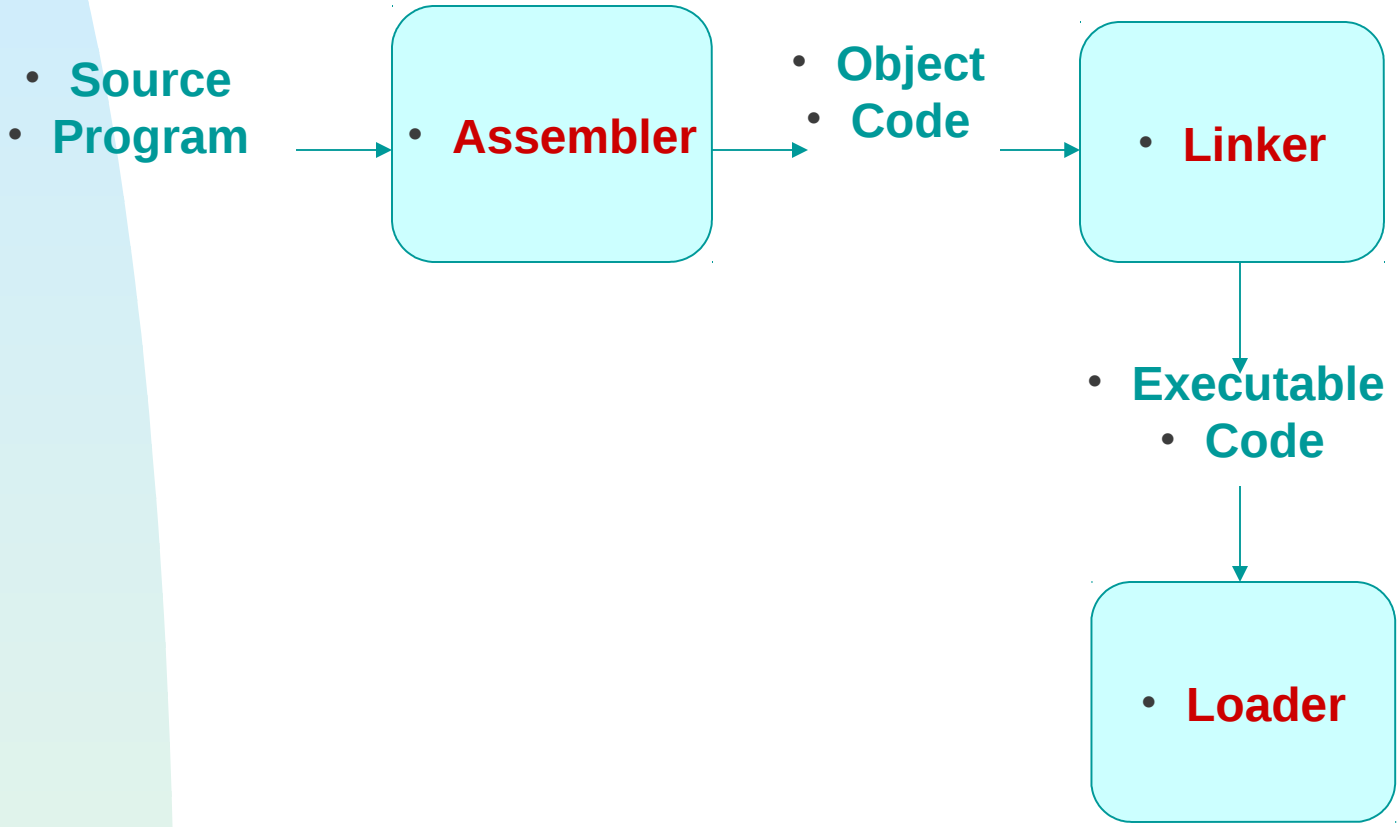
- Role of Assembler



- Introduction to Assemblers

  - Fundamental functions
    - translating mnemonic operation codes to their machine language equivalents
    - assigning machine addresses to symbolic labels

  - Machine dependency
    - different machine instruction formats and codes

- Example Program (Fig. 2.1)

- Purpose
  - reads records from input device (code F1)
  - copies them to output device (code 05)
  - at the end of the file, writes EOF on the output device, then RSUB to the operating system
  - program

- Example Program (Fig. 2.1)

- Data transfer (RD, WD)
  - a buffer is used to store record
  - buffering is necessary for different I/O rates
  - the end of each record is marked with a null character (0016)
  - the end of the file is indicated by a zero-length record
- Subroutines (JSUB, RSUB)
  - RDREC, WRREC
  - save link register first before nested jump

- Assembler Directives

- Pseudo-Instructions
  - Not translated into machine instructions
  - Providing information to the assembler
- Basic assembler directives
  - START
  - END
  - BYTE
  - WORD
  - RESB
  - RESW

- Object Program

- Header
  - Col. 1 H
  - Col. 2~7 Program name
  - Col. 8~13 Starting address (hex)
  - Col. 14-19 Length of object program in bytes (hex)
- Text
  - Col.1 T
  - Col.2~7 Starting address in this record (hex)
  - Col. 8~9 Length of object code in this record in bytes (hex)
  - Col. 10~69 Object code (69-10+1)/6=10 instructions
- End
  - Col.1 E
  - Col.2~7 Address of first executable instruction (hex)
  - (END program_name)

- Fig. 2.3

- H COPY 001000 00107A
- T 001000 1E 141033 482039 001036 281030 301015 482061 ...
- T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 000000
- T 002039 1E 041030 001030 E0205D 30203F D8205D 281030 …
- T 002057 1C 101036 4C0000 F1 001000 041030 E02079 302064 …
- T 002073 07 382064 4C0000 05
- E 001000

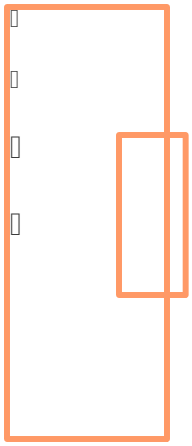
- Assembler’s functions

- Convert mnemonic operation codes to their machine language equivalents
- Convert symbolic operands to their equivalent machine addresses
- Build the machine instructions in the proper format
- Convert the data constants to internal machine representations
- Write the object program and the assembly listing

- Difficulties: Forward Reference

Forward reference: reference to a label that is defined later in the program.

Loc	Label	Operator	Operand
1000	FIRST	STL	RETADR
1003	CLOOP	JSUB	RDREC
…	…	…	…
1012		J	CLOOP
…	…	…	…
1033	RETADR	RESW	1

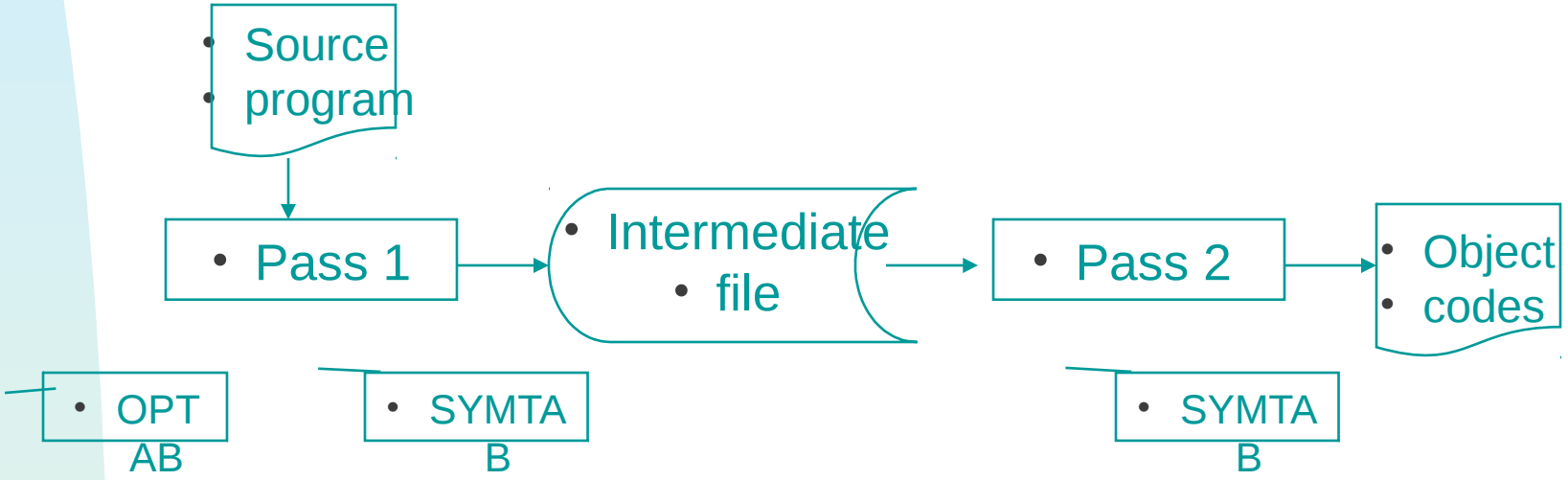


- Two Pass Assembler

  - Pass 1
    - Assign addresses to all statements in the program
    - Save the values assigned to all labels for use in Pass 2
    - Perform some processing of assembler directives
  - Pass 2
    - Assemble instructions
    - Generate data values defined by BYTE, WORD
    - Perform processing of assembler directives not done in Pass 1
    - Write the object program and the assembly listing

- Two Pass Assembler

  - Read from input line
    - LABEL, OPCODE, OPERAND



- Data Structures

- Symbol Table (SYMTAB)
- Location Counter(LOCCTR)

- SYMTAB (symbol table)

  - Content
    - label name, value, flag, (type, length) etc.
  - Characteristic
    - dynamic table (insert, delete, search)
  - Implementation
    - hash table, non-random keys, hashing funct

COPY	1000
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	1024
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039
RDREC	2039

Instruction Format and Addressing Mode

- SIC/XE
  - PC-relative or Base-relative addressing: op m
  - Indirect addressing: op @m
  - Immediate addressing: op #c
  - Extended format: +op m
  - Index addressing: op m,x
  - register-to-register instructions
  - larger memory -> multi-programming (program allocation)
- Example program

- Translation

- Register translation
  - register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
  - preloaded in SYMTAB
- Address translation
  - Most register-memory instructions use program counter relative or base relative addressing
  - Format 3: 12-bit address field
    - base-relative: 0~4095
    - pc-relative: -2048~2047
  - Format 4: 20-bit address field

- PC-Relative Addressing Modes

- PC-relative
  - 10 0000 FIRST STL RETADR 17202D
  - (14)16 1 1 0 0 1 0 (02D) 16
  - displacement= RETADR - PC = 30-3 = 2D
  - 40 0017 J CLOOP 3F2FEC
  - (3C)16 1 1 0 0 1 0 (FEC) 16
  - displacement= CLOOP-PC= 6 - 1A= -14= FEC

- Base-Relative Addressing Modes

- Base-relative
  - base register is under the control of the programmer
  - 12 LDB #LENGTH
  - 13 BASE LENGTH
  - 160 104E STCH BUFFER, X 57C003

  - (54)16 1 1 1 1 0 0 (003) 16
  - (54) 1 1 1 0 1 0 0036-1051= -101B16
  - displacement= BUFFER - B = 0036 - 0033 = 3
  - NOBASE is used to inform the assembler that the contents of the base register no longer be relied upon for addressing

- Immediate Address Translation

- Immediate addressing
  - 55 0020 LDA #3 010003
  - (00)16 0 1 0 0 0 0 (003) 16
  - 133 103C +LDT #4096 75101000
  - (74)16 0 1 0 0 0 1 (01000) 16

• Immediate Address Translation (Cont.)

- Immediate addressing
  - 12 0003 LDB #LENGTH 69202D
  - (68)16 0 1 0 0 1 0 (02D) 16
  - (68)16 0 1 0 0 0 0 (033)16 690033
  - the immediate operand is the symbol LENGTH
  - the address of this symbol LENGTH is loaded into register B
  - LENGTH=0033=PC+displacement=0006+02D
  - if immediate mode is specified, the target address becomes the operand

- Indirect Address Translation

  - Indirect addressing
    - target addressing is computed as usual (PC-relative or BASE-relative)
    - only the n bit is set to 1
    - 70 002A J @RETADR 3E2003
    - (3C)16 1 0 0 0 1 0 (003) 16
    - TA=RETADR=0030
    - TA=(PC)+disp=002D+0003

- Program Relocation

- Example Fig. 2.1
  - *Absolute program*, starting address 1000
  - e.g. 55 101B LDA THREE 00102D
  - Relocate the program to 2000
  - e.g. 55 101B LDA THREE 00202D
  - Each Absolute address should be modified
- Example Fig. 2.5:
  - Except for absolute address, the rest of the instructions need not be modified
  - not a memory address (immediate addressing)
  - PC-relative, Base-relative
  - The only parts of the program that require modification at load time are those that specify direct addresses

- Example

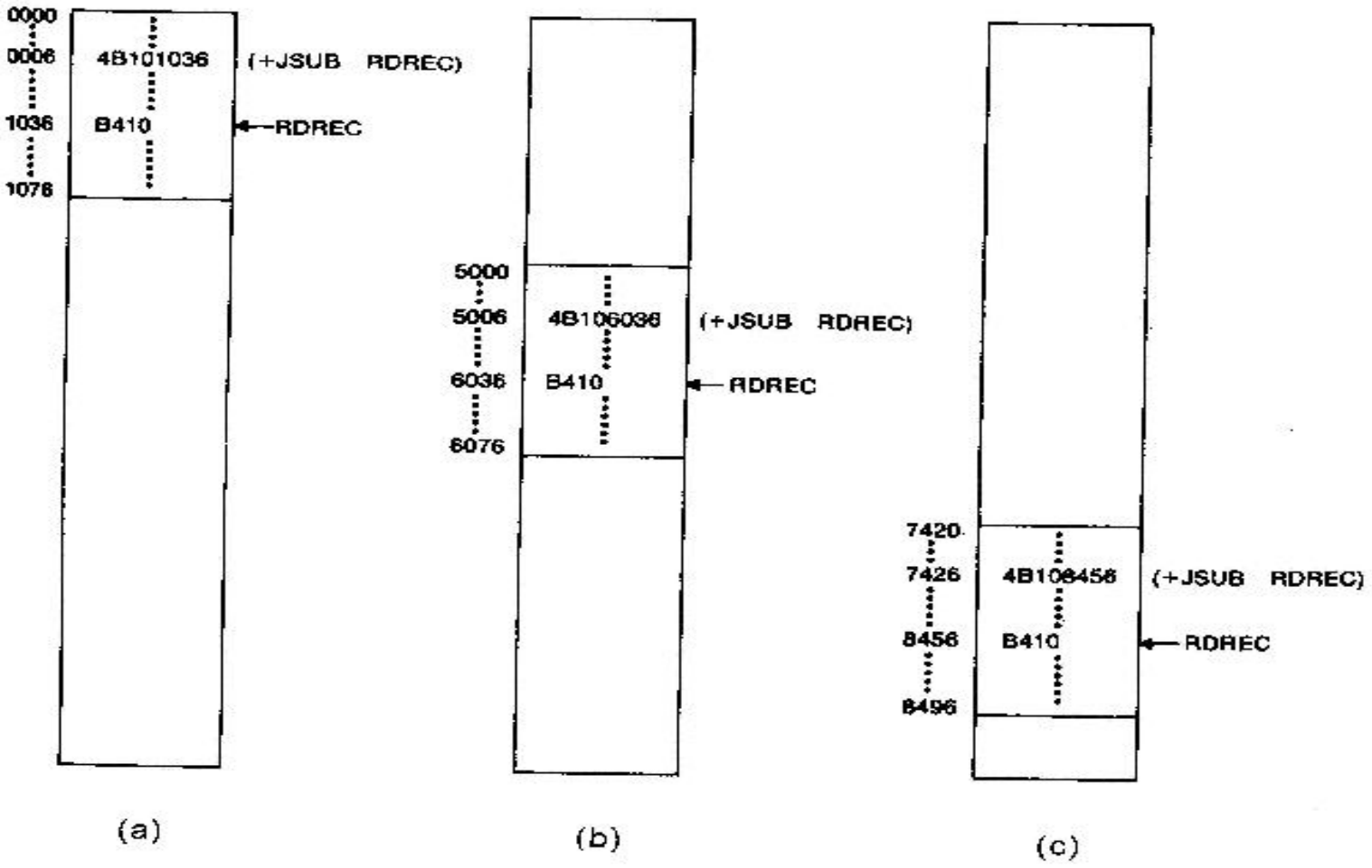


圖2.7 程式重定位範例

An SIC/XE Example (Figure 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   3E2FEC
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
```

Examples of Program Relocation (1/2)

- Example Fig. 2.2
  - *Absolute program*, starting address 1000

```
5    200         PYSTART   1000
10   200         RST    STL  RETADR      141033  142033
15   200         OOP    JSUB RDREC       482039  483039
20   200           LDA  LENGTH      001036  002036
25   200           COMP ZERO    281030  282030
30   200           JEQ  ENDFIL      301015  302015
35   200           JSUB WREC    482061  483061
40   201           J    CLOOP       3C1003  3C2003
45   201         DFIL   LDA  EOF       00102A  00202A
50   201           STA  BUFFER      0C1039  0C2039
55   201           LDA  THREE       00102D  00202D
60   201           STA  LENGTH      0C1036  0C2036
65   202           JSUB WREC    482061  483061
70   202           LDL  RETADR      081033  082033
75   202           RSUB         4C0000  4C0000
80   202         F  BYTE C'EOF'      454E46  454E46
85   202         REE    WORD 3       000003  000003
90   203         ROWORD 0         000000  000000
95   203         TADR   RESW 1
100  203         NGTH   RESW 1
105  203         FFER   RESB 4096
```

- Relocatable Program

- Modification record
  - Col 1 M
  - Col 2-7 Starting location of the address field to be modified, relative to the beginning of the program
  - Col 8-9 length of the address field to be modified, in half-bytes

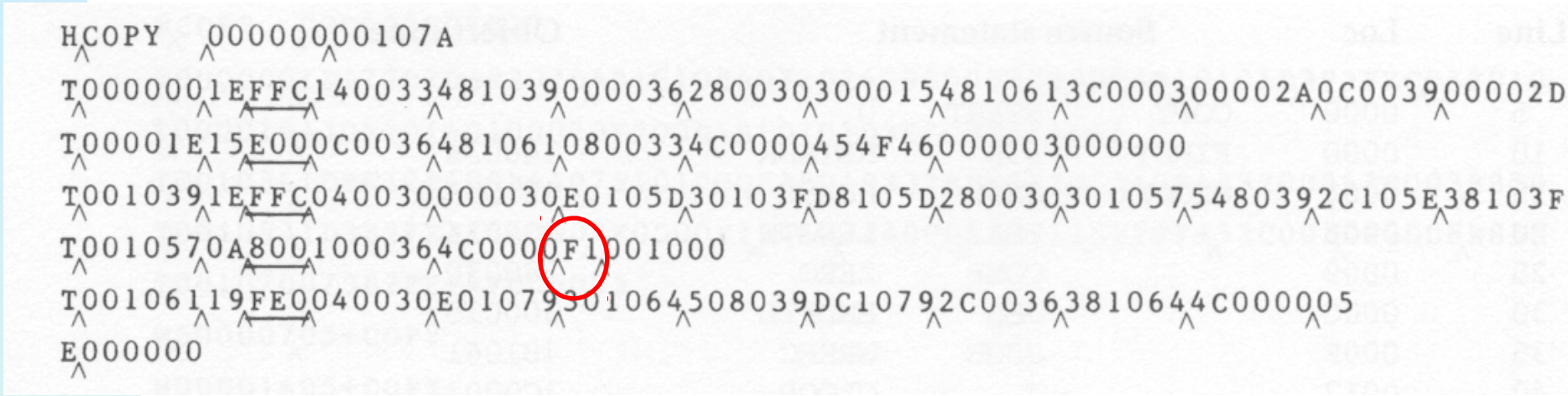
- Object Code

```
HCOPY  000000001077
T00000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T001036 1DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T001053 1D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T00107007 3B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

圖2.8　相對於圖2.6的目的程式

- End of Sec 2-2

- Relocation Bit Mask Example



- This one-byte “F1” makes the LDX instruction on line 210
- begins a new text record. This is because each relocation bit
- should be associated with a three-byte word. However,
- this data item occupies only one byte, which violates the
- Alignment rule.

- One-Pass Assemblers

  - Main problem
    - forward references
    - data items
    - labels on instructions
  - Solution
    - data items: require all such areas be defined before they are referenced
    - labels on instructions: no good solution

- Program Example

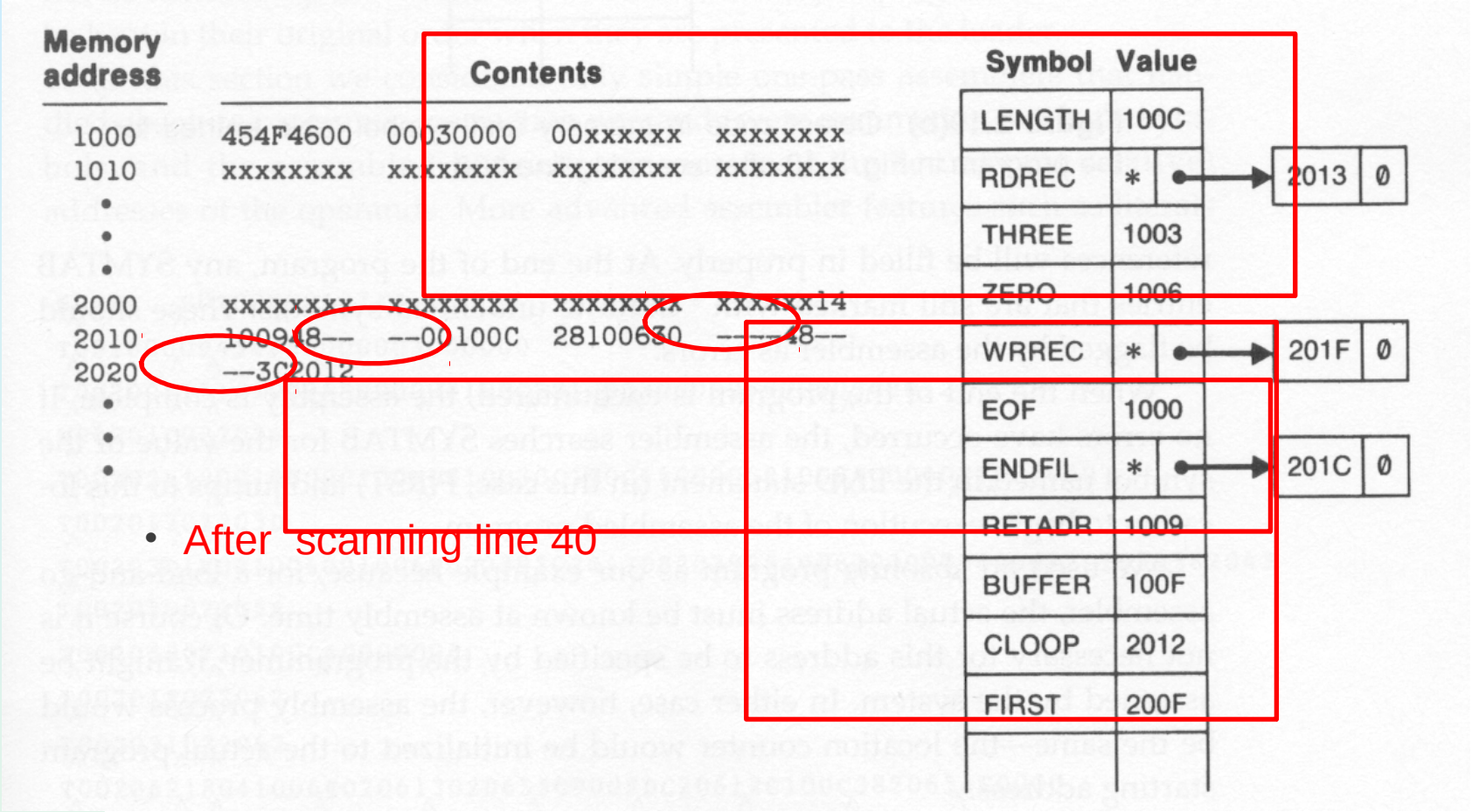
Line	Loc	Source statement			Object code
0	1000	COPY	START	1000	
1	1000	EOF	BYTE	C'EOF'	454F46
2	1003	THREE	WORD	3	000003
3	1006	ZERO	WORD	0	000000
4	1009	RETADR	RESW	1	
5	100C	LENGTH	RESW	1	
6	100F	BUFFER	RESB	4096	
9		.			
10	200F	FIRST	STL	RETADR	141009
15	2012	CLOOP	JSUB	RDREC	48203D
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	302024
35	201E		JSUB	WRREC	482062
40	2021		J	CLOOP	302012
45	2024	ENDFIL	LDA	EOF	001000
50	2027		STA	BUFFER	0C100F
55	202A		LDA	THREE	001003
60	202D		STA	LENGTH	0C100C
65	2030		JSUB	WRREC	482062
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000
110					

```
110                 .
115                 .     SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
121     2039    INPUT     BYTE    X'F1'           F1
122     203A    MAXLEN    WORD    4096            001000
124                 .
125     203D    RDREC     LDX     ZERO            041006
130     2040              LDA     ZERO            001006
135     2043    RLOOP     TD      INPUT           E02039
140     2046              JEQ     RLOOP           302043
145     2049              RD      INPUT           D82039
150     204C              COMP    ZERO            281006
155     204F              JEQ     EXIT            30205B
160     2052              STCH    BUFFER,X        54900F
165     2055              TIX     MAXLEN          2C203A
170     2058              JLT     RLOOP           382043
175     205B    EXIT      STX     LENGTH          10100C
180     205E              RSUB                    4C0000
195
```

```
195                 .
200                 .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205                 .
206     2061        OUTPUT  BYTE    X'05'           05
207                 .
210     2062        WRREC   LDX     ZERO            041006
215     2065        WLOOP   TD      OUTPUT          E02061
220     2068                JEQ     WLOOP           302065
225     206B                LDCH    BUFFER,X        50900F
230     206E                WD      OUTPUT          DC2061
235     2071                TIX     LENGTH          2C100C
240     2074                JLT     WLOOP           382065
245     2077                RSUB                    4C0000
255                         END     FIRST
```

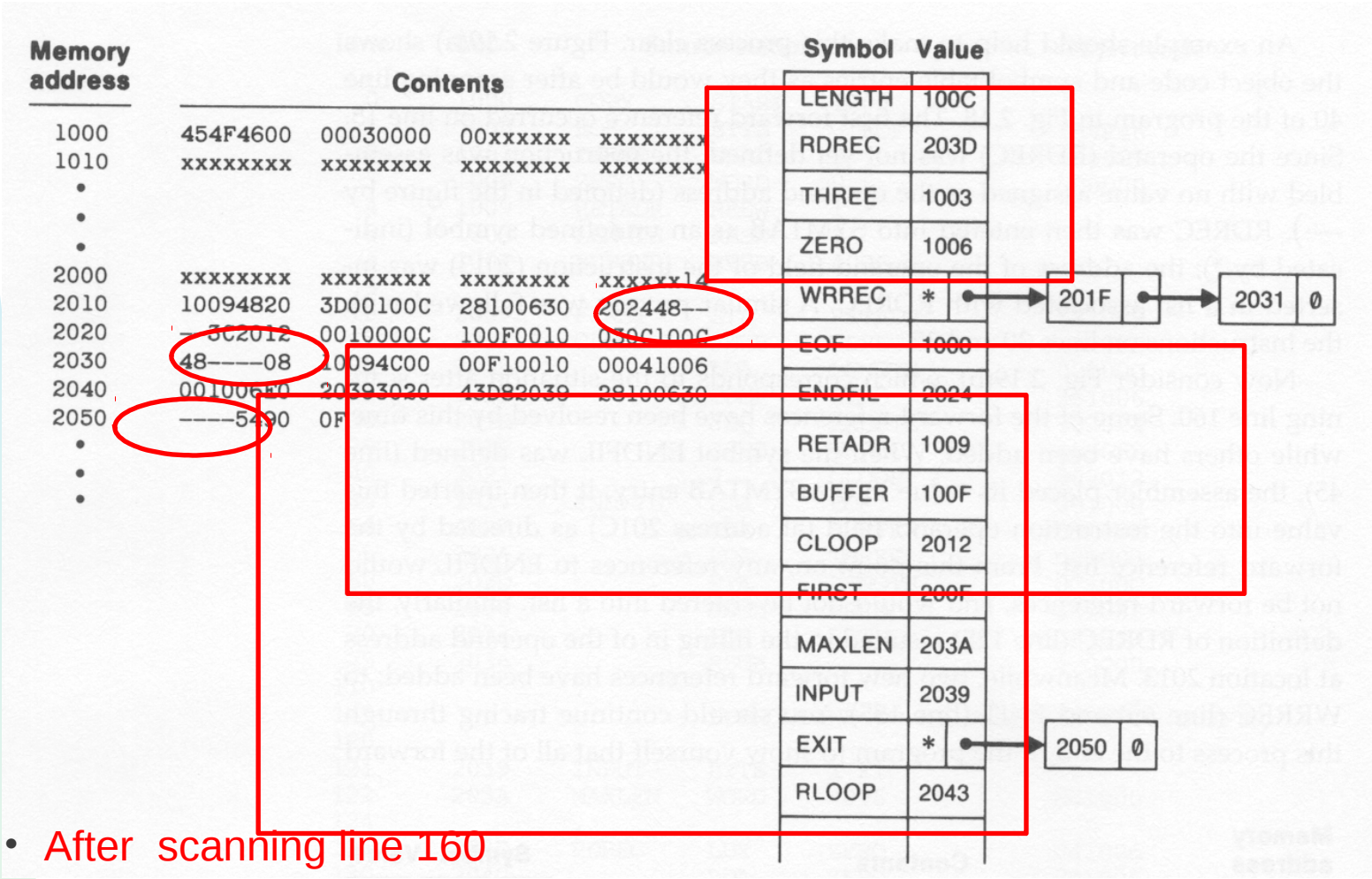
- All variables are defined before they are used.

- Processing Example



- After scanning line 40

- Processing Example (cont'd)



- After scanning line 160

- Processing Example (cont'd)

- Between scanning line 40 and 160:
  - On line 45, when the symbol ENDFIL is defined, the assembler places its value in the SYMTAB entry.
  - The assembler then inserts this value into the instruction operand field (at address 201C).
  - From this point on, any references to ENDFIL would not be forward references and would not be entered into a list.
- At the end of the processing of the program, any SYMTAB entries that are still marked with * indicate undefined symbols.
  - These should be flagged by the assembler as errors.

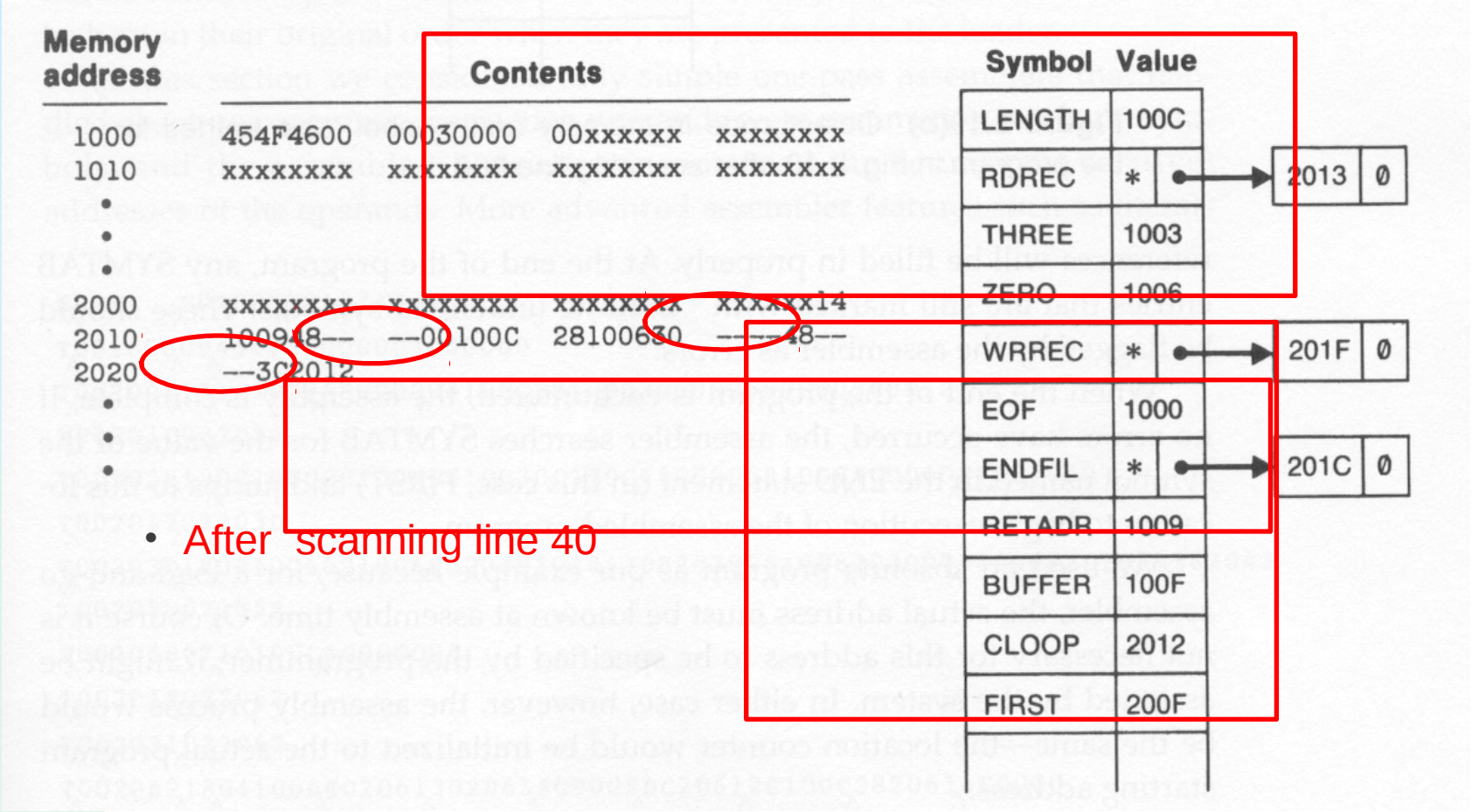
- Two Types

- There are two types of one-pass assembler:
  - Produce object code directly in memory for immediate execution
  - No loader is needed
  - Load-and-go for program development and testing
  - Good for computing center where most students reassemble their programs each time.
  - Can save time for scanning the source code again
  - Produce the usual kind of object program for later execution

- Internal Implementation

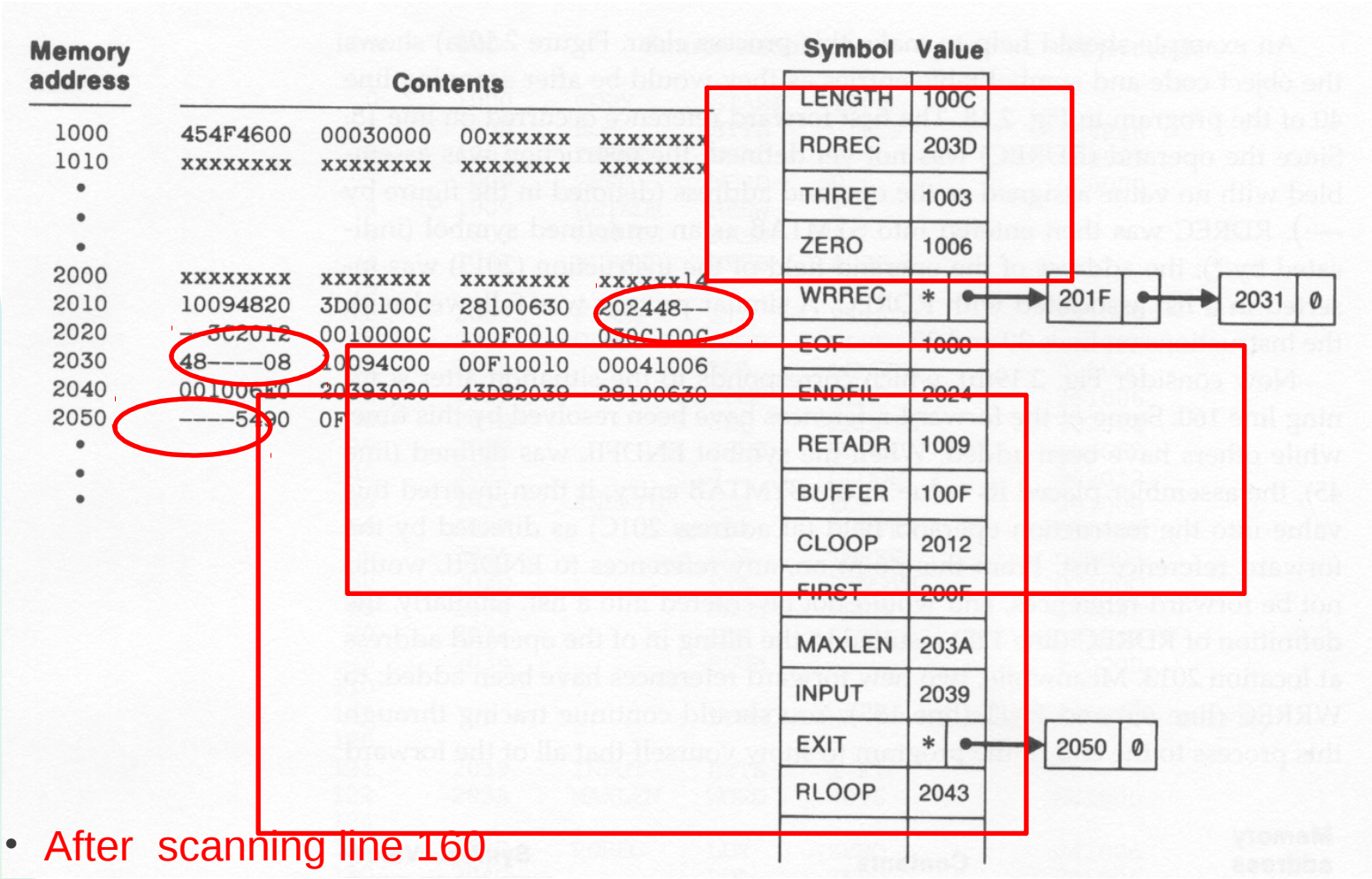
- The assembler generate object code instructions as it scans the source program.
- If an instruction operand is a symbol that has not yet been defined,the operand address is omitted when the instruction is assembled.
- The symbol used as an operand is entered into the symbol table.
- This entry is flagged to indicate that the symbol is undefined yet.

- Processing Example



- After scanning line 40

- Processing Example (cont'd)



- After scanning line 160

- Processing Example (cont'd)

- Between scanning line 40 and 160:
  - On line 45, when the symbol ENDFIL is defined, the assembler places its value in the SYMTAB entry.
  - The assembler then inserts this value into the instruction operand field (at address 201C).
  - From this point on, any references to ENDFIL would not be forward references and would not be entered into a list.
- At the end of the processing of the program, any SYMTAB entries that are still marked with * indicate undefined symbols.
  - These should be flagged by the assembler as errors.

- Load-and-go Assembler

- Characteristics
  - Useful for program development and testing
  - Avoids the overhead of writing the object program out and reading it back
  - Both one-pass and two-pass assemblers can be designed as load-and-go.
  - However one-pass also avoids the over head of an additional pass over the source program
  - For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program

- Three Working Items

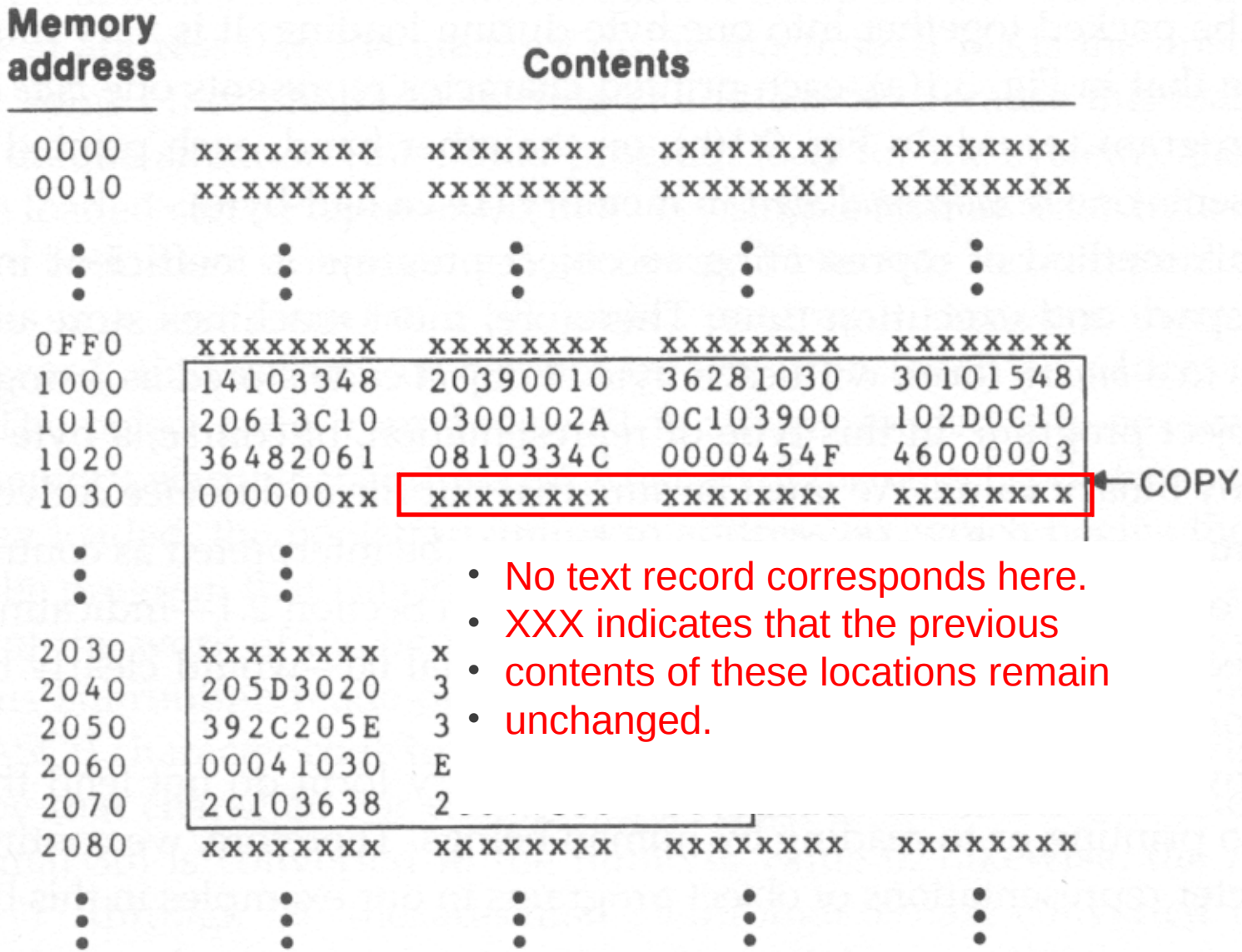
- Loading: loading an object program into memory for execution.
- Relocation: modify the object program so that it can be loaded at an address from the location originally specified.
- Linking: combines two or more separate object programs and supplies the information needed to allow references between them.
- A loader is a system program that performs the loading function. Many loaders also support relocation and linking. Some systems have a linker to perform the linking and a separate loader to handle relocation and loading.

- Absolute Loader

  - An object program is loaded at the address specified on the START directive.
  - No relocation or linking is needed
  - Thus is very simple

```
HCOPY  001000001007A
T001000 1E 141033 482039 001036 281030 301015 482061 3C1003 00102A 0C1039 00102D
T00101E 15 0C1036 482061 081033 4C0000 454F46 000003 000000
T002039 1E 041030 001030 E0205D 30203F D8205D 281030 302057 549039 2C205E 38203F
T002057 1C 101036 4C0000 F10010 000041 030E02 079302 064509 039DC2 079 2C1036
T002073 07 3820 644C 000005
E001000
```

(a) Object program



(b) Program loaded in memory

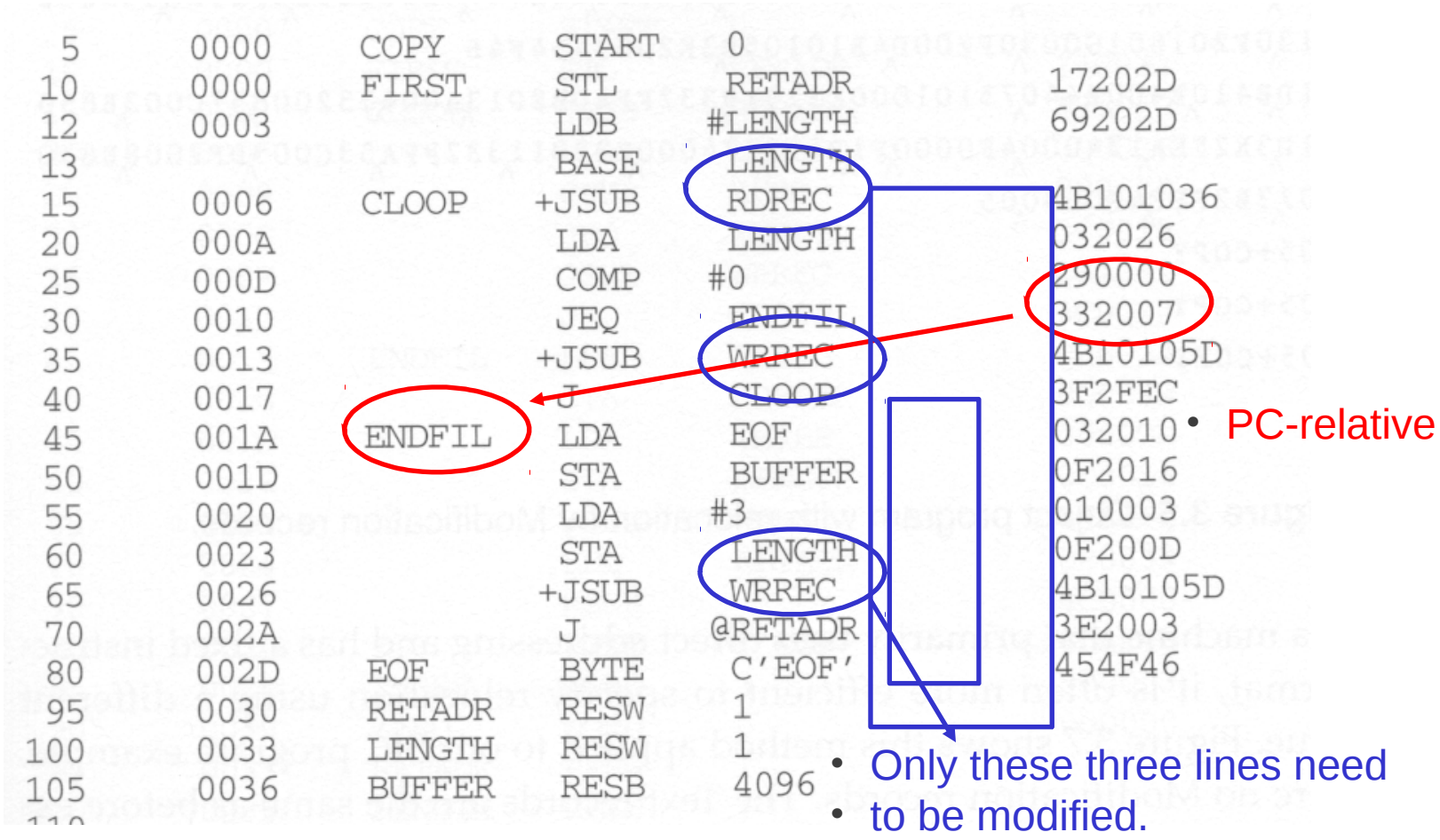
- Relocating Loader

- Two methods to describe where in the object program to modify the address (add the program starting address)
  - Use modification records
  - Suitable for a small number of changes
  - Use relocation bit mask
  - Suitable for a large number of changes

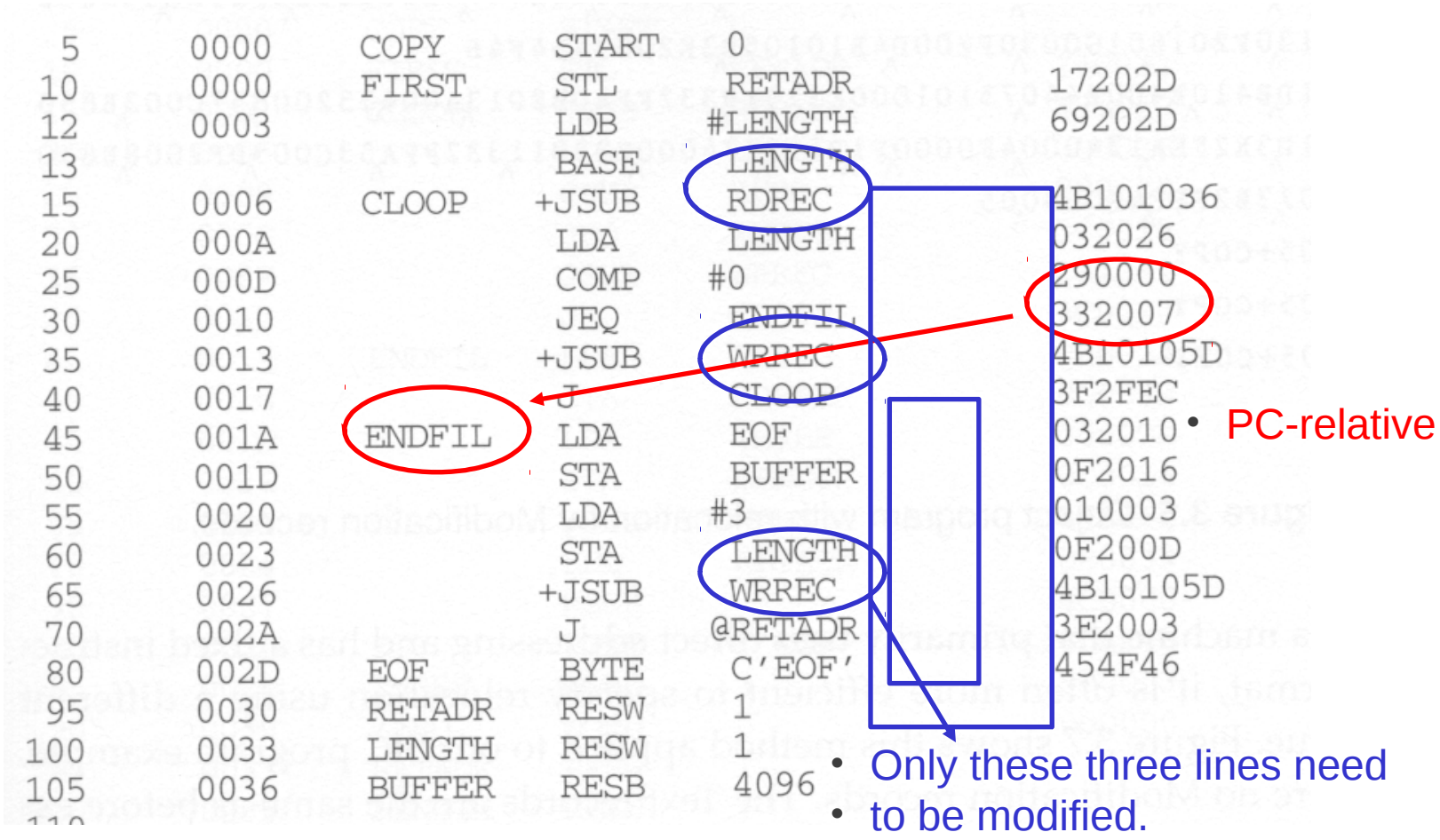
- Relocating Loader

- Two methods to describe where in the object program to modify the address (add the program starting address)
  - Use modification records
  - Suitable for a small number of changes
  - Use relocation bit mask
  - Suitable for a large number of changes

- Program Written in SIC/XE



- Program Written in SIC/XE



```
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
```

- Base-relative

```
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
```

- Base-relative

- This program is written in SIC/XE instructions. Program counter-
- relative and base-relative addressing are extensively used to
- avoid the need for many address modification records.

- The Object Program

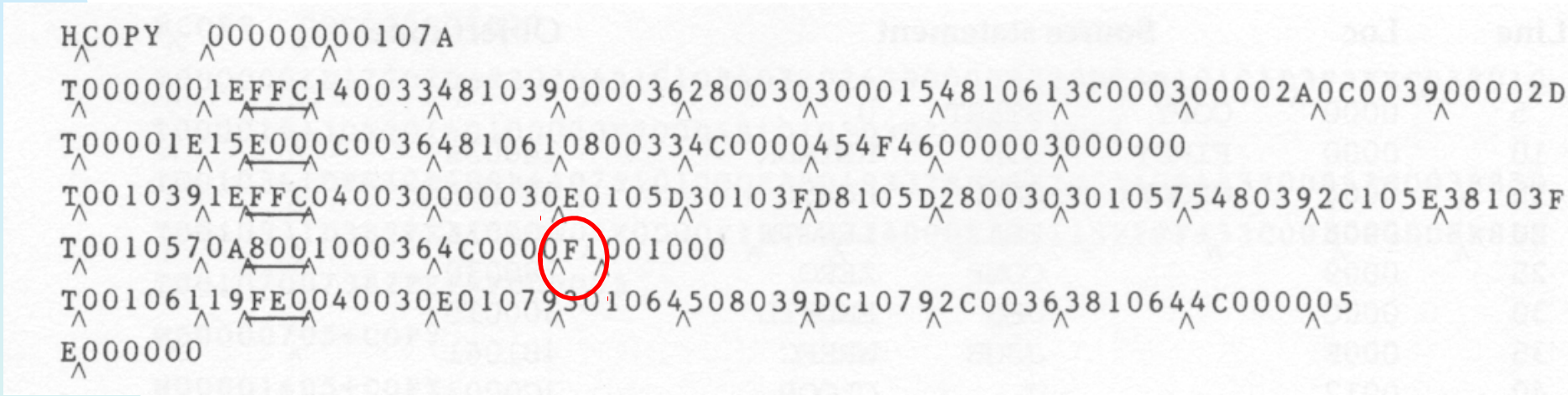
```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200074B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T00107007 3B2FEF4F000005
M00000705+COPY
M00001405+COPY
M00002705+COPY
E000000
```

- Only lines 15, 35, and 65 need to be modified.

- Relocation Bit Mask

- If an object needs too many modification records, it would be more efficient to use a relocation bit mask to indicate where in the object program should be modified when the object program is loaded.
- A relocation bit is associated with each word of object code. Since all SIC instructions occupy one word, this means that there is one relocation bit for each possible instruction.
- If the relocation bit corresponding to a word of object code is set to 1, the program's starting address will be added to this word when the program is relocated.

- Relocation Bit Mask Example



- This one-byte “F1” makes the LDX instruction on line 210
- begins a new text record. This is because each relocation bit
- should be associated with a three-byte word. However,
- this data item occupies only one byte, which violates the
- Alignment rule.

- Absolute Loader Implementation

```
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.

- "14" occupies two bytes if
- it is represented in char form.

- When loaded into
- memory, "14" should
- occupy only one byte.