SWE4001 – System Programming
Module 4: Loader and Linkers
Lesson 2 of 6: Machine Dependent Features

3.2 Machine-Dependent Loader Features

- Absolute loader has several potential disadvantages.
  - The actual address at which it will be loaded into memory.
  - Cannot run several independent programs together, sharing memory between them.
  - It difficult to use subroutine libraries efficiently.

More complex loader.

- Relocation
- Linking
- Linking loader

Machine Dependent Loader Features

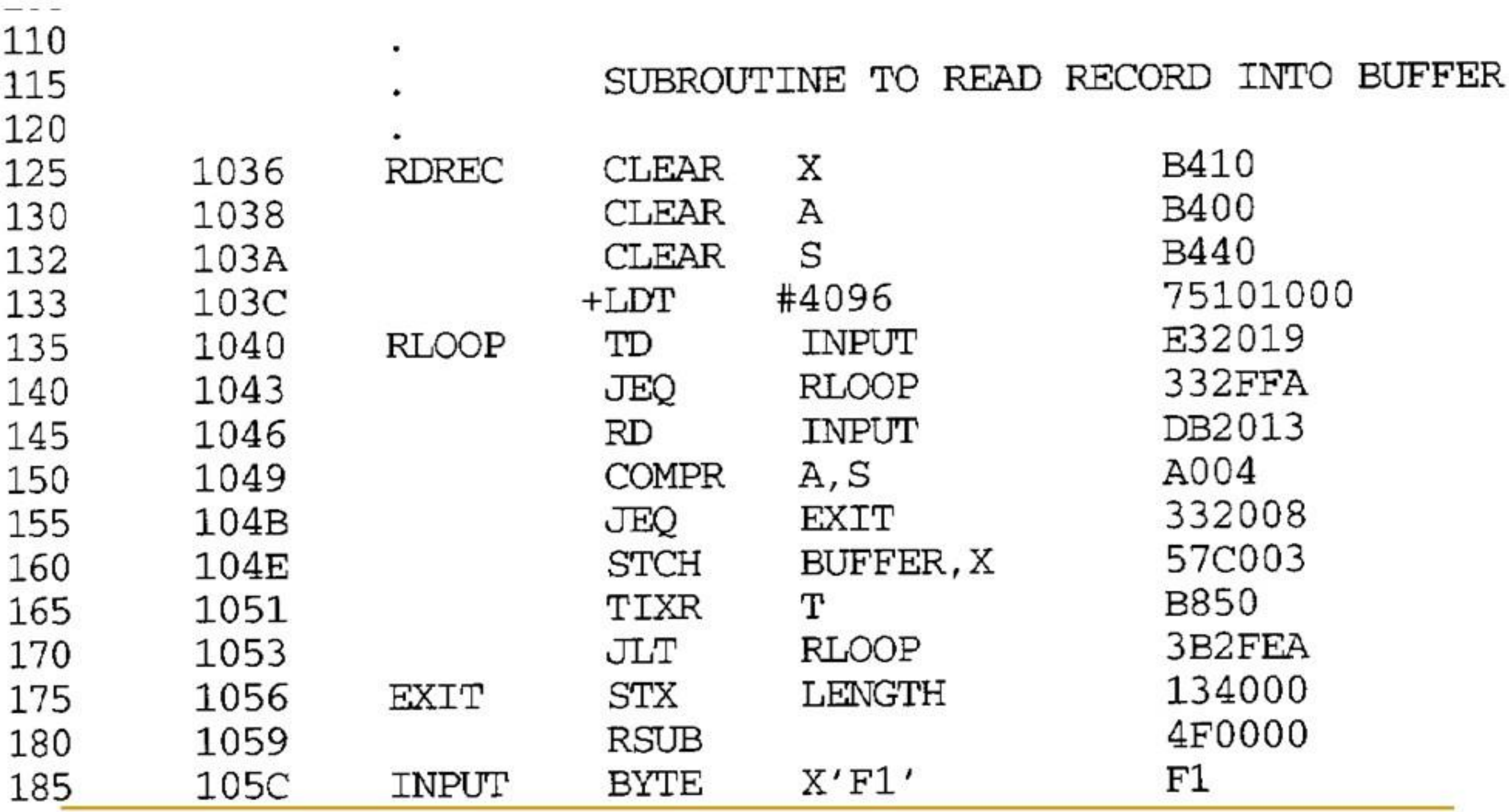
- Program Relocation-Relocatable Loader
  - Modification Record-SIC/XE
  - Bit Mask - SIC
- Program Linking-Linking Loader
  - Pass 1 – ESTAB construction
  - Pass 2 – Linking, Loading, & Relocation

Methods for Relocation

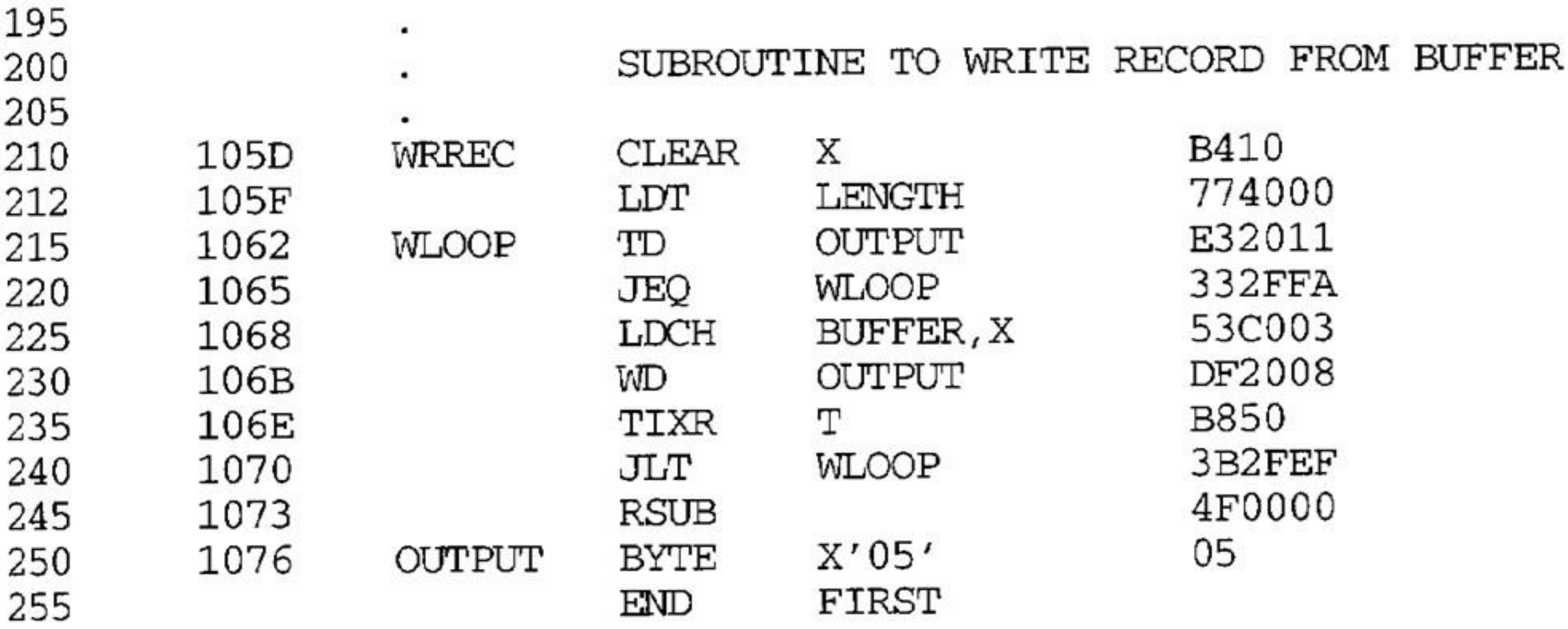
- Two methods for specifying relocation
  - modification record
  - relocation bit
    - each instruction is associated with one relocation bit
    - these relocation bits in a Text record is gathered into bit masks

```
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
```



```
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
```

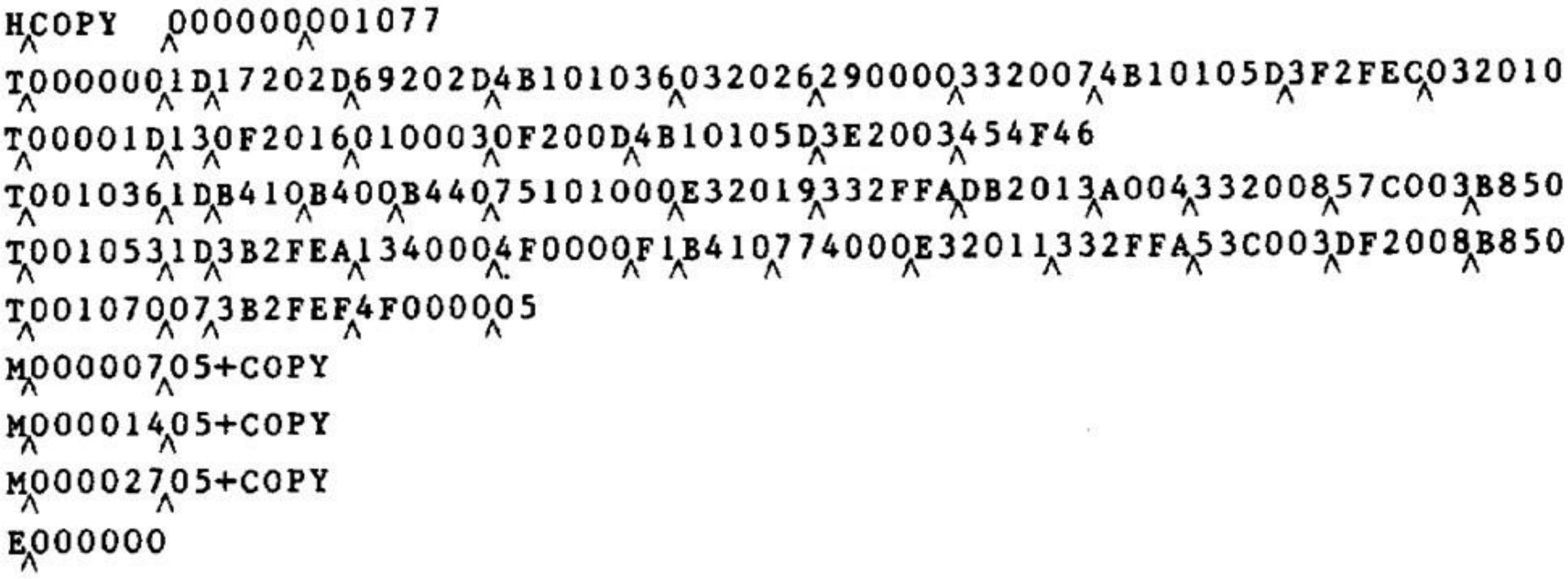


```
195                .
200                .          SUBROUTINE TO WRITE RECORD FROM BUFFER
205                .
210     105D    WRREC      CLEAR    X                      B410
212     105F               LDT      LENGTH                 774000
215     1062    WLOOP      TD       OUTPUT                 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 3.4 Example of a SIC/XE program (from Fig. 2.6).

3.2.1 Relocation

- Modification record, Figure 3.4 and 3.5.
  - To described each part of the object code that must be changed when the program is relocated.
  - The extended format instructions on lines 15, 35, and 65 are affected by relocation. (absolute addressing)
  - In this example, all modifications add the value of the symbol COPY, which represents the starting address.
  - Not well suited for *standard version of SIC*, all the instructions except RSUB must be modified when the program is relocated. (absolute addressing)



```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^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
```

Figure 3.5 Object program with relocation by Modification records.

3.2.1 Relocation

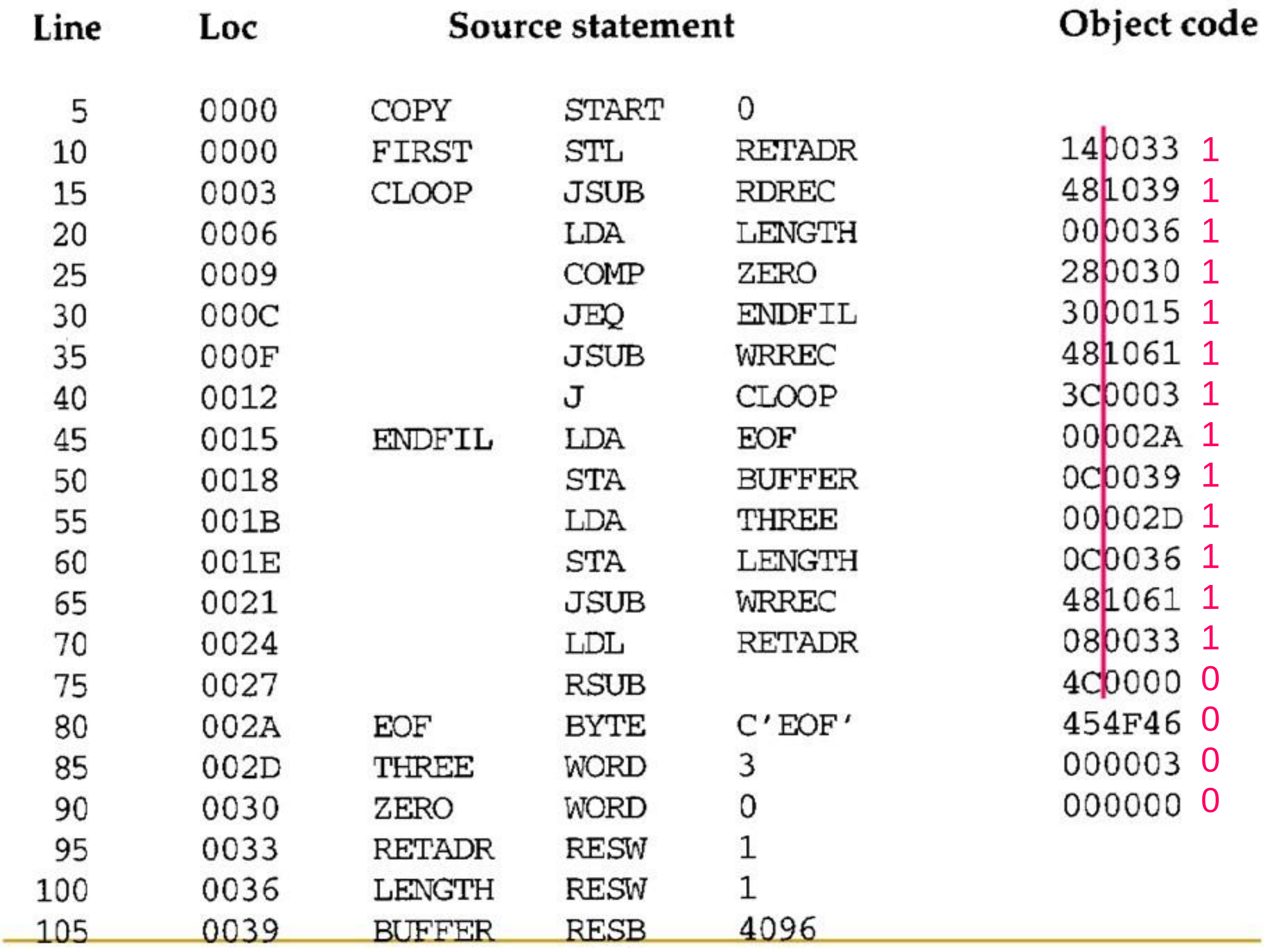
- Figure 3.6 needs 31 Modification records.
- Relocation bit, Figure 3.6 and 3.7.
  - A *relocation bit* associated with each word of object code.
  - The relocation bits are gathered together into a *bit mask* following the length indicator in each Text record.
  - If bit=1, the corresponding word of object code is relocated.

Bit Masking

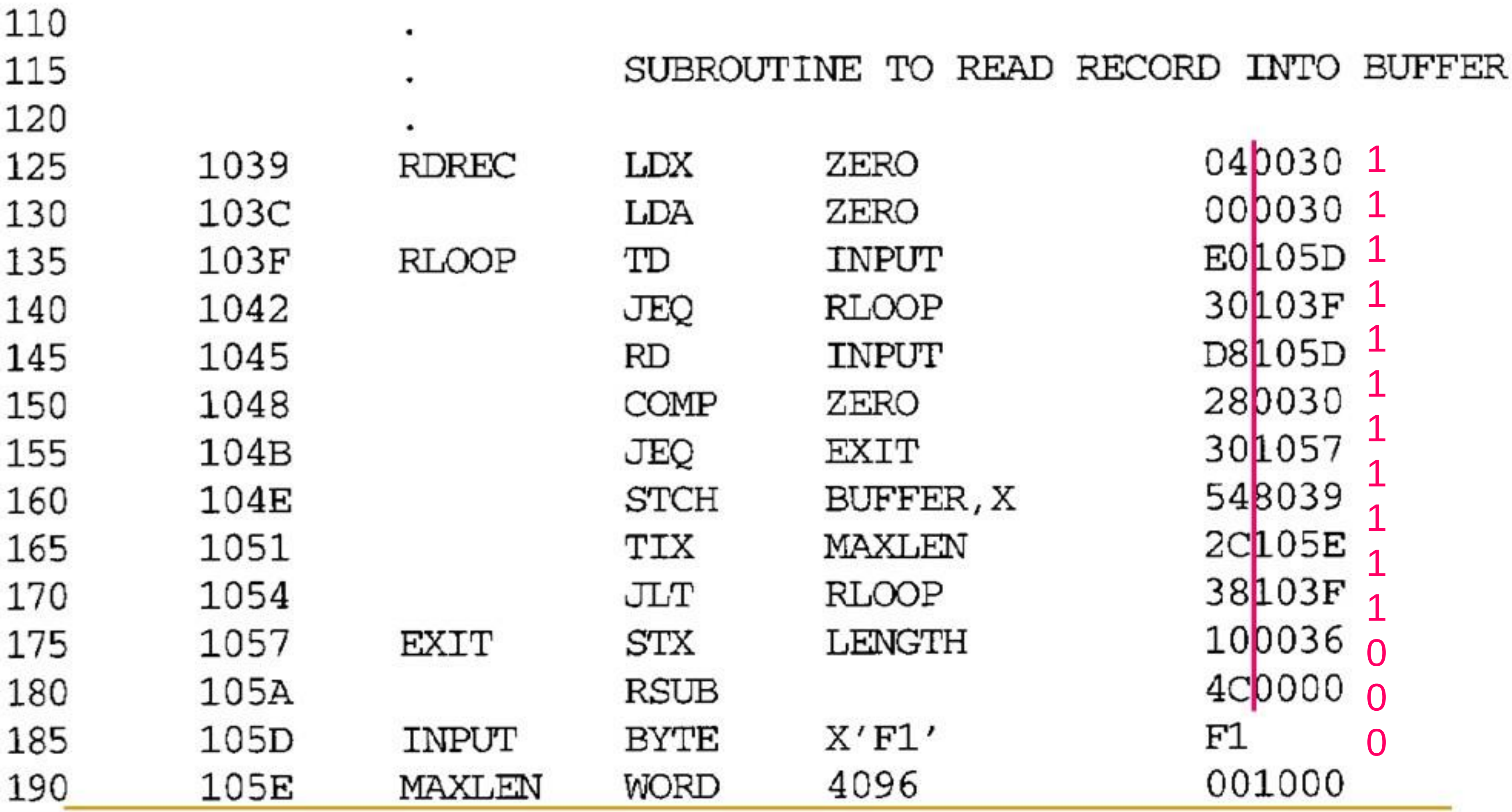
- Twelve-bit mask is used in each Text record
  - since each text record contains less than 12 words
  - unused words are set to 0
  - any value that is to be modified during relocation must coincide with one of these 3-byte segments

Bit Masking

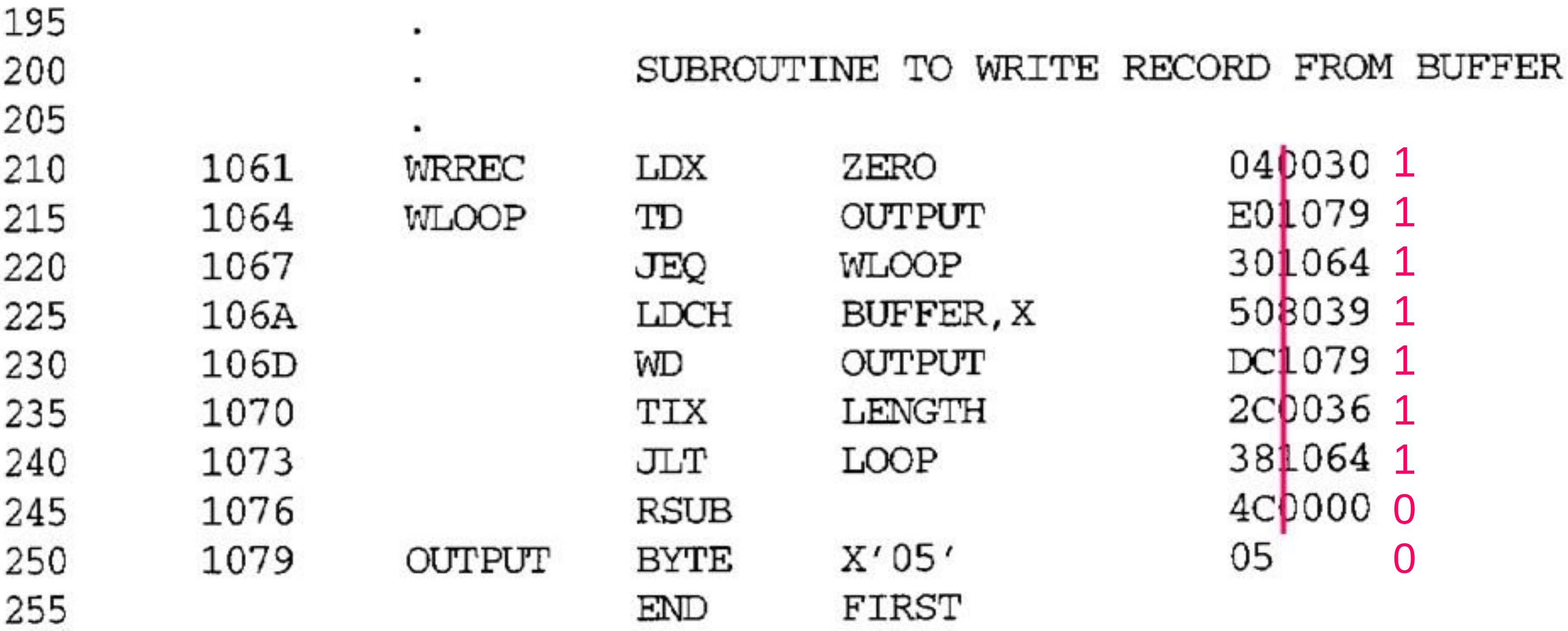
- Text record
  - col 1: T
  - col 2-7: starting address
  - col 8-9: length (byte)
  - col 10-12: relocation bits
  - col 13-72: object code



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



```
110                .
115                .          SUBROUTINE TO READ RECORD INTO BUFFER
120                .
125      1039      RDREC      LDX        ZERO             040030  1
130      103C                 LDA        ZERO             000030  1
135      103F      RLOOP      TD         INPUT            E0105D  1
140      1042                 JEQ        RLOOP            30103F  1
145      1045                 RD         INPUT            D8105D  1
150      1048                 COMP       ZERO             280030  1
155      104B                 JEQ        EXIT             301057  1
160      104E                 STCH       BUFFER,X         548039  1
165      1051                 TIX        MAXLEN           2C105E  1
170      1054                 JLT        RLOOP            38103F  1
175      1057      EXIT       STX        LENGTH           100036  0
180      105A                 RSUB                        4C0000  0
185      105D      INPUT      BYTE       X'F1'            F1      0
190      105E      MAXLEN     WORD       4096             001000
```



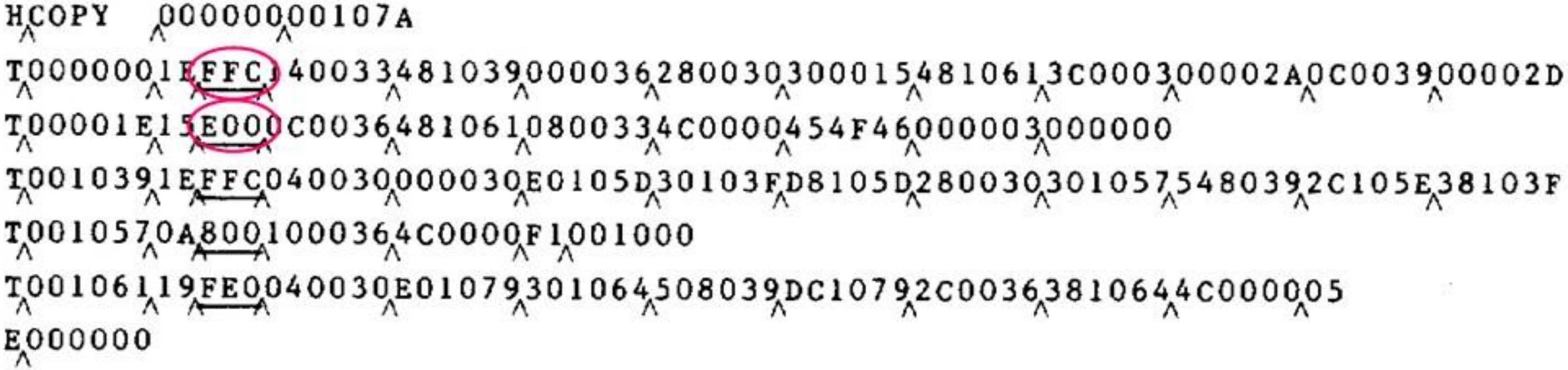
```
195                   .
200                   .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205                   .
210     1061    WRREC      LDX      ZERO                040030 1
215     1064    WLOOP      TD       OUTPUT              E01079 1
220     1067               JEQ      WLOOP               301064 1
225     106A               LDCH     BUFFER,X            508039 1
230     106D               WD       OUTPUT              DC1079 1
235     1070               TIX      LENGTH              2C0036 1
240     1073               JLT      LOOP                381064 1
245     1076               RSUB                         4C0000 0
250     1079    OUTPUT     BYTE     X'05'               05     0
255                        END      FIRST
```

Figure 3.6 Relocatable program for a standard SIC machine.

3.2.1 Relocation

- Relocation bit, Figure 3.6 and 3.7.
  - In Figure 3.7, T000000^1E^FFC^ (111111111100) specifics that all 10 words of object code are to be modified.
  - On line 210 begins a new Text record even though there is room for it in the preceding record.
  - Any value that is to be modified during relocation must coincide with one of these 3-byte segments so that it corresponding to a relocation bit.
  - Because of the 1-byte data value generated form line 185, this instruction must begin a new Text record in object program.

1111 11111100



```
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
```

Figure 3.7 Object program with relocation by bit mask.

1110 0000 0000

3.2.2 Program Linking

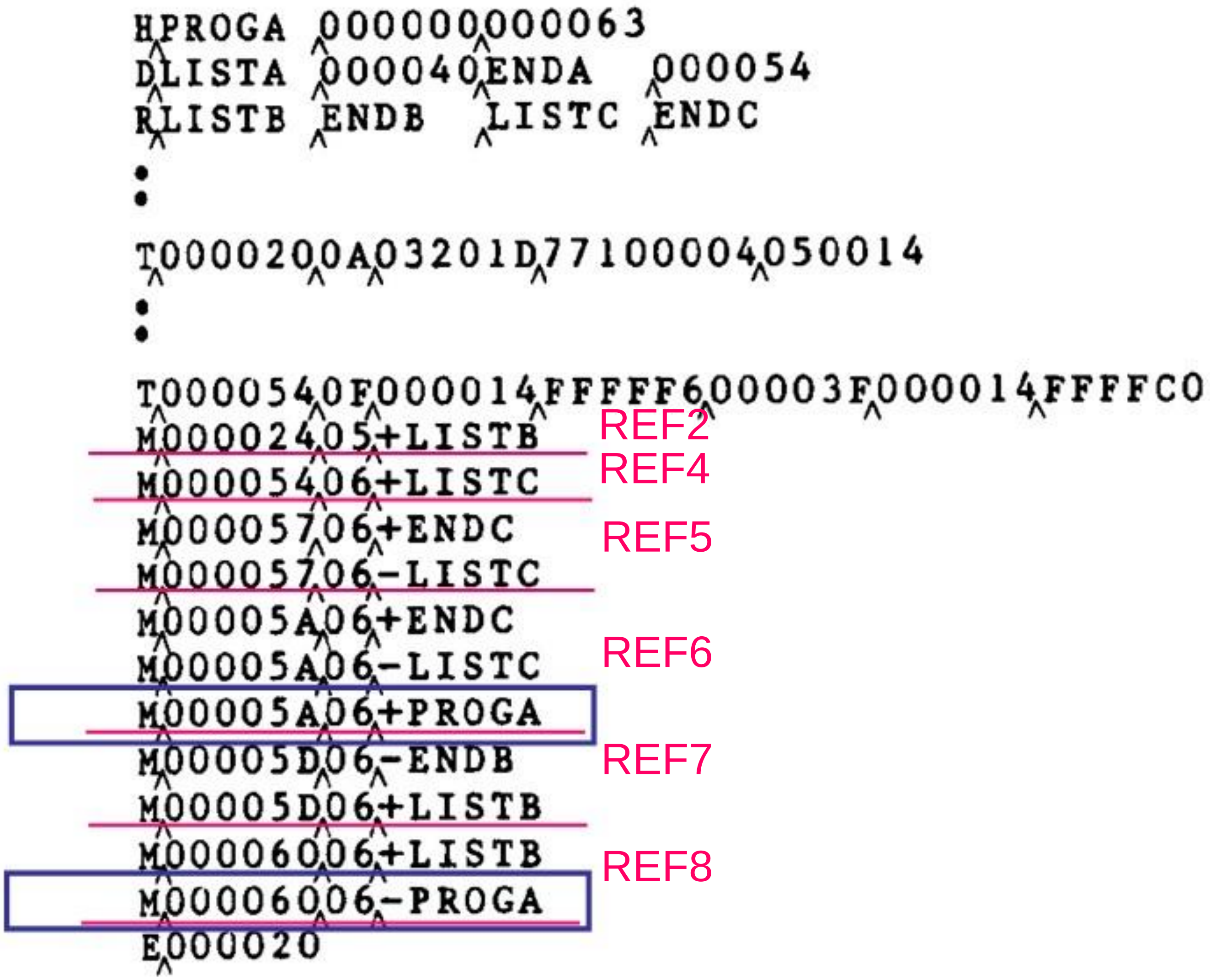
- In Section 2.3.5 showed a program made up of three controls sections.
  - Assembled together or assembled independently.

3.2.2 Program Linking

- Consider the three programs in Fig. 3.8 and 3.9.
  - Each of which consists of a single control section.
  - A list of items, LISTA---ENDA, LISTB---ENDB, LISTC---ENDC.
  - Note that each program contains exactly the same set of references to these external symbols.
  - Instruction operands (REF1, REF2, REF3).
  - The values of data words (REF4 through REF8).
  - Not involved in the relocation and linking are omitted.

```
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     *                       [redacted]
                .
             [redacted]
0054   ENDA     EQU     *
0054   REF4     WORD    ENDA-LISTA+LISTC                      000014
0057   REF5     WORD    ENDC-LISTC-10                         FFFFF6
005A   REF6     WORD    [redacted]+LISTA-1                    00003F
005D   REF7     WORD    ENDA-LISTA-([redacted])               000014
0060   REF8     WORD    [redacted]-LISTA                      FFFFC0
                END     REF1
```



```
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
```

REF2

REF4

REF5

REF6

REF7

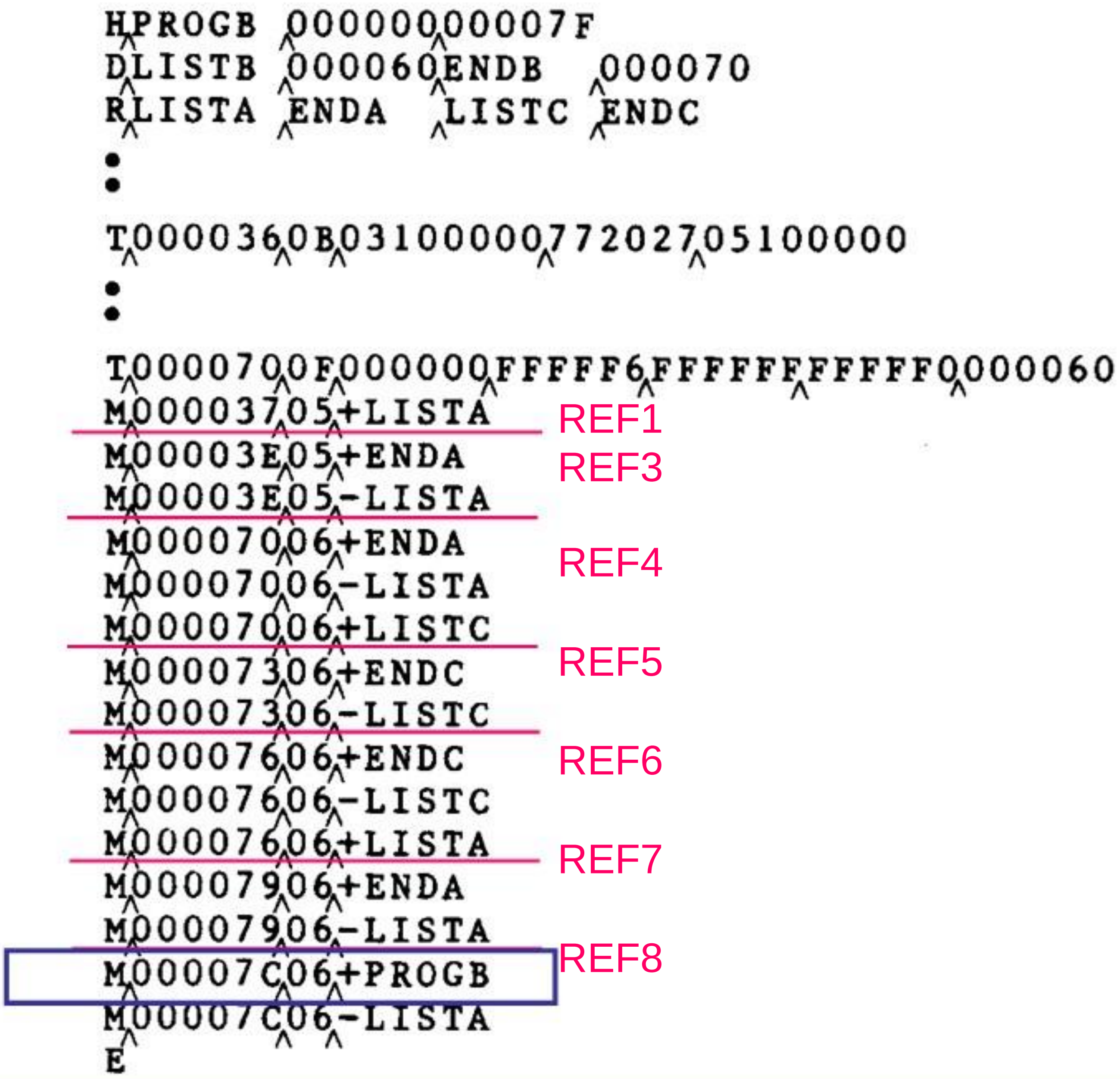
REF8

Figure 3.9 Object programs corresponding to Fig. 3.8.

```
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
```

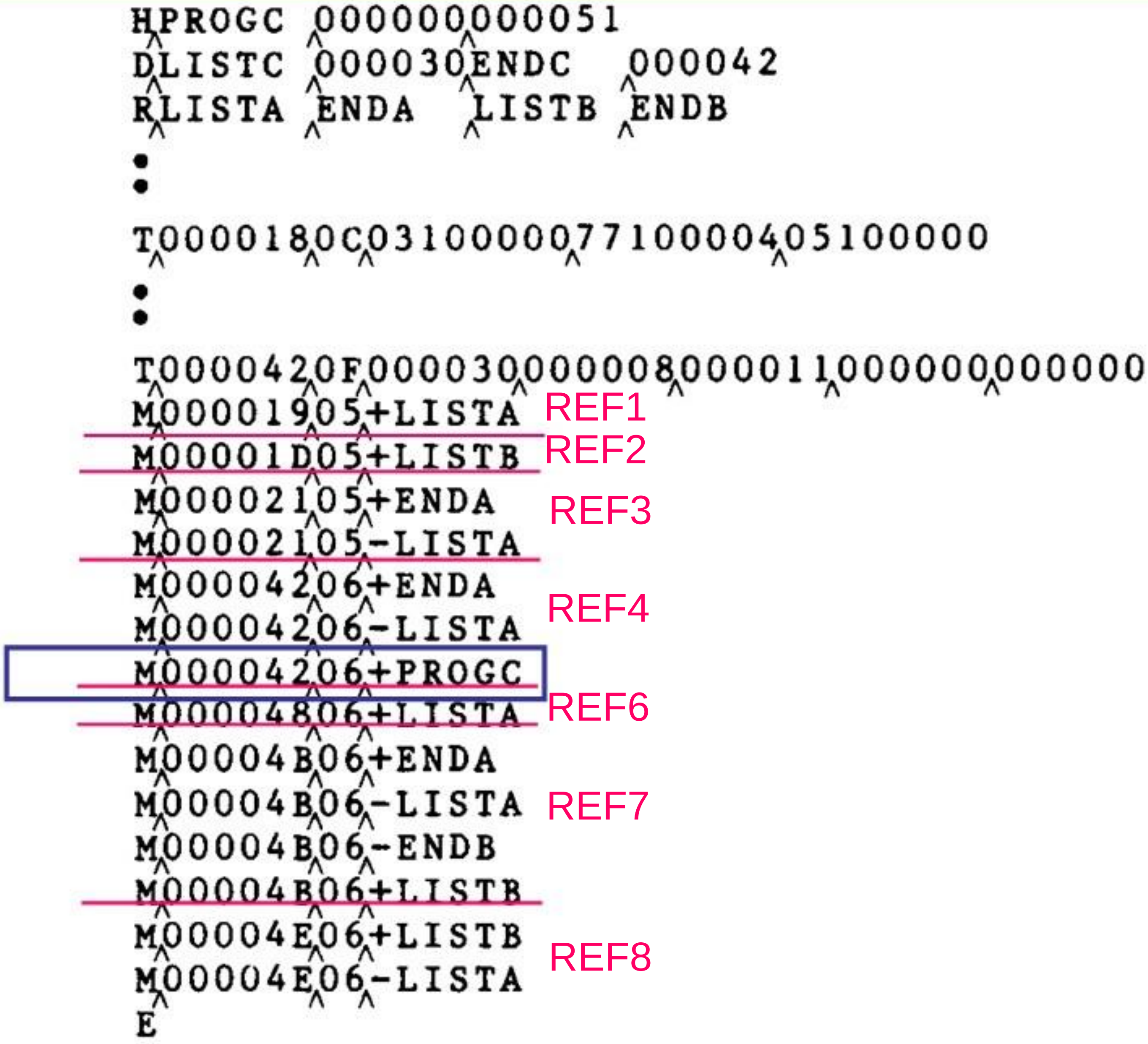
Figure 3.8 Sample programs illustrating linking and relocation.



```
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        REF1
M^00003E^05^+ENDA         REF3
M^00003E^05^-LISTA
M^000070^06^+ENDA
M^000070^06^-LISTA        REF4
M^000070^06^+LISTC
M^000073^06^+ENDC         REF5
M^000073^06^-LISTC
M^000076^06^+ENDC         REF6
M^000076^06^-LISTC
M^000076^06^+LISTA
M^000079^06^+ENDA         REF7
M^000079^06^-LISTA
M^00007C^06^+PROGB        REF8
M^00007C^06^-LISTA
E
```

```
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
```



```
H^PROGC ^000000^000051
D^LISTC ^000030^ENDC  ^000042
R^LISTA ^ENDA  ^LISTB ^ENDB
•
•
T^000018^0C^03100000^77100004^05100000
•
•
T^000042^0F^000030^000008^000011^000000^000000
M^000019^05^+LISTA   REF1
M^00001D^05^+LISTB   REF2
M^000021^05^+ENDA    REF3
M^000021^05^-LISTA
M^000042^06^+ENDA    REF4
M^000042^06^-LISTA
M^000042^06^+PROGC
M^000048^06^+LISTA   REF6
M^00004B^06^+ENDA
M^00004B^06^-LISTA   REF7
M^00004B^06^-ENDB
M^00004B^06^+LISTB
M^00004E^06^+LISTB   REF8
M^00004E^06^-LISTA
E
```

Figure 3.9 *(cont'd)*

3.2.2 Program Linking

- REF1,

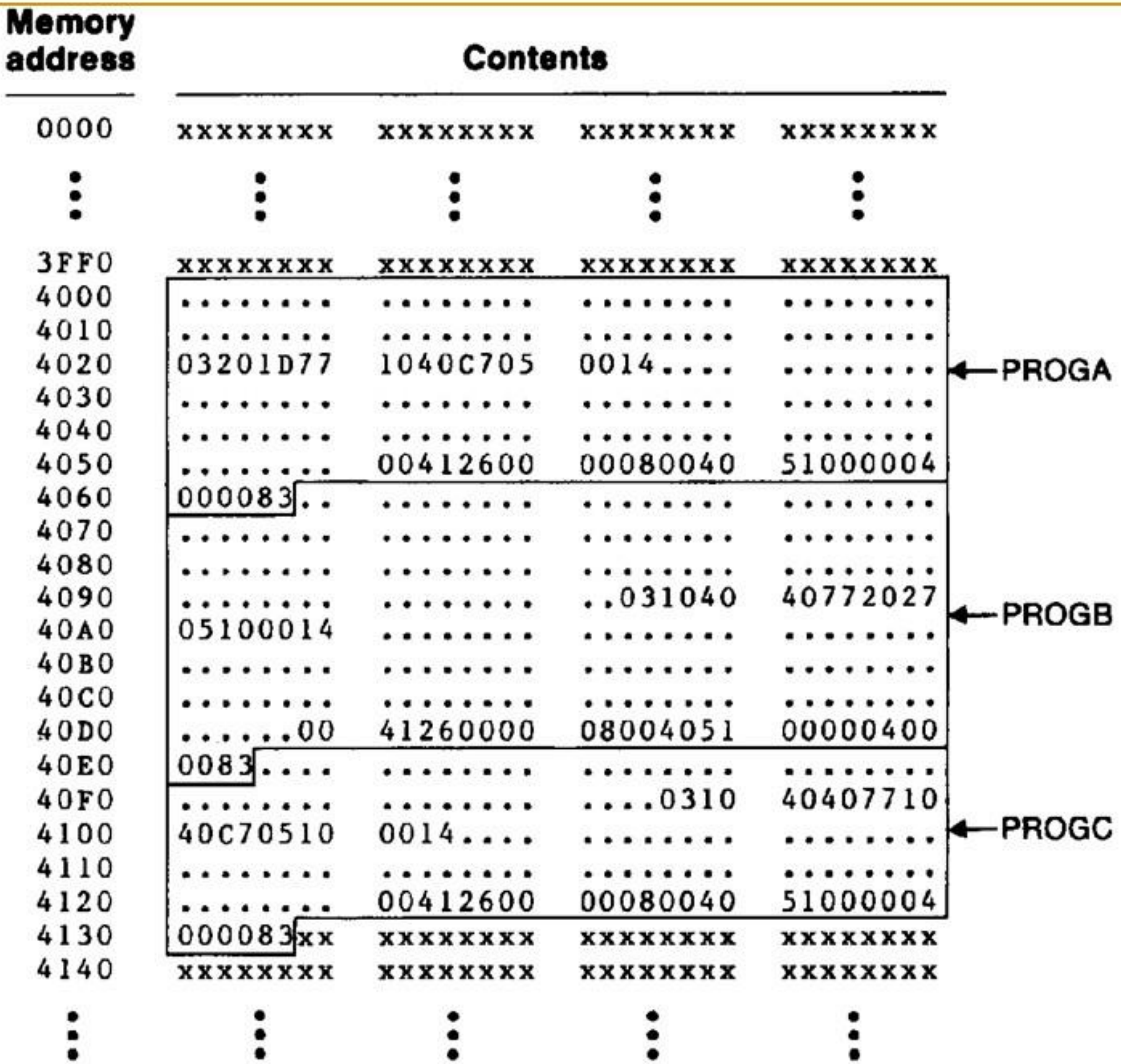
```
LDA    LISTA     03201D       03100000
```

  - In the PROGA, REF1 is simply a reference to a label.
  - In the PROGB and PROGC, REF1 is a reference to an external symbols.
- Need use extended format, Modification record.
- REF2 and REF3.

```
LDT       LISTB+4      772027              77100004
LDX       #ENDA-LISTA  050014              05100000
```

3.2.2 Program Linking

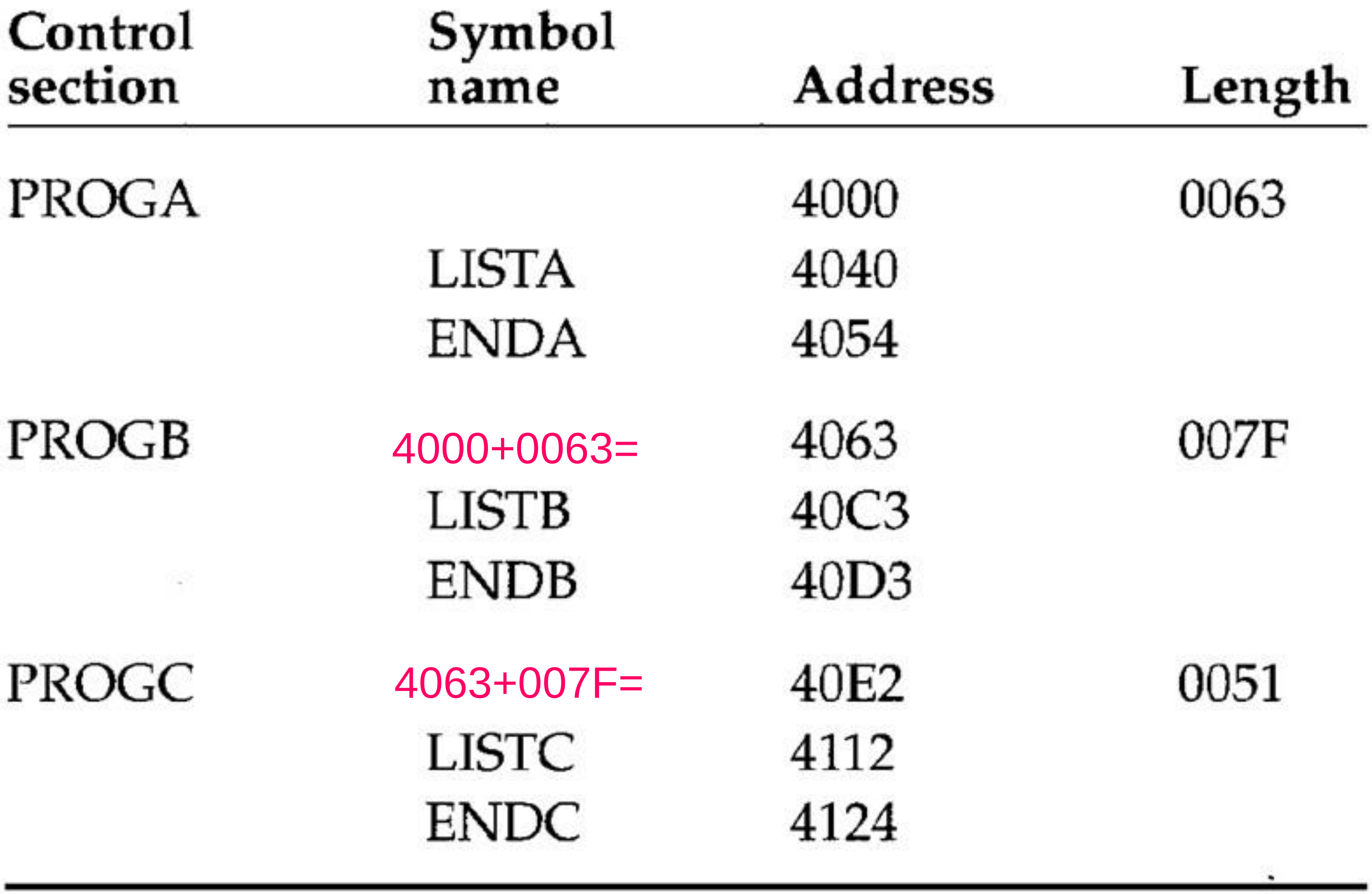
- REF4 through REF8,
  - `WORD      ENDA-LISTA+LISTC     000014+000000`
- Figure 3.10(a) and 3.10(b)
  - Shows these three programs as they might appear in memory after loading and linking.
  - PROGA 004000, PROGB 004063, PROGC 0040E2.
  - REF4 through REF8 in the same value.
  - For the references that are instruction operands, the calculated values after loading do not always appear to be equal.
  - Target address, REF1 4040.



```
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.



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

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

Ref No.	Symbol	Address
1	PROGC	40E2
2	LISTA	4040
3	ENDA	4054
4	LISTB	40C3
5	ENDB	40D3

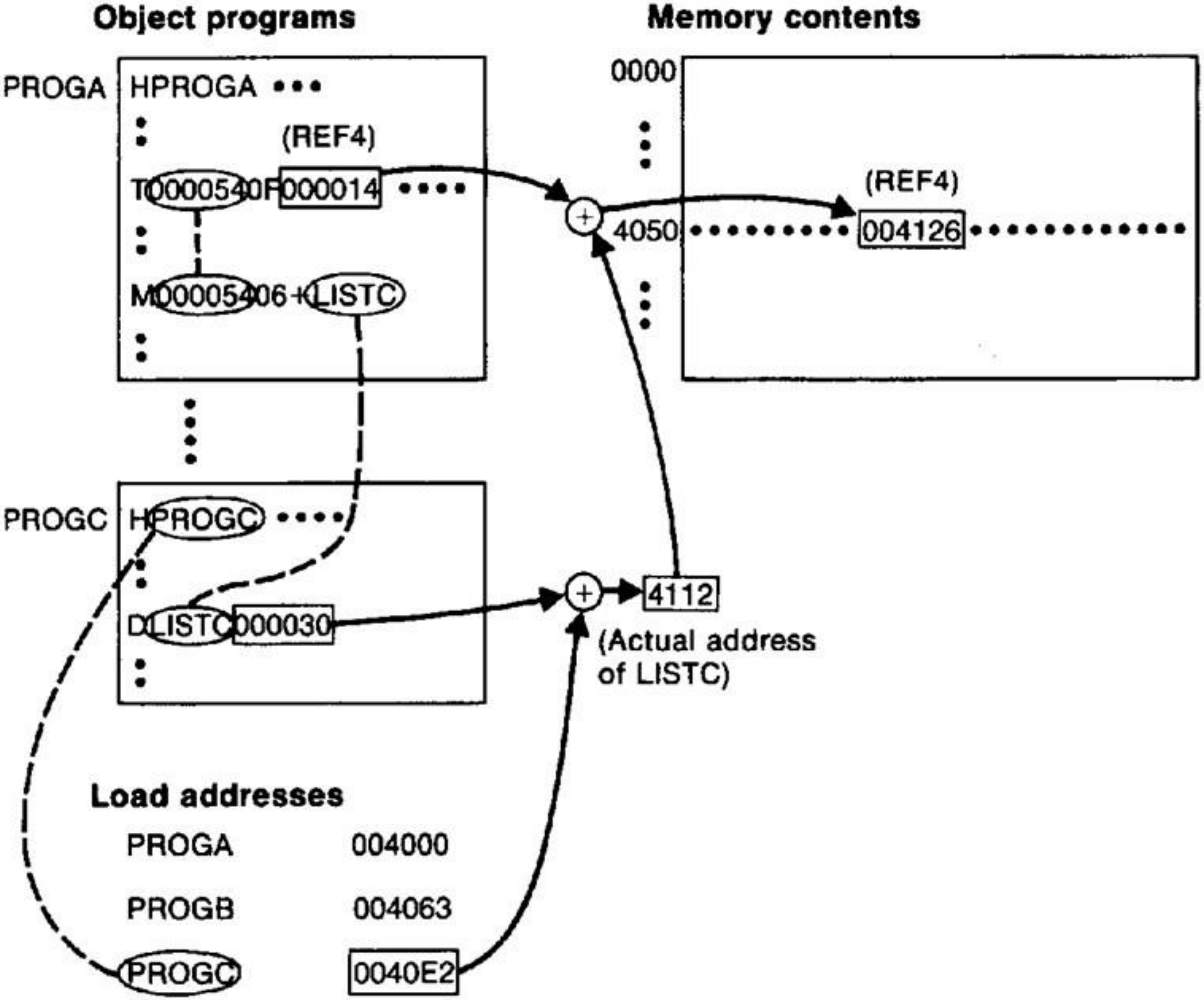


Figure 3.10(b) Relocation and linking operations performed on REF4 from PROGA.

3.2.3 Algorithm and Data Structure for a Linking Loader

- A linking loader usually makes two passes
  - Pass 1 assigns addresses to all external symbols by creating ESTAB.
  - Pass 2 performs the actual loading, relocation, and linking by using ESTAB.
  - The main data structure is ESTAB (hashing table).

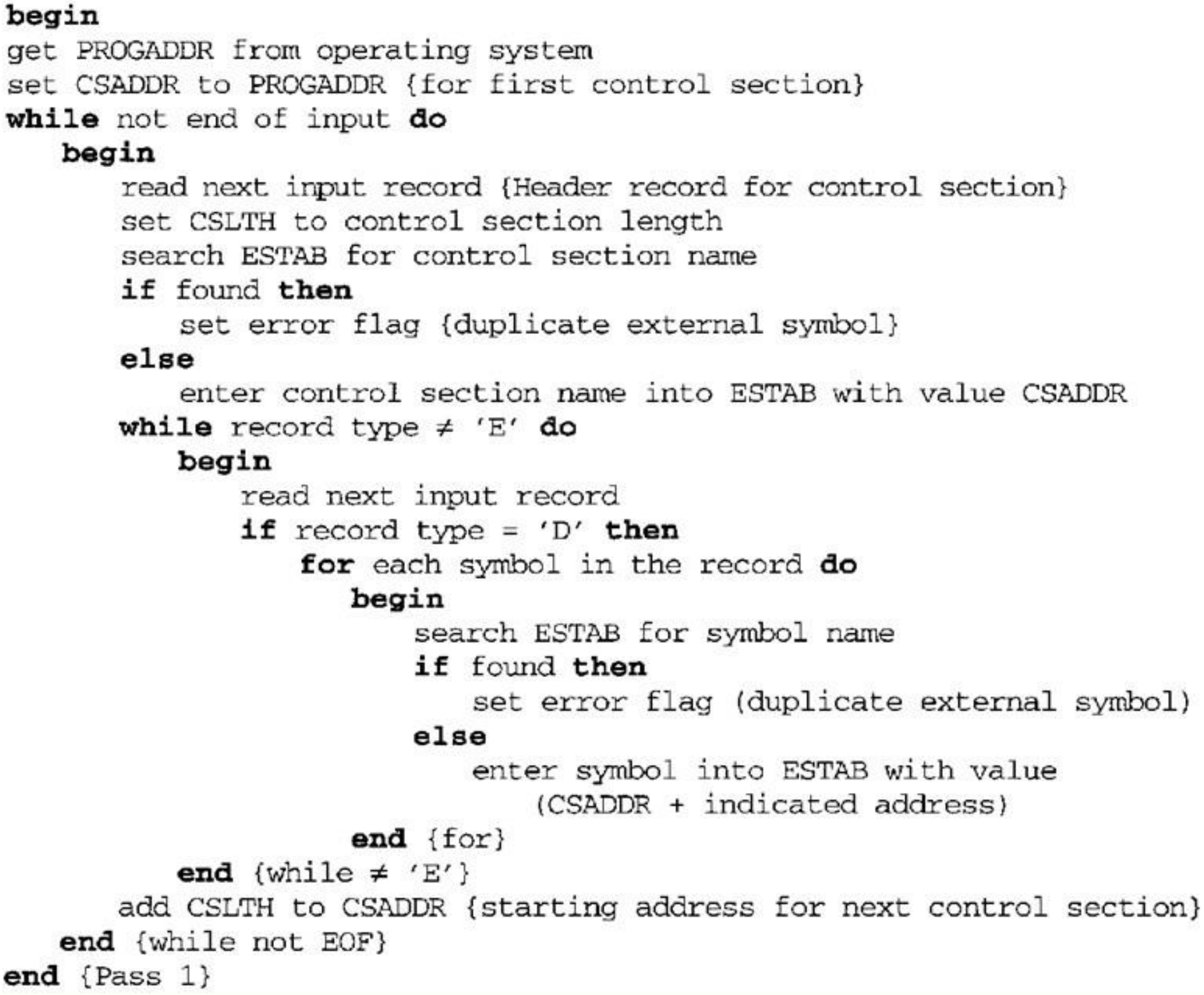
3.2.3 Algorithm and Data Structure for a Linking Loader

- A linking loader usually makes two passes
  - ESTAB is used to store the name and address of each external symbol in the set of control sections being loaded.
  - Two variables PROGADDR and CSADDR.
  - PROGADDR is the beginning address in memory where the linked program is to be loaded.
  - CSADDR contains the starting address assigned to the control section currently being scanned by the loader.

3.2.3 Algorithm and Data Structure for a Linking Loader

- The linking loader algorithm, Fig 3.11(a) & (b).
  - In Pass 1, concerned only Header and Defined records.
  - CSADDR+CSLTH = the next CSADDR.
  - A load map is generated.
  - In Pass 2, as each Text record is read, the object code is moved to the specified address (plus the current value of 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.

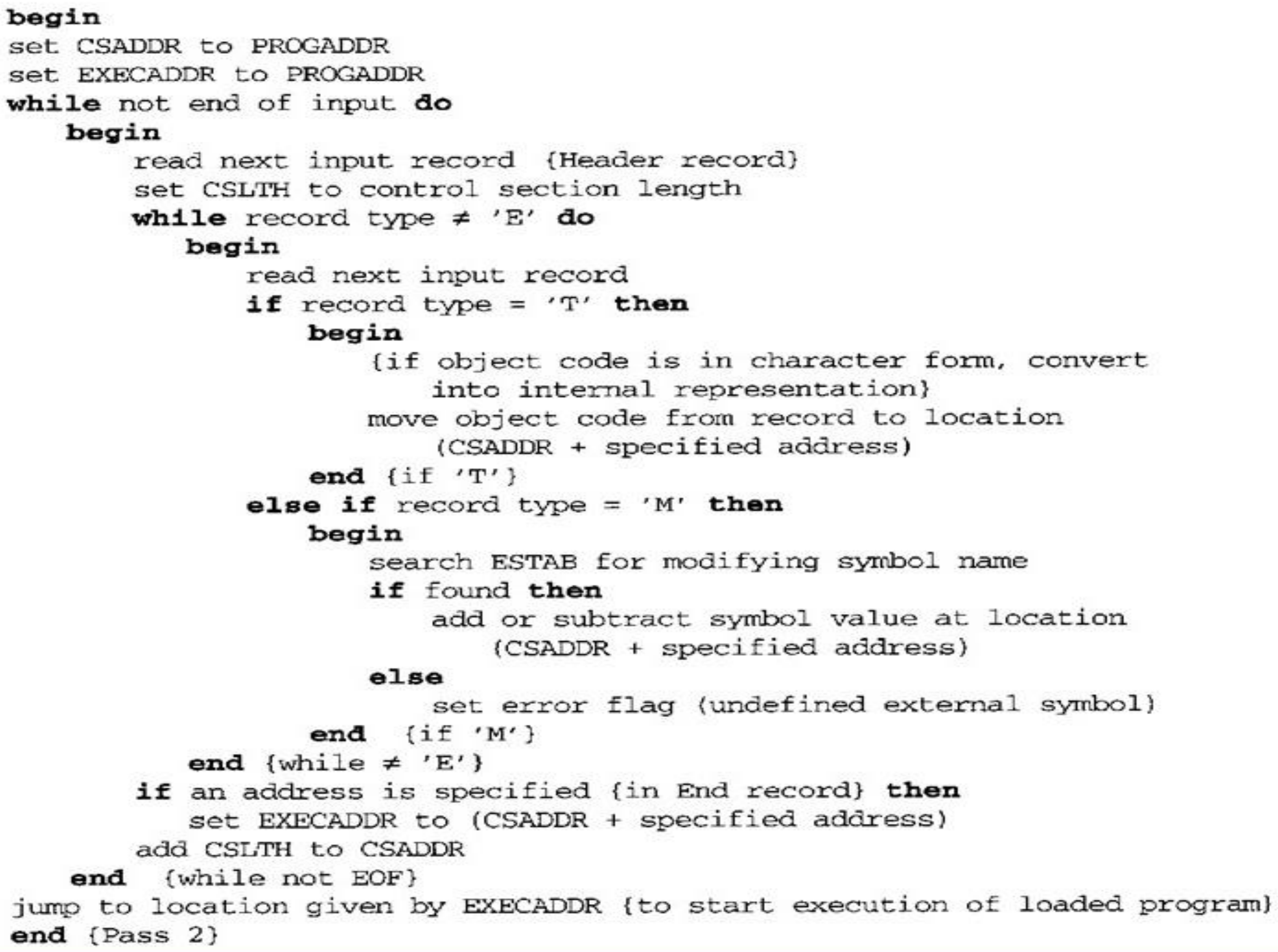
Pass 1:



```
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.

Pass 2:



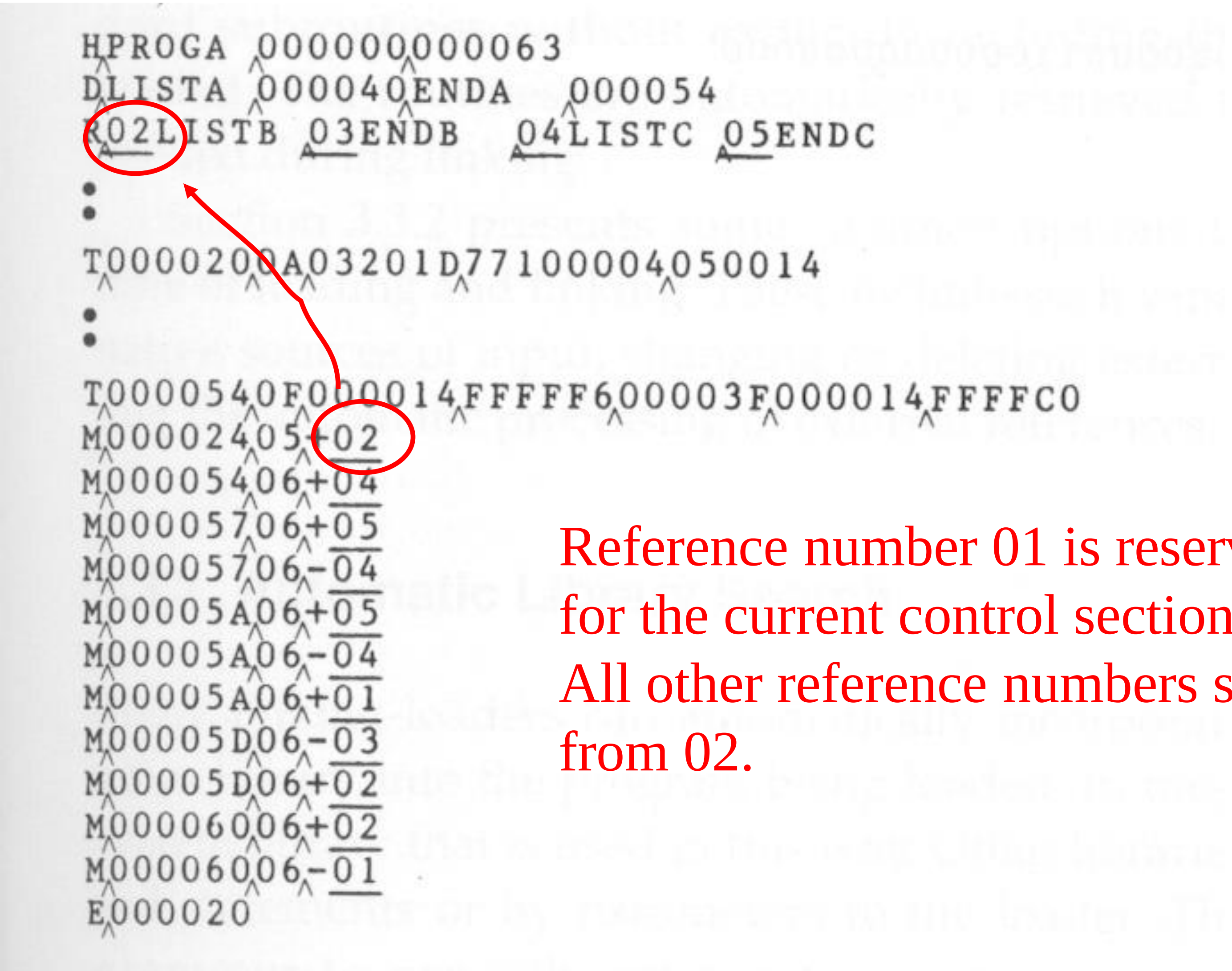
```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
   begin
      read next input record {Header record}
      set CSLTH to control section length
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'T' then
               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
   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.

3.2.3 Algorithm and Data Structure for a Linking Loader

- The algorithm can be made more efficient.
  - A *reference number*, is used in Modification records.
  - The number 01 to the control section name.
  - Figure 3.12, the main advantage of this reference-number mechanism is that it avoids multiple searches of ESTAB for the same symbol during the loading of a control section.

Reference Number Example



```
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
```

Reference number 01 is reserved for the current control section name. All other reference numbers start from 02.

```
H^PROGB ^000000^00007F
D^LISTB ^000060^ENDB  ^000070
R^02LISTA ^03ENDA ^04LISTC ^05ENDC
•
•
T^000036^0B^03100000^772027^05100000
•
•
T^000070^0F^000000^FFFFF6^FFFFFF^FFFFF0^000060
M^000037^05^+02
M^00003E^05^+03
M^00003E^05^-02
M^000070^06^+03
M^000070^06^-02
M^000070^06^+04
M^000073^06^+05
M^000073^06^-04
M^000076^06^+05
M^000076^06^-04
M^000076^06^+02
M^000079^06^+03
M^000079^06^-02
M^00007C^06^+01
M^00007C^06^-02
E
```

```
H^PROGC ^000000^000051
D^LISTC ^000030^ENDC  ^000042
R^02LISTA ^03ENDA  ^04LISTB ^05ENDB
•
•
T^000018^0C^03100000^77100004^05100000
•
•
T^000042^0F^000030^000008^000011^000000^000000
M^000019^05^+02
M^00001D^05^+04
M^000021^05^+03
M^000021^05^-02
M^000042^06^+03
M^000042^06^-02
M^000042^06^+01
M^000048^06^+02
M^00004B^06^+03
M^00004B^06^-02
M^00004B^06^-05
M^00004B^06^+04
M^00004E^06^+04
M^00004E^06^-02
E
```