ENDA-LISTA+LISTC	000030
0045	REF5	WORD	ENDC-LISTC-10	000008
0048	REF6	WORD	ENDC-LISTC+LISTA-1	000011
004B	REF7	WORD	ENDA-LISTA-(ENDB-LISTB)	000000
004E	REF8	WORD	LISTB-LISTA	000000
		END		

Figure 3.8 *(cont'd)*

Sample Programs Illustrating Linking and Relocation

- Each control section defines a list:
  - Control section A: LISTA --- ENDA
  - Control section B: LISTB --- ENDB
  - Control section C: LISTC --- ENDC
- Each control section contains exactly the same set of references to these lists
  - REF1 through REF3: instruction operands
  - REF4 through REF8: values of data words

Sample Programs Illustrating Linking and Relocation (Fig. 3.9) – PROGA

```
H^PROGA ^000000^000063
D^LISTA ^000040^ENDA  ^000054
R^LISTB ^ENDB  ^LISTC ^ENDC
:
T^000020^0A^03201D^77100004^050014
:
T^000054^0F^000014^FFFFF6^00003F^000014^FFFFC0
M^000024^05^+LISTB
M^000054^06^+LISTC
M^000057^06^+ENDC
M^000057^06^-LISTC
M^00005A^06^+ENDC
M^00005A^06^-LISTC
M^00005A^06^+PROGA
M^00005D^06^-ENDB
M^00005D^06^+LISTB
M^000060^06^+LISTB
M^000060^06^-PROGA
E^000020
```

Figure 3.9 Object programs corresponding to Fig. 3.8.

Sample Programs Illustrating Linking and Relocation (Fig. 3.9) – PROGB

```
H^PROGB ^000000^00007F
D^LISTB ^000060^ENDB  ^000070
R^LISTA ^ENDA  ^LISTC ^ENDC
:
T^000036^0B^03100000^772027^05100000
:
T^000070^0F^000000^FFFFF6^FFFFFF^FFFFF0^000060
M^000037^05^+LISTA
M^00003E^05^+ENDA
M^00003E^05^-LISTA
M^000070^06^+ENDA
M^000070^06^-LISTA
M^000070^06^+LISTC
M^000073^06^+ENDC
M^000073^06^-LISTC
M^000076^06^+ENDC
M^000076^06^-LISTC
M^000076^06^+LISTA
M^000079^06^+ENDA
M^000079^06^-LISTA
M^00007C^06^+PROGB
M^00007C^06^-LISTA
E
```

Sample Programs Illustrating Linking and Relocation (Fig. 3.9) – PROGC

```
H^PROGC ^000000^000051
D^LISTC ^000030^ENDC  ^000042
R^LISTA ^ENDA  ^LISTB ^ENDB
:
T^000018^0C^031000000^77100004^05100000
:
T^000042^0F^000030^000008^000011^000000^000000
M^000019^05^+LISTA
M^00001D^05^+LISTB
M^000021^05^+ENDA
M^000021^05^-LISTA
M^000042^06^+ENDA
M^000042^06^-LISTA
M^000042^06^+PROGC
M^000048^06^+LISTA
M^00004B^06^+ENDA
M^00004B^06^-LISTA
M^00004B^06^-ENDB
M^00004B^06^+LISTB
M^00004E^06^+LISTB
M^00004E^06^-LISTA
E
```

Figure 3.9 *(cont'd)*

REF1 (LISTA)

- Control section A
  - LISTA is defined within the control section.
  - Its address is available using *PC-relative addressing.*
  - No modification for relocation or linking is necessary.
- Control sections B and C
  - LISTA is an *external reference.*
  - Its address is not available
    - An *extended-format instruction* with *address field set to 00000* is used.
  - A modification record is inserted into the object code
    - Instruct the loader to *add the value of LISTA to this address field.*

REF2 (LISTB+4)

- Control sections A and C
  - REF2 is an *external reference* (LISTB) plus a constant (4).
  - The address of LISTB is not available
    - An ***extended-format instruction*** with ***address field set to 00004*** is used.
  - A modification record is inserted into the object code
    - Instruct the loader to add the value of LISTB to this address field.
- Control section B
  - LISTB is defined within the control section.
  - Its address is available using *PC-relative addressing.*
  - No modification for relocation or linking is necessary.

REF3 (#ENDA-LISTA)

- Control section A
  - ENDA and LISTA are defined within the control section.
  - The difference between ENDA and LISTA is immediately available.
  - No modification for relocation or linking is necessary.
- Control sections B and C
  - ENDA and LISTA are *external references*.
  - The difference between them is not available
    - An *extended-format instruction* with address field set to 00000 is used.
  - ***Two*** modification records are inserted into the object code
    - +ENDA
    - -LISTA

REF4 (ENDA-LISTA+LISTC)

- Control section A
  - The values of ENDA and LISTA are internal. Only the value of LISTC is unknown.
  - The address field is initialized as 000014 (ENDA-LISTA).
  - **One** Modification record is needed for LISTC:
    - +LISTC
- Control section B
  - ENDA, LISTA, and LISTC are all unknown.
  - The address field is initialized as 000000.
  - **Three** Modification records are needed:
    - +ENDA
    - -LISTA
    - +LISTC
- Control section C
  - LISTC is defined in this control section but ENDA and LISTA are unknown.
  - The address field is initialized as the ***relative address of LISTC (000030***)
  - **Three** Modification records are needed:
    - +ENDA
    - -LISTA
    - **+PROGC (***for relocation***) // Thus, relocation also use modification record**

Program Linking Example

			Program A	Program B	Program C
Label	Expression	Type	LISTA, ENDA	LISTB, ENDB	LISTC, ENDC
REF1	LISTA	R	local, R		
REF2	LISTB+4	R		local, R	
REF3	ENDA-LISTA	A	local, A		
REF4	ENDA-LISTA+LISTC	R	local, A		local, R
REF5	ENDC-LISTC-10	A			local, A
REF6	ENDC-LISTC+LISTA-1	R	local, R		local, A
REF7	ENDA-LISTA-(ENDB-LISTB)	A	local, A	local, A	
REF8	LISTB-LISTA	R	local, R	local, R	

Program Linking Example (Cont.)

- Suppose the loader sequentially allocate the address for object programs
  - See Fig. 3.10
  - Load address for control sections

PROGA	004000	63
PROGB	004063	7F
PROGC	0040E2	51

- Fig. 3.10
  - Actual address of LISTC: 0030+PROGC=4112

Programs From Fig 3.8 After Linking and Loading (Fig. 3.10a)

Values of REF4, REF5, …, REF8 in three places are all the same.

Memory address	Contents				
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	
3FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
4000	....	....	....	....	
4010	....	....	....	....	
4020	03201D77	1040C705	0014....	....	← PROGA
4030	....	....	....	....	
4040	....	....	....	....	
4050	....	00412600	00080040	51000004	
4060	000083..	....	....	....	
4070	....	....	....	....	
4080	....	....	....	....	
4090	....	....	..031040	40772027	← PROGB
40A0	05100014	....	....	....	
40B0	....	....	....	....	
40C0	....	....	....	....	
40D0	00	41260000	08004051	00000400	
40E0	0083....	....	....	....	
40F0	....	....	0310	40407710	
4100	40C70510	0014....	....	....	← PROGC
4110	....	....	....	....	
4120	....	00412600	00080040	51000004	
4130	000083xx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
4140	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	

Figure 3.10(a) Programs from Fig. 3.8 after linking and loading.

Relocation and Linking Operations Performed on REF4 from PROGA (Fig. 3.10b)

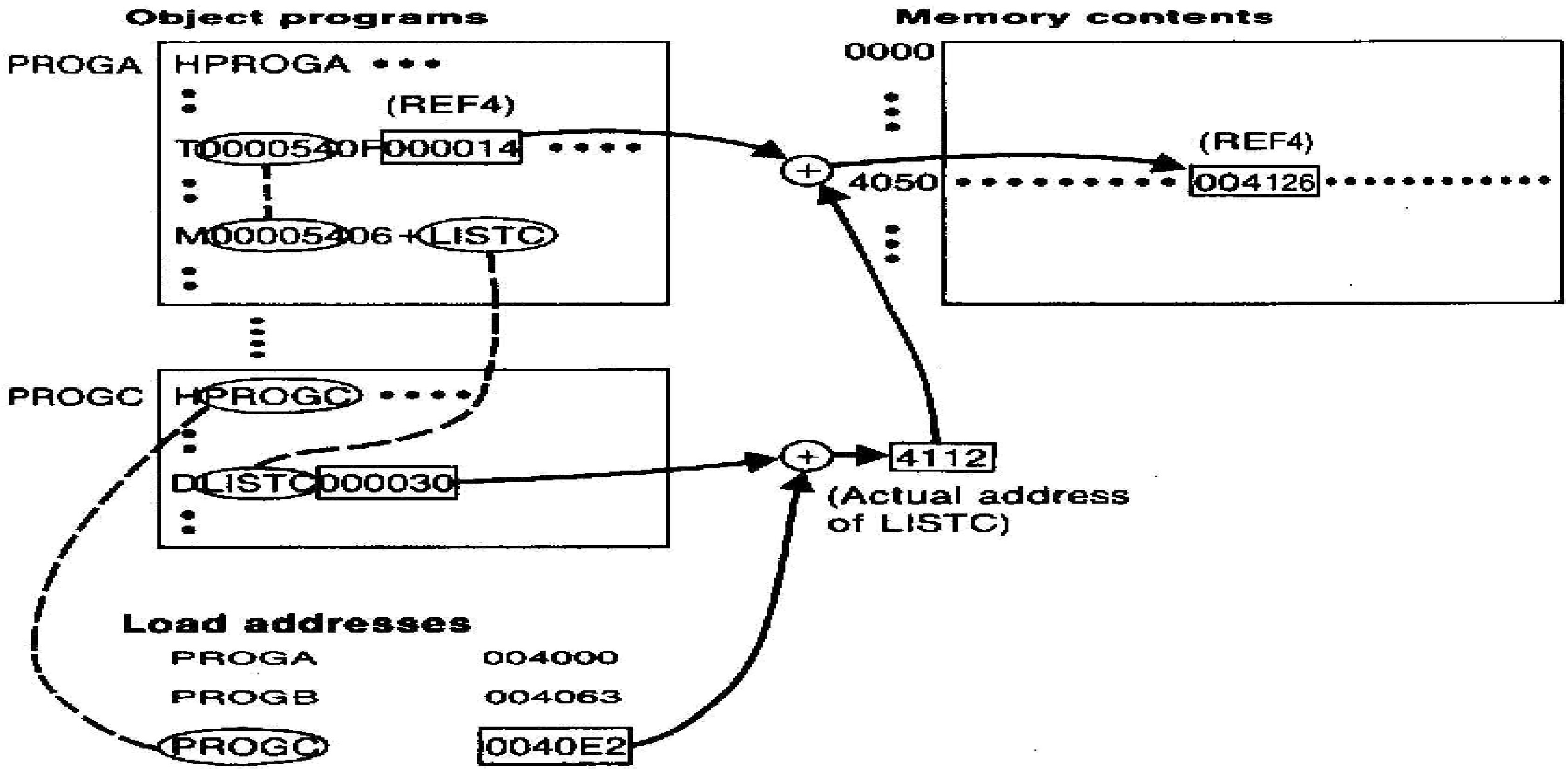


Figure 3.10(b) Relocation and linking operations performed on REF4 from PROGA.

3.2.3 Algorithm and Data Structures for a Linking Loader

Calculation of REF4 (ENDA-LISTA+LISTC)

- Control section A
  - The address of REF4 is 4054 (4000 + 54)
  - The address of LISTC is:

    0040E2 (starting address of PROGC) + 000030 (relative address of LISTC in PROGC) = 004112

  - The value of REF4 is:

    000014 (initial value) + 004112 (address of LISTC) = 004126

- Control section B
  - The address of REF4 is 40D3 (4063 + 70)
  - The value of REF4 is:

    000000 (initial value) + 004054 (address of ENDA) - 004040 (address of LISTA) + 004112 (address of LISTC) = 004126

Target Address are the same

Sample Program for Linking and Relocation

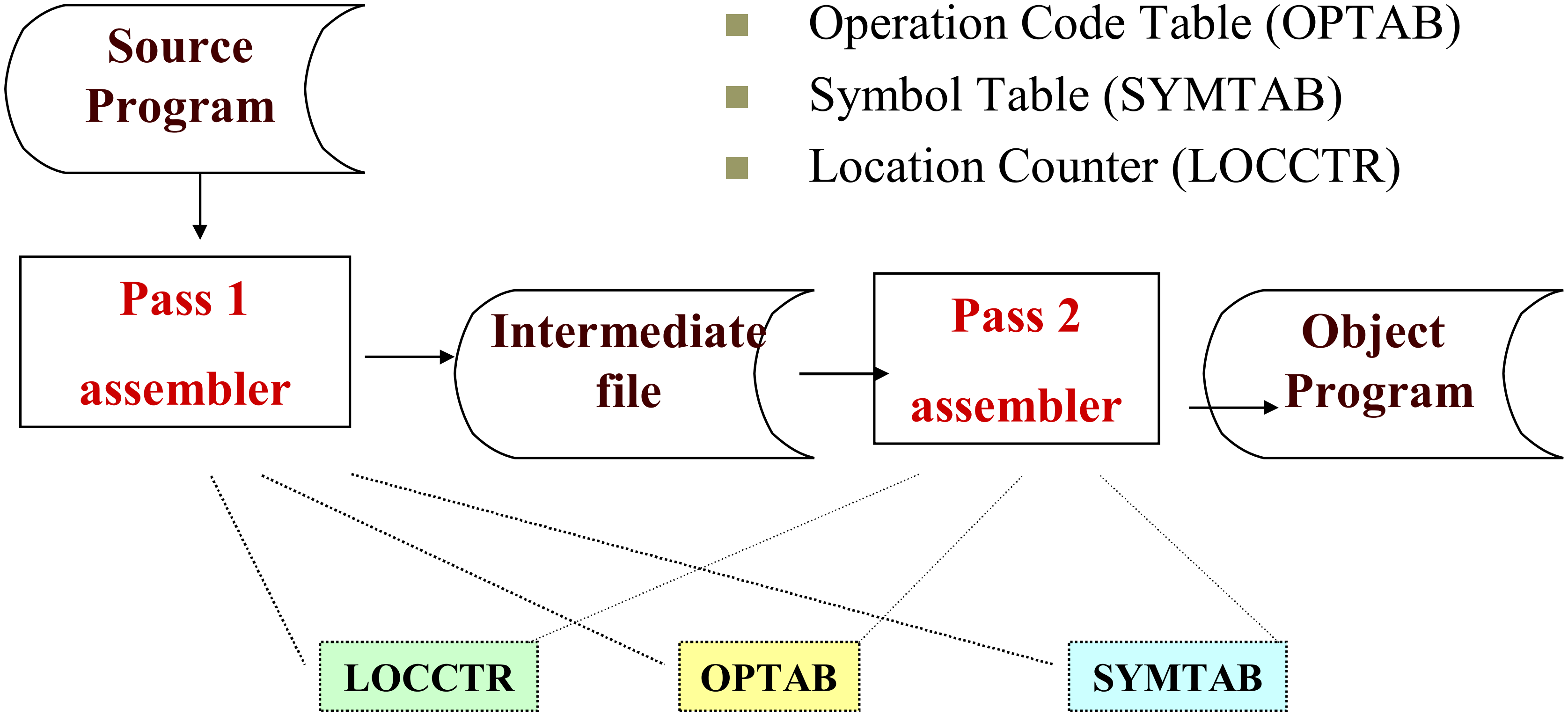
- After these control sections are linked, relocated, and loaded
  - Each of REF4 through REF8 should have the same value in each of the three control sections.
    - They are data labels and have the same expressions
  - But not for REF1 through REF3 (instruction operation)
    - Depends on PC-relative, Base-relative, or direct addressing used in each control section
      - In PROGA, REF1 is a PC-relative
      - In PROGB, REF1 is a direct (actual) address
    - However, the ***target address*** of REF1~REF3 in each control section are the same
      - Target address of REF1 in PROGA, PROGB, PROGC are all 4040

3.2.3 Algorithm and Data Structure for a Linking Loader

- Algorithm for a *linking (and relocating) loader*
  - ***Modification records*** are used for relocation
    - Not use the *modification bits*
    - So that *linking* and *relocation* functions are performed using the same mechanism.
- This type of loader is often found on machines (e.g. SIC/XE)
  - Whose *relative addressing* makes relocation unnecessary for most instructions.

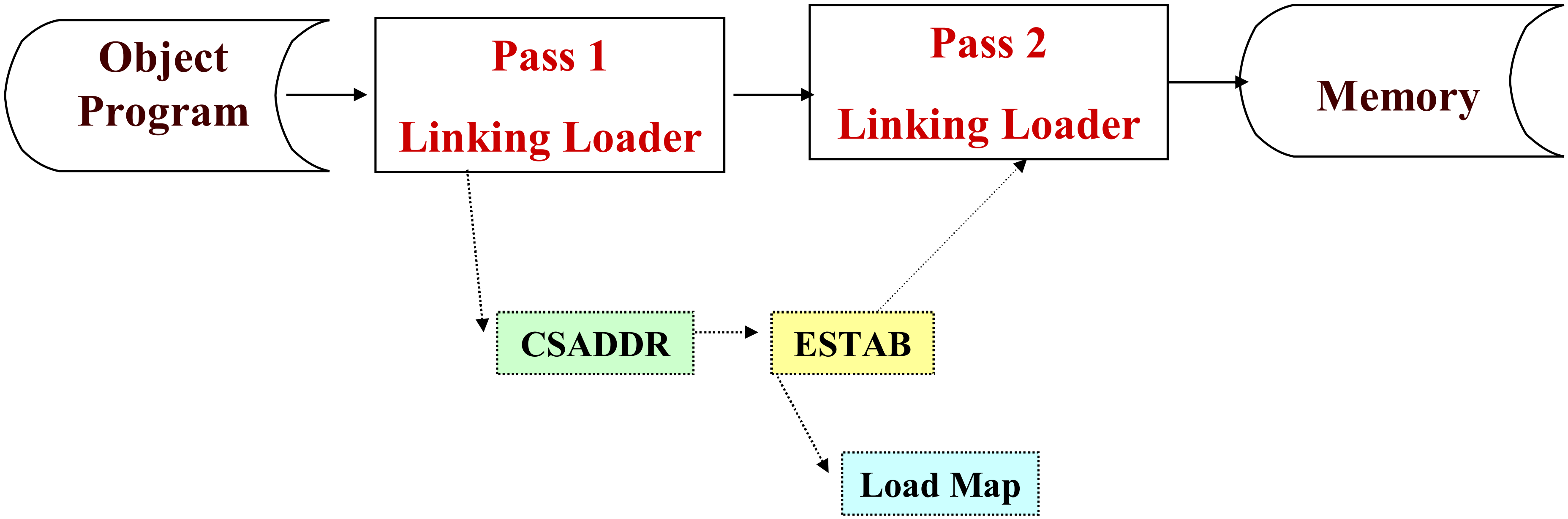
Implementation of An Assembler

- Data Structure
  - Operation Code Table (OPTAB)
  - Symbol Table (SYMTAB)
  - Location Counter (LOCCTR)



Implementation of a Linking Loader

- Two-pass process (similar to the Assembler):
  - Pass 1: assigns addresses to all external symbols
  - Pass 2: performs the actual loading, relocation, and linking



Algorithm for a Linking Loader

- Input is a set of object programs, i.e., control sections
  - Since a control section may make an ***external reference*** to a symbol whose definition will appear later in the input stream
- Thus, a linking loader usually makes two passes over its input, just as an assembler does
  - Pass 1: assign addresses to all *external symbols*
  - Pass 2: perform the actual loading, relocation, and linking

Data Structures

- External Symbol Table (ESTAB)
  - For each external symbol, ESTAB stores
    - its name
    - its address
    - in which control section the symbol is defined
  - *Hashed organization*
- Program Load Address (PROGADDR)
  - PROGADDR is the beginning address in memory where the linked program is to be loaded (supplied by OS).
- Control Section Address (CSADDR)
  - CSADDR is the starting address assigned to the control section currently being scanned by the loader.
- Control section length (CSLTH)

Pass 1 Program Logic (Fig. 3.11a)

- Assign addresses to all external symbols
- The loader is concerned only with Header and Define record types in the control sections
- To build up ESTAB
  - Add *control section name* into ESTAB
  - Add *all external symbols* in the Define record into ESTAB

Pass 1 Program Logic (Fig. 3.11a)

Pass 1: (only ***Header*** and ***Define*** records are concerned)

```
begin
get PROGADDR from operating system
set CSADDR to PROGADDR {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 record 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 {Pass 1}
```

Figure 3.11(a) Algorithm for Pass 1 of a linking loader.

Load Map

- ESTAB (External Symbol Table) may also look like Load MAP

Control section	Symbol name	Address	Length
PROGA			
	LISTA	4040	
	ENDA	4054	
PROGB			
	LISTB	40C3	
	ENDB	40D3	
PROGC			0051
	LISTC	4112	
	ENDC	4124	

Pass 2 Program Logic (Fig. 3.11b)

- Perform the actual loading, relocation, and linking
- When *Text record* is encountered
  - Read into the specified address (+CSADDR)
- When *Modification record* is encountered
  - Lookup the symbol in ESTAB
  - This value is then added to or subtracted from the indicated location in memory
- When the *End record* is encountered
  - Transfer control to the loaded program to begin execution
- **Fig. 3.11(b)**

Pass 2 Program Logic (Fig. 3.11b)

Pass 2:

```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
    begin
        read next input record {Header record}
        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 End record} then
            set EXECADDR to (CSADDR + specified address)
        add CSLTH to CSADDR // the next control section
    end {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

Figure 3.11(b) Algorithm for Pass 2 of a linking loader.

Improve Efficiency

- Use *local searching* instead of multiple searches of ESTAB for the same symbol
  - Assign a ***reference number*** to each external symbol referred to in a control section
  - The reference number (instead of symbol name) is used in Modification records
- Avoiding *multiple searches* of ESTAB for the same symbol during the loading of a control section.
  - Search of ESTAB for each external symbol can be performed **once** and the result is **stored in a new table** indexed by the *reference number*.
  - The values for code modification can then be obtained by simply indexing into the table.

Improve Efficiency (Cont.)

- Implementation
  - 01: control section name
  - other: external reference symbols
- Example
  - Fig. 3.12

Object Programs Corresponding to Fig. 3.8 Using Reference Numbers for Code Modification (Fig. 3.12)

```
H^PROGA ^000000^000063
D^LISTA ^000040^ENDA ^000054
R^02LISTB ^03ENDB ^04LISTC ^05ENDC
.
.
T^000020^0A^03201D^77100004^050014
.
.
T^000054^0F^000014^FFFFF6^00003F^000014^FFFFC0
M^000024^05^+02
M^000054^06^+04
M^000057^06^+05
M^000057^06^-04
M^00005A^06^+05
M^00005A^06^-04
M^00005A^06^+01
M^00005D^06^-03
M^00005D^06^+02
M^000060^06^+02
M^000060^06^-01
E^000020
```

Figure 3.12 Object programs corresponding to Fig. 3.8 using reference numbers for code modification. (Reference numbers are underlined for easier reading.)

Object Programs Corresponding to Fig. 3.8 Using Reference Numbers for Code Modification (Fig. 3.12) (Cont.)

```
HPROGB 000000000007F
DLISTB 000060ENDB  000070
R02LISTA 03ENDA  04LISTC 05ENDC
.
.
T0000360B0310000077202705100000
.
.
T0000700F000000FFFFF6FFFFFFFFFFF0000060
M00003705+02
M00003E05+03
M00003E05-02
M00007006+03
M00007006-02
M00007006+04
M00007306+05
M00007306-04
M00007606+05
M00007606-04
M00007606+02
M00007906+03
M00007906-02
M00007C06+01
M00007C06-02
E
```

Object Programs Corresponding to Fig. 3.8 Using Reference Numbers for Code Modification (Fig. 3.12) (Cont.)

```
HPROGC 000000000051
DLISTC 000030ENDC   000042
R02LISTA 03ENDA   04LISTB 05ENDB
:
T0000180C03100000771000040510000
:
T0000420F000030000008000011000000000000
M00001905+02
M00001D05+04
M00002105+03
M00002105-02
M00004206+03
M00004206-02
M00004206+01
M00004806+02
M00004B06+03
M00004B06-02
M00004B06-05
M00004B06+04
M00004E06+04
M00004E06-02
E
```

Figure 3.12 (*cont'd*)

New Table for Figure 3.12

PROGA

Ref No.	Symbol	Address
1	PROGA	4000
2	LISTB	40C3
3	ENDB	40D3
4	LISTC	4112
5	ENDC	4124

Ref No.	Symbol	Address
1	PROGB	4063
2	LISTA	4040
3	ENDA	4054
4	LISTC	4112
5	ENDC	4124

PROGB

Ref No.	Symbol	Address
1	PROGC	4063
2	LISTA	4040
3	ENDA	4054
4	LISTB	40C3
5	ENDB	40D3

PROGC