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
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

(a) Object program

```
HCOPY 00100000107A
T0010001E1410334820390010362810303010154820613C100300102A0C103900102D
T00101E150C10364820610810334C000454F460000003000000
T0020391E041030001030E0205D30203FD8205D2810303020575490392C205E38203F
T0020571C1010364C0000F1001000041030E02079302064509039DC20792C1036
T002073073820644C000005
E001000
```
Loading of an Absolute Program (Fig 3.1 b)

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>xxxxxxxxxxx</td>
</tr>
<tr>
<td>0010</td>
<td>xxxxxxxxxxx</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>OFF0</td>
<td>xxxxxxxxxxx</td>
</tr>
<tr>
<td>1000</td>
<td>14103348</td>
</tr>
<tr>
<td>1010</td>
<td>20613610</td>
</tr>
<tr>
<td>1020</td>
<td>36482061</td>
</tr>
<tr>
<td>1030</td>
<td>00000000xx</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2030</td>
<td>xxxxxxxxxxx</td>
</tr>
<tr>
<td>2040</td>
<td>205D3020</td>
</tr>
<tr>
<td>2050</td>
<td>392C205E</td>
</tr>
<tr>
<td>2060</td>
<td>00041030</td>
</tr>
<tr>
<td>2070</td>
<td>2C103638</td>
</tr>
<tr>
<td>2080</td>
<td>xxxxxxxxxxx</td>
</tr>
</tbody>
</table>

(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

Figure 3.2 Algorithm for an absolute loader.

E.g., convert the pair of characters “14” (two bytes) in the object program to a single byte with hexadecimal value 14
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
    - 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
    - Begin 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
Bootstrap loader for SIC/XE (Fig. 3.3)

**begin**

\[ X = 0x80 \quad ; \text{the address of the next memory location to be loaded} \]

**Loop**

\[ A \leftarrow \text{GETC} \quad ; \text{read one char. From device F1 and convert it from the} \]
\[ \quad ; \text{ASCII character code to the value of the hex digit} \]
\[ \text{save the value in the high-order 4 bits of S} \]

\[ A \leftarrow \text{GETC} \]

\[ A \leftarrow (A+S) \quad ; \text{combine the value to form one byte} \]

\[ A \leftarrow (A+S) \]

**end**

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

\[ X \leftarrow X+1 \]
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**: 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)

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

Only three addresses need to be relocated.
Example of a SIC/XE Program (Fig 3.4,2.6) (Cont.)

110
115
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    BYTE    X’F1’    4F0000
185  105C   INPUT    BYTE    X’F1’    F1
Example of a SIC/XE Program (Fig 3.4,2.6) (Cont.)

```
195  .
200  .
205  .
210  105D  WRREC  CLEAR  X
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,000007,05

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)

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)

<table>
<thead>
<tr>
<th>Line</th>
<th>Loc</th>
<th>Source statement</th>
<th>Object code</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0000</td>
<td>COPY START 0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0000</td>
<td>FIRST STL RETADR</td>
<td>140033</td>
</tr>
<tr>
<td>15</td>
<td>0003</td>
<td>CLOOP JSUB RDREC</td>
<td>481039</td>
</tr>
<tr>
<td>20</td>
<td>0006</td>
<td>LDA LENGTH</td>
<td>000036</td>
</tr>
<tr>
<td>25</td>
<td>0009</td>
<td>COMP ZERO</td>
<td>280030</td>
</tr>
<tr>
<td>30</td>
<td>000C</td>
<td>JEQ ENDFIL</td>
<td>300015</td>
</tr>
<tr>
<td>35</td>
<td>000F</td>
<td>JSUB WRREC</td>
<td>481061</td>
</tr>
<tr>
<td>40</td>
<td>0012</td>
<td>J CLOOP</td>
<td>3C0003</td>
</tr>
<tr>
<td>45</td>
<td>0015</td>
<td>ENDFIL LDA EOF</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0018</td>
<td>STA BUFFER</td>
<td>0C0039</td>
</tr>
<tr>
<td>55</td>
<td>001B</td>
<td>LDA THREE</td>
<td>00002D</td>
</tr>
<tr>
<td>60</td>
<td>001E</td>
<td>STA LENGTH</td>
<td>0C0036</td>
</tr>
<tr>
<td>65</td>
<td>0021</td>
<td>JSUB WRREC</td>
<td>481061</td>
</tr>
<tr>
<td>70</td>
<td>0024</td>
<td>LDL RETADR</td>
<td>080033</td>
</tr>
<tr>
<td>75</td>
<td>0027</td>
<td>RSUB</td>
<td>4C0000</td>
</tr>
<tr>
<td>80</td>
<td>002A</td>
<td>EOF BYTE C’EOF’</td>
<td>454F46</td>
</tr>
<tr>
<td>85</td>
<td>002D</td>
<td>THREE WORD 3</td>
<td>000003</td>
</tr>
<tr>
<td>90</td>
<td>0030</td>
<td>ZERO WORD 0</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>0033</td>
<td>RETADR RESW 1</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>0036</td>
<td>LENGTH RESW 1</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>0039</td>
<td>BUFFER RESB 4096</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>SUBROUTINE TO READ RECORD INTO BUFFER</td>
<td></td>
</tr>
</tbody>
</table>
A relocatable program for a standard SIC machine (Fig. 3.6) (Cont.)

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Operation</th>
<th>Address</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>1039</td>
<td>RDREC</td>
<td>LDX</td>
<td>ZERO</td>
</tr>
<tr>
<td>130</td>
<td>103C</td>
<td>LDA</td>
<td>ZERO</td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>103F</td>
<td>RLOOP</td>
<td>TD</td>
<td>INPUT</td>
</tr>
<tr>
<td>140</td>
<td>1042</td>
<td>JEQ</td>
<td>RLOOP</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>1045</td>
<td>RD</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1048</td>
<td>COMP</td>
<td>ZERO</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>104B</td>
<td>JEQ</td>
<td>EXIT</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>104E</td>
<td>STCH</td>
<td>BUFFER, X</td>
<td>548039</td>
</tr>
<tr>
<td>165</td>
<td>1051</td>
<td>TIX</td>
<td>MAXLEN</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>1054</td>
<td>JLT</td>
<td>RLOOP</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>1057</td>
<td>EXIT</td>
<td>STX</td>
<td>LENGTH</td>
</tr>
<tr>
<td>180</td>
<td>105A</td>
<td>RSUB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>105D</td>
<td>INPUT</td>
<td>BYTE</td>
<td>X’F1’</td>
</tr>
<tr>
<td>190</td>
<td>105E</td>
<td>MAXLEN</td>
<td>WORD</td>
<td>4096</td>
</tr>
<tr>
<td>195</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SUBROUTINE TO WRITE RECORD FROM BUFFER
Relocatable program for a Standard SIC Machine (fig. 3.6) (Cont.)

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 Relocation 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 Relocation 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

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
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
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)

<table>
<thead>
<tr>
<th>Line</th>
<th>Loc</th>
<th>Source statement</th>
<th>Object code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>COPY</td>
<td>START</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>EXTDEF</td>
<td>BUFFER, BUFEND, LENGTH</td>
<td></td>
</tr>
<tr>
<td>0002</td>
<td>EXTEVR</td>
<td>RDREC, WRREC</td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>FIRST</td>
<td>SL</td>
<td>RETADR</td>
</tr>
<tr>
<td>0004</td>
<td>CLOOP</td>
<td>+JSUB</td>
<td>RDREC</td>
</tr>
<tr>
<td>0007</td>
<td>LDA</td>
<td>LENGTH</td>
<td>032023</td>
</tr>
<tr>
<td>0008</td>
<td>JEQ</td>
<td>ENDFIL</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>+JSUB</td>
<td>WRREC</td>
<td></td>
</tr>
<tr>
<td>0014</td>
<td>CLOOP</td>
<td>3F2FEC</td>
<td></td>
</tr>
<tr>
<td>0017</td>
<td>ENDFIL</td>
<td>LDA =C’EOF’</td>
<td>032016</td>
</tr>
<tr>
<td>001D</td>
<td>STA</td>
<td>BUFFER</td>
<td>0F2016</td>
</tr>
<tr>
<td>001E</td>
<td>LDA</td>
<td>#3</td>
<td>010003</td>
</tr>
<tr>
<td>0020</td>
<td>STA</td>
<td>LENGTH</td>
<td>0F200A</td>
</tr>
<tr>
<td>0023</td>
<td>+JSUB</td>
<td>WRREC</td>
<td></td>
</tr>
<tr>
<td>0027</td>
<td>J</td>
<td>RETADR</td>
<td>3E2000</td>
</tr>
<tr>
<td>002A</td>
<td>RETADR</td>
<td>RESW</td>
<td>1</td>
</tr>
<tr>
<td>002D</td>
<td>LENGTH</td>
<td>RESW</td>
<td>1</td>
</tr>
<tr>
<td>0030</td>
<td>LITORG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0033</td>
<td>BUFFER</td>
<td>RESB</td>
<td>4096</td>
</tr>
<tr>
<td>1033</td>
<td>BUFEED</td>
<td>EQU</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>MAXLEN</td>
<td>EQU</td>
<td>BUFEND–BUFFER</td>
</tr>
</tbody>
</table>
### Illustration of Control Sections and Program Linking (Fig 2.15 & 2.16) (Cont.)

<table>
<thead>
<tr>
<th>Line</th>
<th>Location</th>
<th>Instruction</th>
<th>Parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>109</td>
<td>0000</td>
<td>RDREC</td>
<td>CSECT</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td></td>
<td></td>
<td>SUBROUTINE TO READ RECORD INTO BUFFER</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td></td>
<td>EXTREF</td>
<td>BUFFER, LENGTH, BUFEND</td>
<td></td>
</tr>
<tr>
<td>125</td>
<td>0000</td>
<td>CLEAR</td>
<td>X</td>
<td>B410</td>
</tr>
<tr>
<td>130</td>
<td>0002</td>
<td>CLEAR</td>
<td>A</td>
<td>B400</td>
</tr>
<tr>
<td>132</td>
<td>0004</td>
<td>CLEAR</td>
<td>S</td>
<td>B440</td>
</tr>
<tr>
<td>133</td>
<td>0006</td>
<td>LDT</td>
<td>MAXLEN</td>
<td>77201F</td>
</tr>
<tr>
<td>135</td>
<td>0009</td>
<td>RLOOP</td>
<td>TD</td>
<td>INPUT</td>
</tr>
<tr>
<td>140</td>
<td>000C</td>
<td>JEQ</td>
<td>RLOOP</td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>000F</td>
<td>RD</td>
<td>INPUT</td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>0012</td>
<td>COMPR</td>
<td>A, S</td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>0014</td>
<td>JEQ</td>
<td>EXIT</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>0017</td>
<td>+STCH</td>
<td>BUFFER, X</td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>001B</td>
<td>TIXR</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>001D</td>
<td>JL T</td>
<td>RLOOP</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>0020</td>
<td>EXIT+STX</td>
<td>LENGTH</td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>0024</td>
<td>RSUB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>185</td>
<td>0027</td>
<td>INPUT</td>
<td>BYTE</td>
<td>X’F1’</td>
</tr>
<tr>
<td>190</td>
<td>0028</td>
<td>MAXLEN</td>
<td>WORD</td>
<td>BUFEND–BUFFER</td>
</tr>
</tbody>
</table>
Illustration of Control Sections and Program Linking (Fig 2.15 & 2.16) (Cont.)

<table>
<thead>
<tr>
<th>193</th>
<th>0000</th>
<th>WRREC</th>
<th>CSECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>195</td>
<td></td>
<td>SUBROUTINE TO WRITE RECORD FROM BUFFER</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td></td>
<td>EXTREF</td>
<td>LENGTH, BUFFER</td>
</tr>
<tr>
<td>205</td>
<td></td>
<td>CLEAR</td>
<td>X</td>
</tr>
<tr>
<td>210</td>
<td>0000</td>
<td>+LDT</td>
<td>LENGTH</td>
</tr>
<tr>
<td>212</td>
<td>0002</td>
<td>WLOOP</td>
<td>=X’05’</td>
</tr>
<tr>
<td>215</td>
<td>0006</td>
<td>JEQ</td>
<td>WLOOP</td>
</tr>
<tr>
<td>220</td>
<td>0009</td>
<td>+LDCH</td>
<td>BUFFER, X</td>
</tr>
<tr>
<td>225</td>
<td>000C</td>
<td>WD</td>
<td>=X’05’</td>
</tr>
<tr>
<td>230</td>
<td>0010</td>
<td>TIXR</td>
<td>T</td>
</tr>
<tr>
<td>235</td>
<td>0013</td>
<td>JLT</td>
<td>WLOOP</td>
</tr>
<tr>
<td>240</td>
<td>0015</td>
<td>RSUB</td>
<td></td>
</tr>
<tr>
<td>245</td>
<td>0018</td>
<td>END</td>
<td>FIRST</td>
</tr>
<tr>
<td>255</td>
<td></td>
<td>001B</td>
<td>*</td>
</tr>
</tbody>
</table>

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 assembler must include information in the object program that will cause the loader 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

<table>
<thead>
<tr>
<th>Loc</th>
<th>Source statement</th>
<th>Object code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>PROGA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>START O</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Extdef LISTA, ENDA</td>
<td>03201D</td>
</tr>
<tr>
<td></td>
<td>Extref LISTB, ENDB, LISTC, ENDC</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>REF1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDA LISTA</td>
<td>03201D</td>
</tr>
<tr>
<td>0023</td>
<td>REF2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+LDT LISTB+4</td>
<td>77100004</td>
</tr>
<tr>
<td>0027</td>
<td>REF3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDX #ENDA-LISTA</td>
<td>050014</td>
</tr>
<tr>
<td>0040</td>
<td>LISTA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EQU *</td>
<td></td>
</tr>
<tr>
<td>0054</td>
<td>ENDA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EQU *</td>
<td>000014</td>
</tr>
<tr>
<td>0054</td>
<td>REF4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD ENDA-LISTA+LISTC</td>
<td>000014</td>
</tr>
<tr>
<td>0057</td>
<td>REF5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD ENDC-LISTC-10</td>
<td>FFFFE6</td>
</tr>
<tr>
<td>005A</td>
<td>REF6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD ENDC-LISTC+LISTA-1</td>
<td>00003F</td>
</tr>
<tr>
<td>005D</td>
<td>REF7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD ENDA-LISTA-(ENDB-LISTB)</td>
<td>000014</td>
</tr>
<tr>
<td>0060</td>
<td>REF8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD LISTB-LISTA</td>
<td>FFFFC0</td>
</tr>
<tr>
<td></td>
<td>END REF1</td>
<td></td>
</tr>
</tbody>
</table>
Sample Programs Illustrating Linking and Relocation (Fig. 3.8) – PROGB

<table>
<thead>
<tr>
<th>Loc</th>
<th>Source statement</th>
<th>Object code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>PROGB START .0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTDEF LISTB, ENDB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTREF LISTA, ENDA, LISTC, ENDC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0036</td>
<td>REF1 +LDA LISTA</td>
<td>03100000</td>
</tr>
<tr>
<td>003A</td>
<td>REF2 LDT LISTB+4</td>
<td>772027</td>
</tr>
<tr>
<td>003D</td>
<td>REF3 +LDX #ENDA-LISTA</td>
<td>05100000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0060</td>
<td>LISTB EQU *</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0070</td>
<td>ENDB EQU *</td>
<td></td>
</tr>
<tr>
<td>0070</td>
<td>REF4 WORD ENDA-LISTA+LISTC</td>
<td>000000</td>
</tr>
<tr>
<td>0073</td>
<td>REF5 WORD ENDC-LISTC-10</td>
<td>FFFFF6</td>
</tr>
<tr>
<td>0076</td>
<td>REF6 WORD ENDC-LISTC+LISTA-1</td>
<td>FFFFFF</td>
</tr>
<tr>
<td>0079</td>
<td>REF7 WORD ENDA-LISTA- (ENDB-LISTB)</td>
<td>FFFFFF0</td>
</tr>
<tr>
<td>007C</td>
<td>REF8 WORD LISTB-LISTA</td>
<td>000060</td>
</tr>
</tbody>
</table>

**Figure 3.8** Sample programs illustrating linking and relocation.
Sample Programs Illustrating Linking and Relocation (Fig. 3.8) – PROGC

<table>
<thead>
<tr>
<th>Loc</th>
<th>Source statement</th>
<th>Object code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>PROGC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>START 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTDDEF LISTC, ENDC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTREF LISTA, ENDA, LISTB, ENDB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>REF1 +LDA LISTA</td>
<td>03100000</td>
</tr>
<tr>
<td>001C</td>
<td>REF2 +LDT LISTB+4</td>
<td>77100004</td>
</tr>
<tr>
<td>0020</td>
<td>REF3 +LDX #ENDA-LISTA</td>
<td>05100000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0030</td>
<td>LISTC EQU *</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0042</td>
<td>ENDC EQU *</td>
<td></td>
</tr>
<tr>
<td>0042</td>
<td>REF4 WORD ENDA-LISTA+LISTC</td>
<td>0000030</td>
</tr>
<tr>
<td>0045</td>
<td>REF5 WORD ENDC-LISTC-10</td>
<td>0000008</td>
</tr>
<tr>
<td>0048</td>
<td>REF6 WORD ENDC-LISTC+LISTA-1</td>
<td>000011</td>
</tr>
<tr>
<td>004B</td>
<td>REF7 WORD ENDA-LISTA-(ENDB-LISTB)</td>
<td>0000000</td>
</tr>
<tr>
<td>004E</td>
<td>REF8 WORD LISTB-LISTA</td>
<td>0000000</td>
</tr>
</tbody>
</table>

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

Figure 3.9  Object programs corresponding to Fig. 3.8.
Sample Programs Illustrating Linking and Relocation (Fig. 3.9) – PROGB
Sample Programs Illustrating Linking and Relocation (Fig. 3.9) – PROGC

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.
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 (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.

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.
Calculation of REF4 (ENDA-LISTA+LISTC)

- **Control section A**
  - The address of REF4 is 4054 (4000 + 54)
  - The address of LISTC is:
    
    \[\text{0040E2} + \text{000030} = \text{004112}\]
    
    (starting address of PROGC) (relative address of LISTC in PROGC)
  - The value of REF4 is:
    
    \[\text{000014} + \text{004112} = \text{004126}\]
    
    (initial value) (address of LISTC)

- **Control section B**
  - The address of REF4 is 40D3 (4063 + 70)
  - The value of REF4 is:
    
    \[\text{000000} + \text{004054} - \text{004040} + \text{004112} = \text{004126}\]
    
    (initial value) (address of ENDA) (address of LISTA) (address of LISTC)

Target Address are the same
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)

Source Program

Pass 1 assembler

Intermediate file

Pass 2 assembler

Object Program
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

- 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**
  - Loader is concerned only with **Header** and **Define** records 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: (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

<table>
<thead>
<tr>
<th>Control section</th>
<th>Symbol name</th>
<th>Address</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGA</td>
<td>LISTA</td>
<td>4040</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENDA</td>
<td>4054</td>
<td></td>
</tr>
<tr>
<td>PROGB</td>
<td>LISTB</td>
<td>40C3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENDB</td>
<td>40D3</td>
<td></td>
</tr>
<tr>
<td>PROGC</td>
<td>LISTC</td>
<td>4112</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENDC</td>
<td>4124</td>
<td></td>
</tr>
</tbody>
</table>
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:

\begin{verbatim}
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\}
\end{verbatim}

\textbf{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 also 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)

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

```
PROC 00000000000051
DLISTC 000030ENDC 000042
R02LISTA Q3ENDA Q4LISTB Q5ENDB

T0000180C031000007710000405100000

T0000420F0000300000800001100000000000
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
```

Figure 3.12 (cont’d)
New Table for Figure 3.12

<table>
<thead>
<tr>
<th>Ref No.</th>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGA</td>
<td>4000</td>
</tr>
<tr>
<td>2</td>
<td>LISTB</td>
<td>40C3</td>
</tr>
<tr>
<td>3</td>
<td>ENDB</td>
<td>40D3</td>
</tr>
<tr>
<td>4</td>
<td>LISTC</td>
<td>4112</td>
</tr>
<tr>
<td>5</td>
<td>ENDC</td>
<td>4124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref No.</th>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGB</td>
<td>4063</td>
</tr>
<tr>
<td>2</td>
<td>LISTA</td>
<td>4040</td>
</tr>
<tr>
<td>3</td>
<td>ENDA</td>
<td>4054</td>
</tr>
<tr>
<td>4</td>
<td>LISTC</td>
<td>4112</td>
</tr>
<tr>
<td>5</td>
<td>ENDC</td>
<td>4124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref No.</th>
<th>Symbol</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGC</td>
<td>4063</td>
</tr>
<tr>
<td>2</td>
<td>LISTA</td>
<td>4040</td>
</tr>
<tr>
<td>3</td>
<td>ENDA</td>
<td>4054</td>
</tr>
<tr>
<td>4</td>
<td>LISTB</td>
<td>40C3</td>
</tr>
<tr>
<td>5</td>
<td>ENDB</td>
<td>40D3</td>
</tr>
</tbody>
</table>