CST 305- SYSTEM SOFTWARE

MODULE 3

Syllabus

Machine Dependent Assembler Features

- Instruction Format and Addressing Modes.
- Program Relocation.

Machine Independent Assembler Features

- Literals.
- Symbol Defining Statements.
- Expressions.
- Program Blocks.
- Control Sections and Program Linking.

Assembler Design Options

- One Pass Assembler, Multi Pass Assembler.

Implementation Example-MASM Assembler.

Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	FIRST	STL	RETADR	SAVE RETURN ADDRESS
12		LDB	#LENGTH	ESTABLISH BASE REGISTER
13		BASE	LENGTH	
15	CLOOP	+JSUB	RDREC	READ INPUT RECORD
20		LDA	LENGTH	TEST FOR EOF (LENGTH = 0)
25		COMP	#0	
30		JEQ	ENDFIL	EXIT IF EOF FOUND
35		+JSUB	WRREC	WRITE OUTPUT RECORD
40		J	CLOOP	LOOP
45	ENDFIL	LDA	EOF	INSERT END OF FILE MARKER
50		STA	BUFFER	
55		LDA	#3	SET LENGTH = 3
60		STA	LENGTH	
65		+JSUB	WRREC	WRITE EOF
70		J	@RETADR	RETURN TO CALLER
80	EOF	BYTE	C'EOF'	
95	RETADR	RESW	1	
100	LENGTH	RESW	1	LENGTH OF RECORD
105	BUFFER	RESB	4096	4096-BYTE BUFFER AREA
110	.			

```
115         .           SUBROUTINE TO READ RECORD INTO BUFFER
120         .
125         RDREC       CLEAR     X               CLEAR LOOP COUNTER
130                     CLEAR     A               CLEAR A TO ZERO
132                     CLEAR     S               CLEAR S TO ZERO
133                     +LDT      #4096
135         RLOOP       TD        INPUT           TEST INPUT DEVICE
140                     JEQ       RLOOP           LOOP UNTIL READY
145                     RD        INPUT           READ CHARACTER INTO REGISTER A
150                     COMPR     A,S             TEST FOR END OF RECORD (X'00')
155                     JEQ       EXIT            EXIT LOOP IF EOR
160                     STCH      BUFFER,X        STORE CHARACTER IN BUFFER
165                     TIXR      T               LOOP UNLESS MAX LENGTH
170                     JLT       RLOOP             HAS BEEN REACHED
175         EXIT        STX       LENGTH          SAVE RECORD LENGTH
180                     RSUB                      RETURN TO CALLER
185         INPUT       BYTE      X'F1'           CODE FOR INPUT DEVICE
195         .
```

```
200         .           SUBROUTINE TO WRITE RECORD FROM BUFFER
205         .
210         WRREC       CLEAR     X               CLEAR LOOP COUNTER
212                     LDT       LENGTH
215         WLOOP       TD        OUTPUT          TEST OUTPUT DEVICE
220                     JEQ       WLOOP           LOOP UNTIL READY
225                     LDCH      BUFFER,X        GET CHARACTER FROM BUFFER
230                     WD        OUTPUT          WRITE CHARACTER
235                     TIXR      T               LOOP UNTIL ALL CHARACTERS
240                     JLT       WLOOP             HAVE BEEN WRITTEN
245                     RSUB                      RETURN TO CALLER
250         OUTPUT      BYTE      X'05'           CODE FOR OUTPUT DEVICE
255                     END       FIRST
```

Figure 2.5 Example of a SIC/XE program.

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                .
```

```
HCOPY  000000001077
T00000001D17202D69202D4B10103603202629000033200 74B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B4107740 00E32011332FFA53C003DF2008B850
T001070073B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

Figure 2.8 Object program corresponding to Fig. 2.6.

Additional features of SIC/XE than the standard version

- Here we consider the design and implementation of an assembler for more complex XE version of SIC.
- **Indirect addressing** is indicated by adding the prefix @ to the operand (line70).
- **Immediate operands** are denoted with the prefix # (lines 25, 55,133).

- Instructions that refer to memory are normally assembled using either the **program counter relative or base counter relative mode.**

- The assembler directive **BASE** (line 13) is used in conjunction with base relative addressing.

- The four byte **extended instruction format** is specified with the **prefix +** added to the operation code in the source statement.

- Using register to register instructions instead of register to memory.

Advantages of SIC/XE

- Improve the execution speed of the program.
- Register-to-register instructions are faster than the corresponding register-to-memory operations because they are shorter and do not require another memory reference.
- While using immediate addressing, the operand is already present as part of the instruction and need not be fetched from anywhere.

Machine dependent assembler features

- Instruction Formats and Addressing Modes.
- Program Relocation.

Instruction Formats and Addressing Modes.

Translation of register to register instruction.

Convert into object code

- 125 RDREC CLEAR X B410
- 150 COMPR A,S A004

2.2.1 Instruction Formats and Addressing Modes

- START now specifies a beginning program address of 0
  - Indicate a *relocatable program*
- Register translation
  - For example: *COMPR A, S => A004*
  - Must keep the register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
    - Keep in SYMTAB

Explanation

- Assembler must simply convert the mnemonic operation code to machine language.(using OPTAB) and change each register mnemonic to its numeric equivalents.
- During pass 2 this conversion is possible.
- Register name and its value can be entered in SYMTAB.
- Register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9) are preloaded in SYMTAB.

Translation of register to memory instructions

- In SIC/XE machine there are 4 instruction format and 5 addressing modes.
- 1 byte instruction.
- 2 byte instruction.
- 3 byte instruction.
- 4 byte instruction. Both 3 & 4 are register to memory type of instructions.

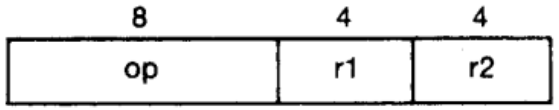
Addressing modes

- Program counter relative and base relative

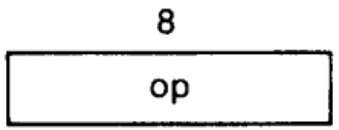
1.3.2 SIC/XE Machine Architecture

- **Instruction formats**
  - Relative addressing (相對位址) - format 3 (e=0)
  - Extend the address to 20 bits - format 4 (e=1)
  - Don't refer memory at all - formats 1 and 2

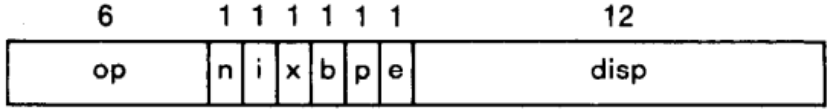
Format 2 (2 bytes):



Format 1 (1 byte):



Format 3 (3 bytes):



Format 4 (4 bytes):

1.3.2 SIC/XE Machine Architecture

- Addressing modes
  - n i x b p e
  - Simple n=0, i=0 (SIC) or n=1, i=1
  - Immediate n=0, i=1 TA=Valus
  - Indirect n=1, i=0 TA=(Operand)
  - Base relative b=1, p=0 TA=(B)+disp

    0 <= disp <= 4095
  - PC relative b=0, p=1 TA=(PC)+disp

    -2048 <= disp <= 2047

Mode	Indication	Target address calculation	
Base relative	b = 1, p = 0	TA = (B) + disp	(0 ≤ disp ≤ 4095)
Program-counter relative	b = 0, p = 1	TA = (PC) + disp	(–2048 ≤ disp ≤ 2047)

1.3.2 SIC/XE Machine Architecture

- Addressing mode
  - Direct — b=0, p=0 — TA=disp
  - Index — x=1 — $TA_{new}=TA_{old}+(X)$
  - Index+Base relative — x=1, b=1, p=0 — TA=(B)+disp+(X)
  - Index+PC relative — x=1, b=0, p=1 — TA=(PC)+disp+(X)
  - Index+Direct — x=1, b=0, p=0
  - Format 4 — e=1
- Appendix and Fig. 1.1 Example

Address Translation

- Most register-to-memory instructions are assembled using *PC relative* or *base relative* addressing
  - Assembler must calculate a *displacement* as part of the object instruction
  - If displacement can be fit into 12-bit field, format 3 is used.
  - Format 3: 12-bit address field
    - Base-relative: 0~4095
    - PC-relative: -2048~2047
  - Assembler attempts to translate using PC-relative first, then base-relative
    - If displacement in PC-relative is out of range, then try base-relative

Address Translation (Cont.)

- If displacement can not be fit into 12-bit field in the object instruction, format 4 must be used.
  - Format 4: 20-bit address field
  - No displacement need to be calculated.
    - 20-bit is large enough to contain the full memory address
  - Programmer must specify extended format: +op m
  - For example: *+JSUB RDREC => 4B101036*
    - *LOC(RDREC) = 1036, get it from SYMTAB*

Program counter relative addressing mode

- Usually format 3 is used.
- Instruction contains opcode followed by a 12 bit displacement value.
- **TA= (PC)+ disp**

Question

- 10 0000 FIRST STL RETADR
- 40 0017 J CLOOP

PC-Relative Addressing Modes

- *10 0000 FIRST STL RETADR 17202D*
  - Displacement= RETADR – (PC) = 30-**3** = 2D
  - Opcode (6 bits) =14_{16}=00010100_2
  - nixbpe=110010
    - n=1, i = 1: indicate neither *indirect* nor *immediate* addressing
    - p = 1: indicate *PC-relative* addressing

OPCODE	n	i	x	b	p	e	Address
0001 01	1	1	0	0	1	0	$(02D)_{16}$

Object Code = 17202D

PC-Relative Addressing Modes (Cont.)

- 40 0017 J CLOOP 3F2FEC
  - Displacement= CLOOP - (PC) = 6 - **1A** = -14 = FEC (2's complement for negative number)
  - Opcode=$3C_{16}$ = 00111100_2
  - nixbpe=110010

OPCODE	n	i	x	b	p	e	Address
0011 11	1	1	0	0	1	0	$(FEC)_{16}$

Object Code = 3F2FEC

Base-Relative Addressing Modes

- Base register is under the control of the programmer
  - Programmer use assembler directive ***BASE*** to specify which value to be assigned to base register (B)
  - Assembler directive ***NOBASE***: inform the assembler that the contents of base register no longer be used for addressing
  - ***BASE*** and ***NOBASE*** produce no executable code

Base relative addressing mode

- Usually format 3 is used.
- Instruction contains opcode followed by a 12 bit displacement value.

TA= (B)+ disp

- This addressing mode is used when the range of displacement value is not sufficient.
- Whenever this mode is used it is indicated by using a directive **BASE.**
- The moment the assembler encounters this directive base relative addressing modes to calculate target address of the operand.
- When **NOBASE** directive is used then it indicates the base register is no more used to calculate the target address of the operand.

- The statement BASE LENGTH (line 13) informs the assembler that the base register will contain address of LENGTH .
- The preceding instruction LDB #LENGTH loads this value into the register during program execution.
- The assembler assumes for addressing purposes that register B contains this address until it encounters another BASE statement.
- Later in the program it may be desirable to use register B for another purposes.

Question

- Assemble the following instruction

160	104E		STCH	BUFFER,X
175	1056	EXIT	STX	LENGTH

Base-Relative Addressing Modes (Cont.)

- ***12 LDB #LENGTH***
- ***13 BASE LENGTH ;no object code***
- ***160 104E STCH BUFFER, X 57C003***
  - Displacement= BUFFER – (B) = 0036 – 0033(=LOC(LENGTH)) = 3
  - Opcode=54
  - nixbpe=111100
    - n=1, i = 1: indicate neither *indirect* nor *immediate* addressing
    - x = 1: *indexed* addressing
    - b = 1: *base-relative* addressing

OPCODE	n	i	x	b	p	e	Address
0101 01	1	1	1	1	0	0	$(003)_{16}$

Object Code = 57C003

Address Translation

- Assembler attempts to translate using *PC-relative* first, then *base-relative*
  - e.g. 175 1053 STX LENGTH 134000
    - Try PC-relative first
      - Displacement= LENGTH - (PC) = 0033 - 1056 = -1026 (hex)
    - Try base-relative next
      - displacement= LENGTH – (B) = 0033 – 0033 =0
      - Opcode=10
      - nixbpe=110100
        - n=1, i = 1: indicate neither *indirect* nor *immediate* addressing
        - b = 1: *base-relative* addressing

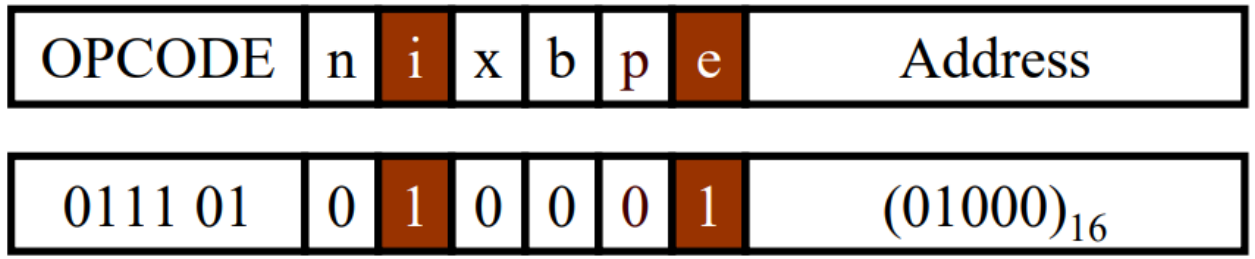
Immediate addressing modes

- In this mode no memory reference is involved , if immediate mode is used target address is the operand itself.

- 55 0020 LDA #3

Immediate Address Translation (Cont.)

- 133 103C +LDT #4096 75101000
  - Opcode=74
  - nixbpe=010001
    - i = 1: *immediate addressing*
    - e = 1: *extended instruction format* since 4096 is too large to fit into the 12-bit displacement field



Object Code = 75101000

Immediate Address Translation (Cont.

- 12 0003 LDB #LENGTH 69202D
  - The immediate operand is the symbol LENGTH
    - The address of LENGTH is loaded into register B
  - Displacement=LENGTH – (PC) = 0033 – 0006 = 02D
  - Opcode=68_{16} = 01101000_2
  - nixbpe=010010
    - Combined *PC relative* (p=1) with *immediate addressing* (i=1)

OPCODE	n	i	x	b	p	e	Address
0110 10	0	1	0	0	1	0	$(02D)_{16}$

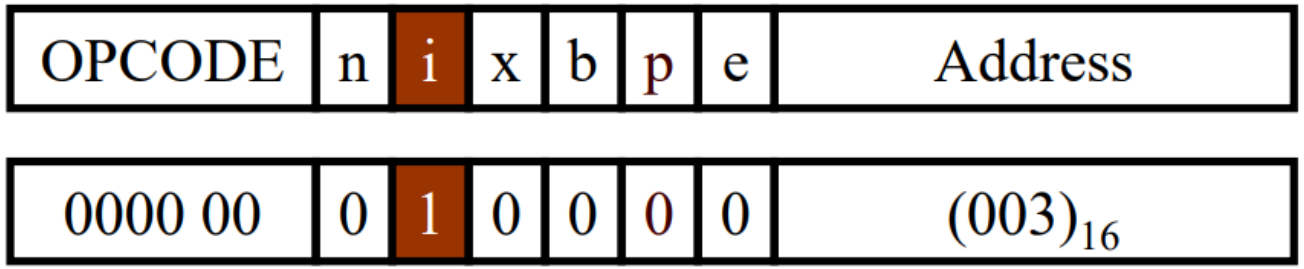
Immediate Address Translation (Cont.)

- 55 0020 LDA #3 010003
  - Opcode = $00_{16} = 00000000_2$
  - nixbpe=010000
    - i = 1: immediate addressing

OPCODE	n	i	x	b	P	e	Address
0110 10	0	1	0	0		0	$(02D)_{16}$

Immediate Address Translation

- Convert the *immediate* operand to its internal representation and insert it into the instruction
- 55 0020 LDA #3 010003
  - Opcode=00
  - nixbpe=010000
    - i = 1: *immediate addressing*



Object Code = 010003

Indirect Address Translation

- Indirect addressing
  - The contents stored at the location represent the *address* of the operand, not the operand itself
  - Target addressing is computed as usual (PC-relative or BASE-relative)
  - *n* bit is set to 1

Indirect Address Translation (Cont.

- 70 002A J @RETADR 3E2003
  - Displacement= RETADR- (PC) = 0030 – 002D =3
  - Opcode= 3C
  - nixbpe=100010
    - n = 1: *indirect addressing*
    - p = 1: *PC-relative addressing*

OPCODE	n	i	x	b	p	e	Address
0011 11	1	0	0	0	1	0	$(003)_{16}$

Indirect and program counter relative

- 70 002A J @RETADR

2.2.2 Program Relocation

- The larger main memory of SIC/XE
  - Several programs can be loaded and run at the same time.
  - This kind of sharing of the machine between programs is called ***multiprogramming***
- To take full advantage
  - Load programs into memory wherever there is room
  - Not specifying a fixed address at assembly time
  - Called ***program relocation***

2.2.2 Program Relocation (Cont.)

- *Absolute program* (or *absolute assembly*)
  - Program must be loaded at the address specified *at assembly time.*
  - E.g. Fig. 2.1

    - e.g. 55 101B LDA THREE 00102D

  - What if the program is loaded to 2000

    e.g. 55 101B LDA THREE 00202D

    - Each absolute address should be modified

Example of Program Relocation (Fig 2.7)

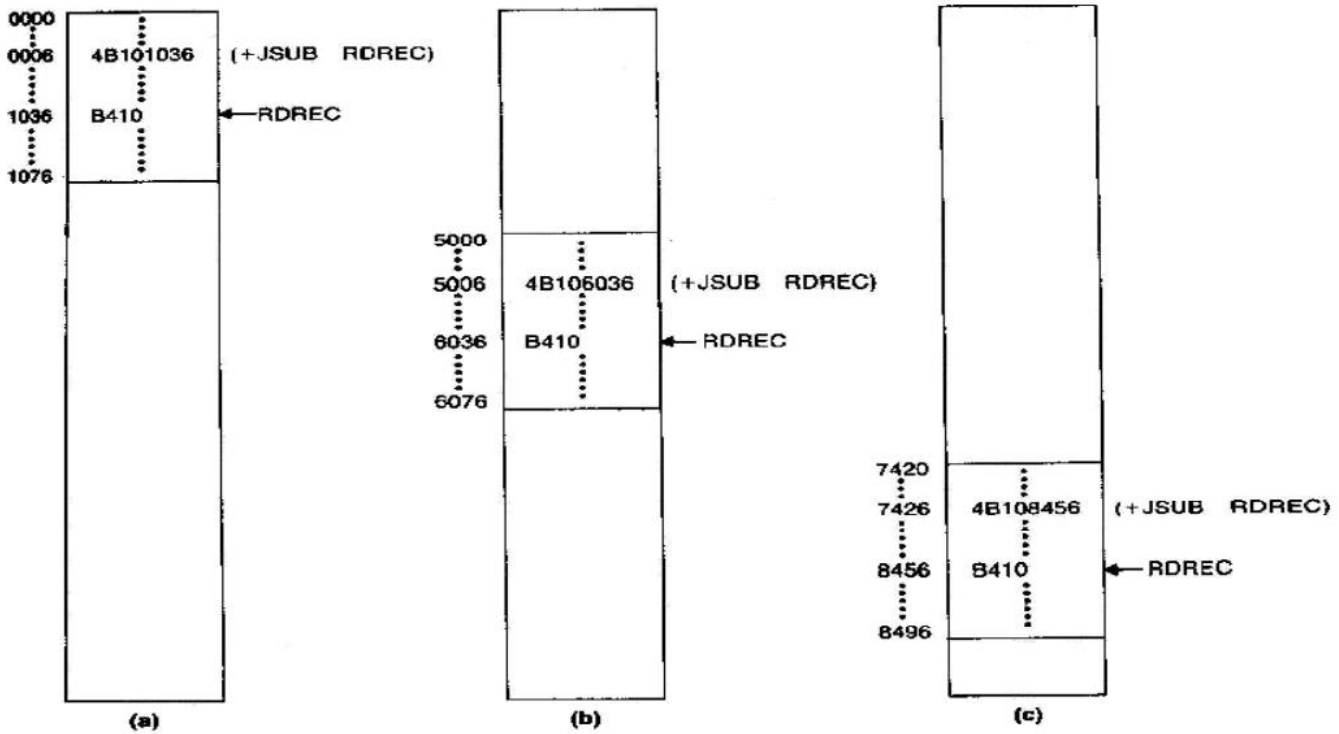


Figure 2.7 Examples of program relocation.

2.2.2 Program Relocation (Cont.)

- *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** must 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*
    - In SIC/XE, only found in extended format instructions

59

Instruction Format vs. Relocatable Loader

- In SIC/XE
  - Format 1, 2, 3
    - Not affect
  - Format 4
    - Should be modified
- In SIC
  - Format 3 with address field
    - Should be modified
    - SIC does not support PC-relative and base-relative addressing

Relocatable Program

- We use modification records that are added to the object files.

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 for Fig 2.6 (Fig 2.8)

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

Figure 2.8 Object program corresponding to Fig. 2.6.

Machine-Independent Assembler Features

- Literals
- Symbol-Defining Statements
- Expressions
- Program Blocks
- Control Sections and Program Linking

2.3.1 Literals

- Design idea
  - Let programmers to be able to write the value of a constant operand as a part of the instruction that uses it.
  - This avoids having to define the constant elsewhere in the program and make up a label for it.
  - Such an operand is called a ***literal*** because the value is stated "literally" in the instruction.
  - A literal is identified with the prefix =
- Examples
  - 45 001A ENDFILLDA =C'EOF' 032010
  - 215 1062 WLOOPTD =X'05' E32011

Original Program (Fig. 2.6)

```
 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
```

Using Literal (Fig. 2.9)

```
 5     COPY      START   0                 COPY FILE FROM INPUT TO OUTPUT
10     FIRST     STL     RETADR            SAVE RETURN ADDRESS
13               LDB     #LENGTH           ESTABLISH BASE REGISTER
14               BASE    LENGTH
15     CLOOP     +JSUB   RDREC             READ INPUT RECORD
20               LDA     LENGTH            TEST FOR EOF (LENGTH = 0)
25               COMP    #0
30               JEQ     ENDFIL            EXIT IF EOF FOUND
35               +JSUB   WRREC             WRITE OUTPUT RECORD
40               J       CLOOP             LOOP
45     ENDFIL    LDA     =C'EOF'           INSERT END OF FILE MARKER
50               STA     BUFFER
55               LDA     #3                SET LENGTH = 3
60               STA     LENGTH
65               +JSUB   WRREC             WRITE EOF
70               J       @RETADR           RETURN TO CALLER
93               LTORG
95     RETADR    RESW    1
100    LENGTH    RESW    1                 LENGTH OF RECORD
105    BUFFER    RESB    4096              4096-BYTE BUFFER AREA
106    BUFEND    EQU     *
107    MAXLEN    EQU     BUFEND-BUFFER     MAXIMUM RECORD LENGTH
```

Object Program Using Literal

```
 5     0000    COPY     START   0
10     0000    FIRST    STL     RETADR          17202D
13     0003             LDB     #LENGTH         69202D
14                      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     =C'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
93                      LTORG
       002D     *       =C'EOF'                 454F46
95     0030    RETADR   RESW    1
```

The same as before

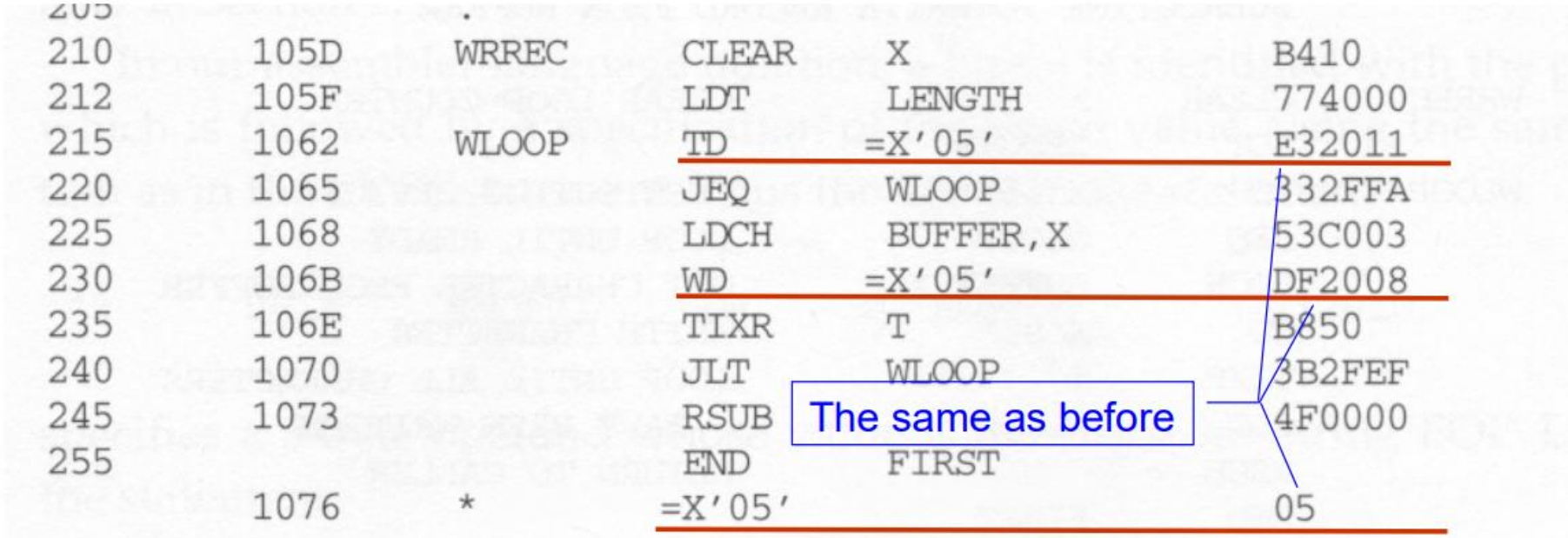
Original Program (Fig. 2.6)

```
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
```

Using Literal (Fig. 2.9)

```
195         .
200         .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205         .
210  WRREC  CLEAR   X               CLEAR LOOP COUNTER
212         LDT     LENGTH
215  WLOOP  TD      =X'05'          TEST OUTPUT DEVICE
220         JEQ     WLOOP           LOOP UNTIL READY
225         LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
230         WD      =X'05'          WRITE CHARACTER
235         TIXR    T               LOOP UNTIL ALL CHARACTERS
240         JLT     WLOOP             HAVE BEEN WRITTEN
245         RSUB                    RETURN TO CALLER
255         END     FIRST
```

Object Program Using Literal



Object Program Using Literal (Fig 2.9 & 2.10)

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
13	0003		LDB	#LENGTH	69202D
14			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	=C'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
93			LTORG		
	002D	*	=C'EOF'		454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
106	1036	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	
110		.			

Object Program Using Literal (Fig 2.9 & 2.10) (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    #MAXLEN         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
```

Object Program Using Literal (Fig 2.9 & 2.10) (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      =X'05'          E32011
220     1065                JEQ     WLOOP           332FFA
225     1068                LDCH    BUFFER,X        53C003
230     106B                WD      =X'05'          DF2008
235     106E                TIXR    T               B850
240     1070                JLT     WLOOP           3B2FEF
245     1073                RSUB                    4F0000
255                         END     FIRST
        1076    *       =X'05'                      05
```

Figure 2.10 Program from Fig. 2.9 with object code.

Literals vs. Immediate Operands

- Immediate Operands
  - The operand value is assembled as *part of the machine instruction*
  - e.g. 55 0020 LDA #3 010003
- Literals

Similar to define constant

  - The assembler generates the specified value as a constant *at some other memory location*
  - The effect of using a literal is exactly the same as if the programmer had *defined the constant* and used the *label* assigned to the constant as the instruction operand.
  - e.g. 45 001A ENDFIL LDA =C'EOF' 032010 (Fig. 2.9)
- Compare (Fig. 2.6)
  - e.g. 45 001A ENDFIL LDA EOF 032010
    80 002D EOF BYTE C'EOF' 454F46

74

Literal - Implementation

- *Literal pools*
  - All of the literal operands are gathered together into one or more *literal pools*
  - Normally, literal are placed at the end of the object program, i.e., following the END statement by the *assembler*
  - E.g., Fig. 2.10 (END statement)

```
255                     END         FIRST
        1076  *         =X'05'                  05
```

Literal – Implementation (Cont.)

- In some case, *programmer* would like to place literals into a pool at some other location in the object program
  - Using assembler directive **LTORG** (see Fig. 2.10)
  - Create a literal pool that contains all of the literal operands used since the previous LTORG
  - e.g., 45 001A ENDFIL LDA =C'EOF' 032010 (Fig.2.10)

    93 **LTORG**

    002D * =C'EOF' 454F46
  - Reason: keep the literal operand close to the instruction referencing it
    - Allow *PC-relative addressing* possible

Literal - Implementation (Cont.)

- Duplicate literals
  - e.g. 215 1062 WLOOP TD =X'05'
  - e.g. 230 106B WD =X'05'
  - The assemblers should recognize duplicate literals and store only one copy of the specified data value
    - Compare the character strings defining them
      - E.g., =X'05'
      - Easier to implement, but has potential problem (see next)

    Or compare the generated data value
      - E.g.the literals =C'EOF' and =X'454F46' would specify identical operand value.
      - Better, but will increase the complexity of the assembler

Same symbols, only one address is assigned

Literal - Implementation (Cont.)

- Be careful when using literal whose value depends on their *location* in the program
- For example, a literal may represent the *current value* of the *location counter*
  - Denoted by *
  - "LDB =*" may result in different object code when it appear *in different location*
  - *Cannot consider as duplicate literals*

Basic Data Structure for Assembler to Handle Literal Operands

- *Data Structure: literal table - LITTAB*
  - Content
    - Literal name
    - The operand value and length
    - Address assigned to the operand
  - Implementation
    - Organized as a hash table, using literal name or value as key.

How the Assembler Handles Literals?

- Pass 1
  - Build LITTAB with literal name, operand value and length, (leaving the address unassigned).
  - Handle duplicate literals. (Ignore duplicate literals)
  - When encounter LTORG statement or end of the program, assign an address to each literal not yet assigned an address
    - Remember to update the PC value to assign each literal's address
- Pass 2
  - Search LITTAB for each literal operand encountered
  - Generate data values in the object program exactly as if they are generated by BYTE or WORD statements
  - Generate modification record for literals that represent an *address* in the program (e.g. a location counter value)

SYMTAB & LITTAB

SYMTAB

Name	Value
COPY	0
FIRST	0
CLOOP	6
ENDFIL	1A
RETADR	30
LENGTH	33
BUFFER	36
BUFEND	1036
MAXLEN	1000
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WREC	105D
WLOOP	1062

LITTAB

Literal	Hex Value	Length	Address
C'EOF'	454F46	3	002D
X'05'	05	1	1076

2.3.2 Symbol-Defining Statements

- ***Labels*** on instructions or data areas
  - The value of such a label is the *address* assigned to the statement on which it appears
- Defining symbols
  - All programmer to define symbols and specify their values
  - Format: symbol **EQU** value
    - Value can be *constant* or *expression involving constants and previously defined symbols*
  - Example
    - MAXLEN EQU 4096
    - +LDT #MAXLEN

2.3.2 Symbol-Defining Statements (Cont.)

- Usage:
  - Make the source program easier to understand
- How assembler handles it?
  - In pass 1: when the assembler encounters the EQU statement, it enters the symbol into SYMTAB for later reference.
  - In pass 2: assemble the instruction with the *value* of the symbol
    - Follow the previous approach

Examples of Symbol-Defining Statements

- E.g. +LDT #4096 (Fig 2.5)
  - MAXLEN EQU 4096
  - +LDT #MAXLEN
- E.g. define mnemonic names for registers
  - A EQU 0
  - X EQU 1
  - L EQU 2
  - …
- E.g. define names that reflect the logical function of the registers in the program
  - BASE EQU R1
  - COUNT EQU R2
  - INDEX EQU R3

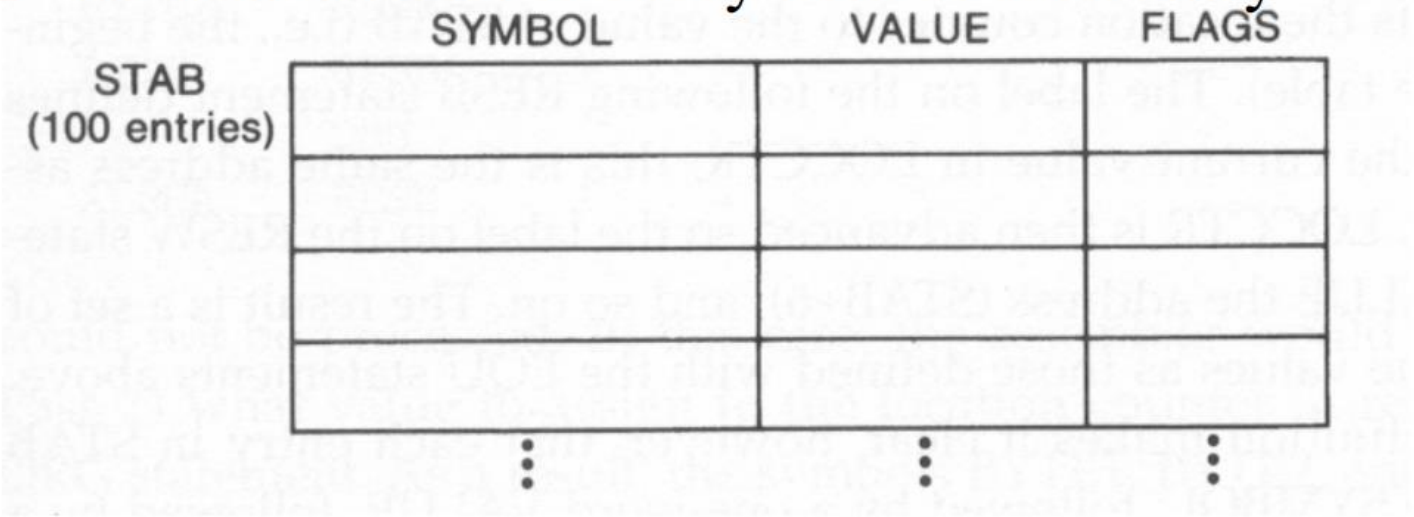
ORG

- ORG (origin)
  - Assembler directive: **ORG** value
    - Value can be *constant* or *expression involving constants and previously defined symbols*
  - Assembler resets the *location counter (LOCCTR)* to the specified value

LOCCTR control assignment of storage in the object program

Example of Using ORG

- Consider the following data structure
  - SYMBOL: 6 bytes
  - VALUE: 3 bytes (one word)
  - FLAGS: 2 bytes
- we want to refer to every field of each entry



ORG Example

- Using EQU statements

```
STAB      RESB   1100
SYMBOL    EQU    STAB
VALUE     EQU    STAB+6
FLAG      EQU    STAB+9
```

  - We can fetch the VALUE field by

    LDA VALUE,X

  - X = 0, 11, 22, … for each entry

Refer to entries in the table using indexed addressing

ORG Example (Cont.)

- Using ORG statements

```
STAB        RESB    1100
            ORG     STAB
SYMBOL      RESB    6
VALUE       RESW    1
FLAGS       RESB    2
            ORG     STAB+1100
```

Set the LOCCTR to STAB

Size of field more meaningful

Restore the LOCCTR to its previous value

  - This method of definition makes the structure more clear.
  - The last ORG is very important
    - Set program counter (LOCCTR) back to its previous value

Forward Reference

- All symbol-defining directives do ***not*** allow forward reference for 2-pass assembler
  - e.g., EQU, ORG…
  - All symbols used on the *right-hand side* of the statement must have been defined previously

E.g. (Cannot be assembled in 2-pass assm.)

```
ALPHA   EQU   BETA
BETA    EQU   DELTA
DELTA   RESW  1
```

Forward Reference (Cont.)

- E.g. (Cannot be assembled in 2-pass assm.)

```
            ORG         ALPHA
BYTE1       RESB        1
BYTE2       RESB        1
BYTE3       RESB        1
            ORG
ALPHA       RESB        1
```

2.3.3 Expressions

- Most assemblers allow the use of *expression* to replace symbol in the operand field.
  - Expression is evaluated by the assembler
  - Formed according to the rules using the operators +, -, *, /
    - Division is usually defined to produce an integer result
    - Individual terms can be
      - Constants
      - User-defined symbols
      - Special terms: e.g., * (= current value of location counter)

2.3.3 Expressions (Cont.)

- Review
  - Values in the object program are
    - *relative* to the beginning of the program or
    - *absolute* (independent of program location)
  - For example
    - Constants: absolute
    - Labels: relative

2.3.3 Expressions (Cont.)

- **Expressions can also be classified as *absolute expressions* or *relative expressions***
  - E.g. (Fig 2.9)

    107 MAXLEN EQU BUFEND-BUFFER
    - Both BUFEND and BUFFER are *relative terms*, representing addresses within the program
    - However the expression BUFEND-BUFFER represents an *absolute value: the difference between the two addresses*
  - When relative terms are paired with opposite signs
    - The dependency on the program starting address is canceled out
    - The result is an ***absolute value***

2.3.3 Expressions (Cont.)

- Absolute expressions
  - An expression that contains only absolute terms
  - An expression that contain relative terms but *in pairs* and the terms in each such pair have *opposite* signs
- Relative expressions
  - All of the relative terms *except one* can be paired and the remaining *unpaired relative terms* must have a *positive sign*
- ***No relative terms*** can enter into a multiplication or division operation no matter in absolute or relative expression

2.3.3 Expressions (Cont.)

- Errors: **(represent neither absolute values nor locations within the program)**
  - BUFEND+BUFFER // not opposite terms
  - 100-BUFFER // not in pair
  - 3*BUFFER // multiplication

2.3.3 Expressions (Cont.)

- Assemblers should determine the type of an expression
  - Keep track of the *types* of all symbols defined in the program in the symbol table.
  - Generate *Modification records* in the object program for relative values.

SYMTAB for Fig. 2.10

Symbol	Type	Value
RETADR	R	30
BUFFER	R	36
BUFEND	R	1036
MAXLEN	A	1000

2.3.4 Program Blocks

- Previously, main program, subroutines, and data area are treated as a unit and are assembled at the same time.
  - Although the source program logically contains subroutines, data area, etc, they were assembled into a single block of object code
  - To improve memory utilization, main program, subroutines, and data blocks may be allocated in separate areas.
- Two approaches to provide such a flexibility:
  - Program blocks
    - Segments of code that are rearranged within a single object program unit
  - Control sections
    - Segments of code that are translated into independent object program units

2.3.4 Program Blocks

- *Solution 1: Program blocks*
  - Refer to segments of code that are rearranged within a single object program unit
  - **Assembler directive: USE blockname**
    - Indicates which portions of the source program belong to which blocks.
  - Codes or data with same block name will allocate together
  - At the beginning, statements are assumed to be part of the unnamed (default) block
  - If no USE statements are included, the entire program belongs to this single block.

2.3.4 Program Blocks (Cont.)

- E.g: Figure 2.11
  - Three blocks
    - First: unnamed, i.e., default block
      - Line 5~ Line 70 + Line 123 ~ Line 180 + Line 208 ~ Line 245
    - Second: CDATA
      - Line 92 ~ Line 100 + Line 183 ~ Line 185 + Line 252 ~ Line 255
    - Third: CBLKS
      - Line 105 ~ Line 107
  - Each program block may actually contain *several separate segments* of the source program.
  - The assembler will (logically) rearrange these segments to gather together the pieces of each block.

Line	Source statement			
5	COPY	START	0	COPY FILE FROM INPUT TO OUTPUT
10	FIRST	STL	RETADR	SAVE RETURN ADDRESS
15	CLOOP	JSUB	RDREC	READ INPUT RECORD
20		LDA	LENGTH	TEST FOR EOF (LENGTH = 0)
25		COMP	#0	
30		JEQ	ENDFIL	EXIT IF EOF FOUND
35		JSUB	WRREC	WRITE OUTPUT RECORD
40		J	CLOOP	LOOP
45	ENDFIL	LDA	=C'EOF'	INSERT END OF FILE MARKER
50		STA	BUFFER	
55		LDA	#3	SET LENGTH = 3
60		STA	LENGTH	
65		JSUB	WRREC	WRITE EOF
70		J	@RETADR	RETURN TO CALLER
92		USE	CDATA	
95	RETADR	RESW	1	
100	LENGTH	RESW	1	LENGTH OF RECORD
103		USE	CBLKS	
105	BUFFER	RESB	4096	4096-BYTE BUFFER AREA
106	BUFEND	EQU	*	FIRST LOCATION AFTER BUFFER
107	MAXLEN	EQU	BUFEND-BUFFER	MAXIMUM RECORD LENGTH

```
120     .
123                 USE
125     RDREC       CLEAR     X             CLEAR LOOP COUNTER
130                 CLEAR     A             CLEAR A TO ZERO
132                 CLEAR     S             CLEAR S TO ZERO
133                +LDT       #MAXLEN
135     RLOOP       TD        INPUT         TEST INPUT DEVICE
140                 JEQ       RLOOP         LOOP UNTIL READY
145                 RD        INPUT         READ CHARACTER INTO REGISTER A
150                 COMPR     A,S           TEST FOR END OF RECORD (X'00')
155                 JEQ       EXIT          EXIT LOOP IF EOR
160                 STCH      BUFFER,X      STORE CHARACTER IN BUFFER
165                 TIXR      T             LOOP UNLESS MAX LENGTH
170                 JLT       RLOOP           HAS BEEN REACHED
175     EXIT        STX       LENGTH        SAVE RECORD LENGTH
180                 RSUB                    RETURN TO CALLER
183                 USE       CDATA
185     INPUT       BYTE      X'F1'         CODE FOR INPUT DEVICE
195     .
```

```
205     .
208             USE
210     WRREC   CLEAR   X           CLEAR LOOP COUNTER
212             LDT     LENGTH
215     WLOOP   TD      =X'05'      TEST OUTPUT DEVICE
220             JEQ     WLOOP       LOOP UNTIL READY
225             LDCH    BUFFER,X    GET CHARACTER FROM BUFFER
230             WD      =X'05'      WRITE CHARACTER
235             TIXR    T           LOOP UNTIL ALL CHARACTERS
240             JLT     WLOOP         HAVE BEEN WRITTEN
245             RSUB                RETURN TO CALLER
252             USE     CDATA
253             LTORG
255             END     FIRST
```

Figure 2.11 Example of a program with multiple program blocks.

Program with Multiple Program Blocks (Fig 2.11 & 2.12)

Line	Loc/Block		Source statement			Object code
5	0000	0	COPY	START	0	
10	0000	0	FIRST	STL	RETADR	172063
15	0003	0	CLOOP	JSUB	RDREC	4B2021
20	0006	0		LDA	LENGTH	032060
25	0009	0		COMP	#0	290000
30	000C	0		JEQ	ENDFIL	332006
35	000F	0		JSUB	WRREC	4B203B
40	0012	0		J	CLOOP	3F2FEE
45	0015	0	ENDFIL	LDA	=C'EOF'	032055
50	0018	0		STA	BUFFER	0F2056
55	001B	0		LDA	#3	010003
60	001E	0		STA	LENGTH	0F2048
65	0021	0		JSUB	WRREC	4B2029
70	0024	0		J	@RETADR	3E203F
92	0000	1		USE	CDATA	
95	0000	1	RETADR	RESW	1	
100	0003	1	LENGTH	RESW	1	
103	0000	2		USE	CBLKS	
105	0000	2	BUFFER	RESB	4096	
106	1000	2	BUFEND	EQU	*	
107	1000		MAXLEN	EQU	BUFEND-BUFFER	
110						

Program with Multiple Program Blocks (Fig 2.11 & 2.12) (Cont.)

```
110                             .
115                             .       SUBROUTINE TO READ RECORD INTO BUFFER
120                             .
123     0027    0               USE
125     0027    0       RDREC   CLEAR   X               B410
130     0029    0               CLEAR   A               B400
132     002B    0               CLEAR   S               B440
133     002D    0               +LDT    #MAXLEN         75101000
135     0031    0       RLOOP   TD      INPUT           E32038
140     0034    0               JEQ     RLOOP           332FFA
145     0037    0               RD      INPUT           DB2032
150     003A    0               COMPR   A,S             A004
155     003C    0               JEQ     EXIT            332008
160     003F    0               STCH    BUFFER,X        57A02F
165     0042    0               TIXR    T               B850
170     0044    0               JLT     RLOOP           3B2FEA
175     0047    0       EXIT    STX     LENGTH          13201F
180     004A    0               RSUB                    4F0000
183     0006    1               USE     CDATA
185     0006    1       INPUT   BYTE    X'F1'           F1
195
```

Program with Multiple Program Blocks (Fig 2.11 & 2.12)

```
195                     .
200                     .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205                     .
208     004D    0               USE
210     004D    0       WRREC   CLEAR   X               B410
212     004F    0               LDT     LENGTH          772017
215     0052    0       WLOOP   TD      =X'05'          E3201B
220     0055    0               JEQ     WLOOP           332FFA
225     0058    0               LDCH    BUFFER,X        53A016
230     005B    0               WD      =X'05'          DF2012
235     005E    0               TIXR    T               B850
240     0060    0               JLT     WLOOP           3B2FEF
245     0063    0               RSUB                    4F0000
252     0007    1               USE     CDATA
253                             LTORG
        0007    1       *       =C'EOF'                 454F46
        000A    1       *       =X'05'                  05
255                             END     FIRST
```

Figure 2.12 Program from Fig. 2.11 with object code.

Basic Data Structure for Assembler to Handle Program Blocks

- *Block name table*
  - Block name, block number, address, length

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

How the Assembler Handles Program Blocks?

- Pass 1
  - Maintaining ***separate location counter*** for each program block
  - Each label is assigned an address that is relative to *the start of the block* that contains it
  - When labels are entered into SYMTAB, the *block name* or *number* is stored along with the assigned relative addresses.
  - At the end of Pass 1, the latest value of the location counter for each block indicates the length of that block
  - The assembler can then assign to each block a starting address in the object program

1

How the Assembler Handles Program Blocks? (Cont.)

- Pass 2
  - The address of each symbol can be computed by adding the *assigned block starting address* and the relative address of the symbol to the start of its block
    - The assembler needs the address for each symbol *relative to the start of the object program*, not the start of an individual program block

Table for Program Blocks

- At the end of Pass 1 in Fig 2.11:

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Example of Address Calculation

- Each source line is given a *relative address assigned* and a *block number*
  - ***Loc/Block*** Column in Fig. 2.11
- For an *absolute symbol* (whose value is not relative to the start of any program block), there is no block number
  - E.g. 107 1000 MAXLEN EQU BUFEND-BUFFER
- Example: calculation of address in Pass 2
  - 20 0006 0 LDA LENGTH 032060

LENGTH = (block 1 starting address)+0003 = 0066+0003= 0069

LOCCTR = (block 0 starting address)+0009 = 0009

PC-relative: Displacement = 0069 - (LOCCTR) = 0069-0009=0060

2.3.4 Program Blocks (Cont.)

- Program blocks reduce addressing problem:
  - No needs for extended format instructions (lines 15, 35, 65)
    - The larger buffer is moved to the end of the object program
  - No needs for base relative addressing (line 13, 14)
    - The larger buffer is moved to the end of the object program
  - LTORG is used to make sure the literals are placed ahead of any large data areas (line 253)
    - Prevent literal definition from its usage too far

2.3.4 Program Blocks (Cont.)

- Object code
  - It is not necessary to physically rearrange the generated code in the object program to place the pieces of each program block together.
  - Loader will load the object code from each record at the *indicated addresses*.
- For example (Fig. 2.13)
  - The first two Text records are generated from line 5~70
  - When the USE statement is recognized
    - Assembler writes out the current Text record, even if there still room left in it
    - Begin a new Text record for the new program block

Object Program Corresponding to Fig. 2.11 (Fig. 2.13)

```
HCOPY  000000001071
T0000001E1720634B2021032060290000332006 4B203B3F2FEE032055 0F2056010003
T00001E090F20484B20293E203F
T0000271DB410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T000044093B2FEA13201F4F0000
T00006C01F1
T00004D19B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
T00006D04454F4605
E000000
```

Figure 2.13 Object program corresponding to Fig. 2.11.

```
begin
  block number = 0 LOCCTR[i] = 0 for all i
  read the first input line
  if OPCODE = 'START' then
  begin
    write line to intermediate file
    read next input line
  end {if START}
  while OPCODE ≠ 'END' do
  if OPCODE = 'USE'
  begin
    if there is no OPEREND name then
      set block name as default
    else block name as OPERAND name
    if there is no entry for block name then
      insert (block name, block number ++) in block table
    i = block number for block name
    if this is not a comment line then
      begin
      if there is a symbol in the LABEL field then
        begin
        search SYMTAB for LABEL
        if found then
          set error flag (duplicate symbol)
        else
          insert (LABEL, LOCCTR[i]) into SYMTAB
        end {if symbol}
```

```
      Search OPTAB for OPCODE
      if found then
        add 3 instruction length to LOCCTR[i]
      else if OPCODE = 'WORD' then
        add 3 to LOCCTR[i]
      else if OPCODE = 'RESW' then
        add 3 * #[OPERAND] to LOCCTR[i]
      else if OPCODE = 'RESB' then
        add #[OPERAND] to LOCCTR[i]
      else if OPCODE = 'BYTE' then
      begin
        find length of constant in bytes
        add length to LOCCTR[i]
      end {if byte}
else
```

Figure 2.12(b) Pass 1 of program blocks

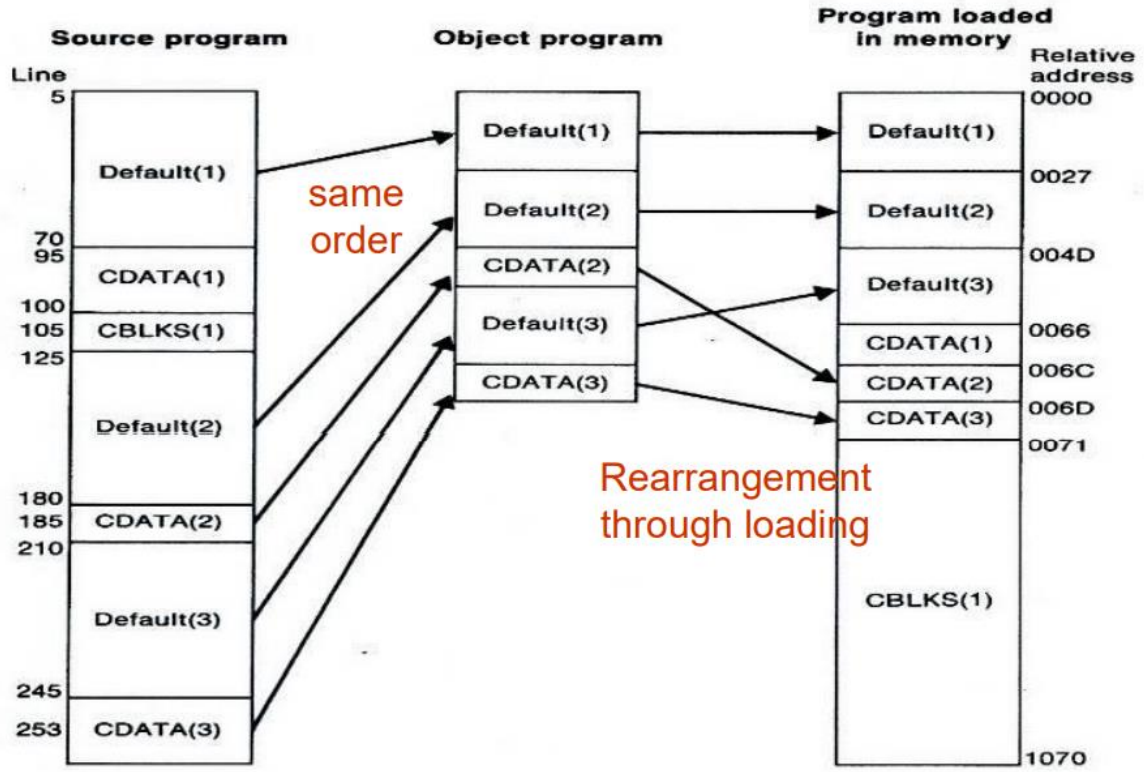
```
      Set error flag
      end {if not a comment}
    write line to intermediate file
    read next input line
  end {while not END}
write last line to intermediate file
save Length[i] as LOCCTR[i] for all i
Address[0] = starting address
Address[i] = address(i - 1) + Length(i - 1)
             [for i = 1 to max(block number)]
insert(address[i], Length[i]) in block table for all i
end {Pass 1}
```

Figure 2.12(b) (*cont'd*)

```
If OPCODE = 'USE' then
  set block number for block name with OPERAND field
  search SYMTAB for OPERAND
  store symbol value + address [block number] as operand address
end {Pass 2}
```

Figure 2.12(c) Pass 2 of program blocks.

Program blocks for the Assembly and Loading Processes (Fig. 2.14)



110

Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

2.3.5 Control Sections and Program Linking

- Control sections
  - A part of the program that maintains its *identity* after reassembly
    - Each control section can be loaded and relocated independently
    - Programmer can assemble, load, and manipulate each of these control sections separately
  - Often used for subroutines or other logical subdivisions of a program

2.3.5 Control Sections and Program Linking (Cont.)

- Instruction in one control section may need to refer to instructions or data located in another section
  - Called *external reference*
- However, assembler have no idea where any other control sections will be located at execution time
- The assembler has to generate information for such kind of references, called external references, that will allow the loader to perform the required linking.

Program Blocks v.s. Control Sections

- Program blocks
  - Refer to segments of code that are rearranged with *a single object program unit*
- Control sections
  - Refer to segments that are translated into *independent object program units*

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

First control section: COPY

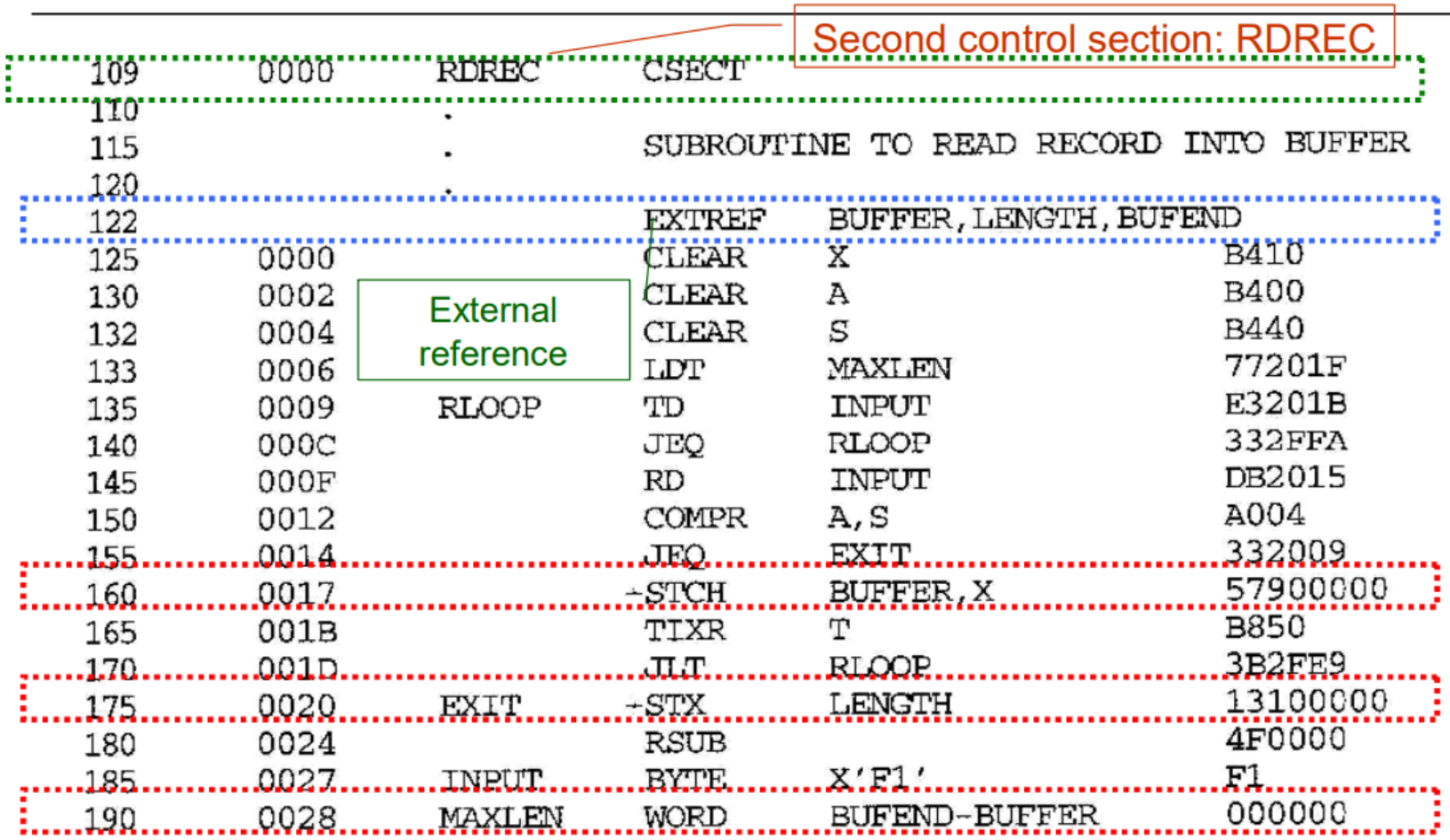
Implicitly defined as an external symbol

Define external symbols

External reference

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.

2.3.5 Control Sections and Program Linking (Cont.)

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

External Definition and References

- *External definition*
  - **Assembler directives: EXTDEF name [, name]**
  - EXTDEF names symbols, called ***external symbols***, that are defined in this control section and may be used by other sections
  - Control section names do not need to be named in an EXTDEF statement (e.g., COPY, RDREC, and WRREC)
    - They are automatically considered to be external symbols
- *External reference*
  - **Assembler directives: EXTREF name [,name]**
  - EXTREF names symbols that are used in this control section and are defined elsewhere

2.3.5 Control Sections and Program Linking (Cont.)

- Any instruction whose operand involves an external reference
  - Insert an address of zero and pass information to the loader
    - Cause the proper address to be inserted *at load time*
  - *Relative addressing* is not possible
    - The address of external symbol have no predictable relationship to anything in this control section
    - An *extended format instruction* must be used to provide enough room for the actual address to be inserted

Example of External Definition and References

- Example

15	0003	CLOOP	+JSUB	RDREC	4B100000
160	0017		+STCH	BUFFER,X	57900000
190	0028	MAXLEN	WORD	BUFEND-BUFFER	000000

How the Assembler Handles Control Sections?

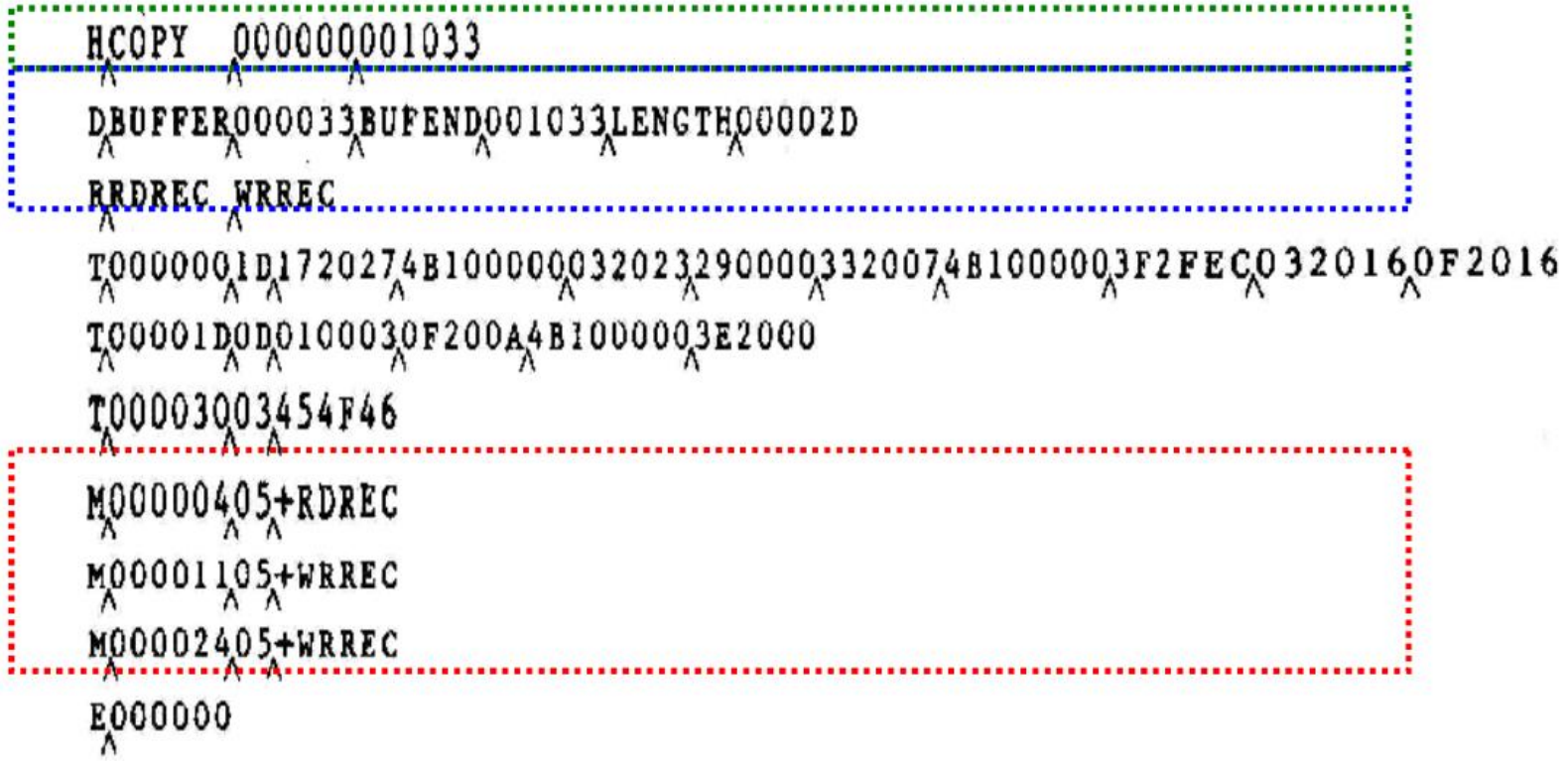
- **The assembler must include information in the object program that will cause the loader to insert proper values where they are required**
- *Define record:* gives information about external symbols named by EXTDEF
  - Col. 1 — D
  - Col. 2-7 — Name of external symbol defined in this section
  - Col. 8-13 — Relative address within this control section (hex)
  - Col.14-73 — Repeat information in Col. 2-13 for other external symbols
- *Refer record:* lists symbols used as external references, i.e., symbols named by EXTREF
  - Col. 1 — R
  - Col. 2-7 — Name of external symbol referred to in this section
  - Col. 8-73 — Name of other external reference symbols

121

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.
- Control section name is automatically an external symbol, it is available for use in Modification records.
- Example (Figure 2.17)
  - M000004^05^+RDREC
  - M000011^05^+WRREC
  - M000024^05^+WRREC
  - M000028^06^+BUFEND //Line 190 BUFEND-BUFFER
  - M000028^06^-BUFFER

Object Program Corresponding to Fig. 2.15 (Fig. 2.17)



Object Program Corresponding to Fig. 2.15 (Fig. 2.17) (Cont.)

```
HRDREC 000000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E
```

Object Program Corresponding to Fig. 2.15 (Fig. 2.17) (Cont.)

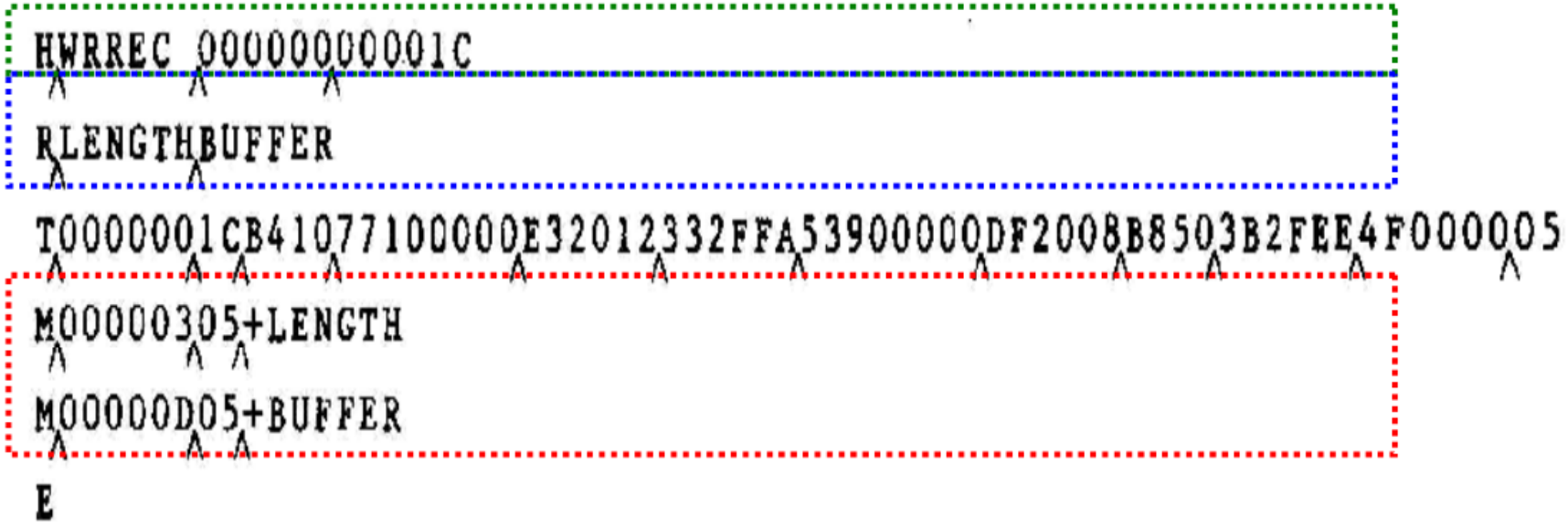


Figure 2.17 Object program corresponding to Fig. 2.15.

Program Linking & Relocation

- Note: the revised Modification record may still be used to perform *program relocation*.
  - E.g. (Fig. 2.8.)
    - M00000705
    - M00001405
    - M00002705 *are changed to*
    -
    - M00000705+COPY //add the beginning address of its section
    - M00001405+COPY
    - M00002705+COPY
- So the same mechanism can be used for program relocation and for program linking.

External References in Expression

- Earlier definitions
  - Required all of the relative terms be paired in an expression (an *absolute expression*), or that all except one be paired (a *relative expression*)
- New restriction
  - Both terms in each pair must be *relative within the same control section*
  - Ex: BUFEND-BUFFER: Legal
  - Ex: RDREC-COPY: illegal

External References in Expression (Cont.)

- However, when an expression involves external references, the assembler cannot determine whether or not the expression is legal.
- How to enforce this restriction
  - The assembler evaluates all of the terms it can, combines these to form an initial expression value, and generates Modification records.
  - The *loader* checks the expression for errors and finishes the evaluation.

2.4 Assembler Design Options

- One-pass assemblers
- Multi-pass assemblers

2.4.1 One-Pass Assemblers

- Goal: avoid a second pass over the source program
- Main problem
  - Forward references to *data items* or *labels on instructions*
- Solution
  - Data items: require all such areas be defined before they are referenced
  - Label on instructions: cannot be eliminated
    - E.g. the logic of the program often requires a forward jump
    - It is too inconvenient if forward jumps are not permitted

Two Types of One-Pass Assemblers:

- Load-and-go assembler
  - Produces object code directly in memory for immediate execution
- The other assembler
  - Produces usual kind of object code for later execution

Load-and-Go Assembler

- No object program is written out, no loader is needed
- Useful for program development and testing
  - Avoids the overhead of writing the object program out and reading it back in
- Both one-pass and two-pass assemblers can be designed as load-and-go
  - However, one-pass also avoids the overhead of an additional pass over the source program
- For a load-and-go assembler, the actual address must be known at assembly time.

Forward Reference Handling in One-pass Assembler

- When the assembler encounter an instruction operand that has not yet been defined:
  1. The assembler omits the translation of operand address
  2. Insert the symbol into SYMTAB, if not yet exist, and mark this symbol ***undefined***
  3. The address that refers to the undefined symbol is added to *a list of forward references* associated with the symbol table entry
  4. When the definition for a symbol is encountered
     1. The forward reference list for that symbol is scanned
     2. The proper address for the symbol is inserted into any instructions previous generated.

Handling Forward Reference in One-pass Assembler (Cont.)

- At the end of the program
  - Any SYMTAB entries that are still marked with * indicate *undefined symbols*
    - Be flagged by the assembler as errors
  - Search SYMTAB for the symbol named in the END statement and jump to this location to begin execution of the assembled program.

Sample Program for a One-Pass Assembler (Fig. 2.18)

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					

Sample Program for a One-Pass Assembler (Fig. 2.18) (Cont.)

```
110                 .
115                 .       SUBROUTINE TO READ RECORD INTO BUFFE
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                 .
```

Sample Program for a One-Pass Assembler (Fig. 2.18) (Cont.)

```
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
```

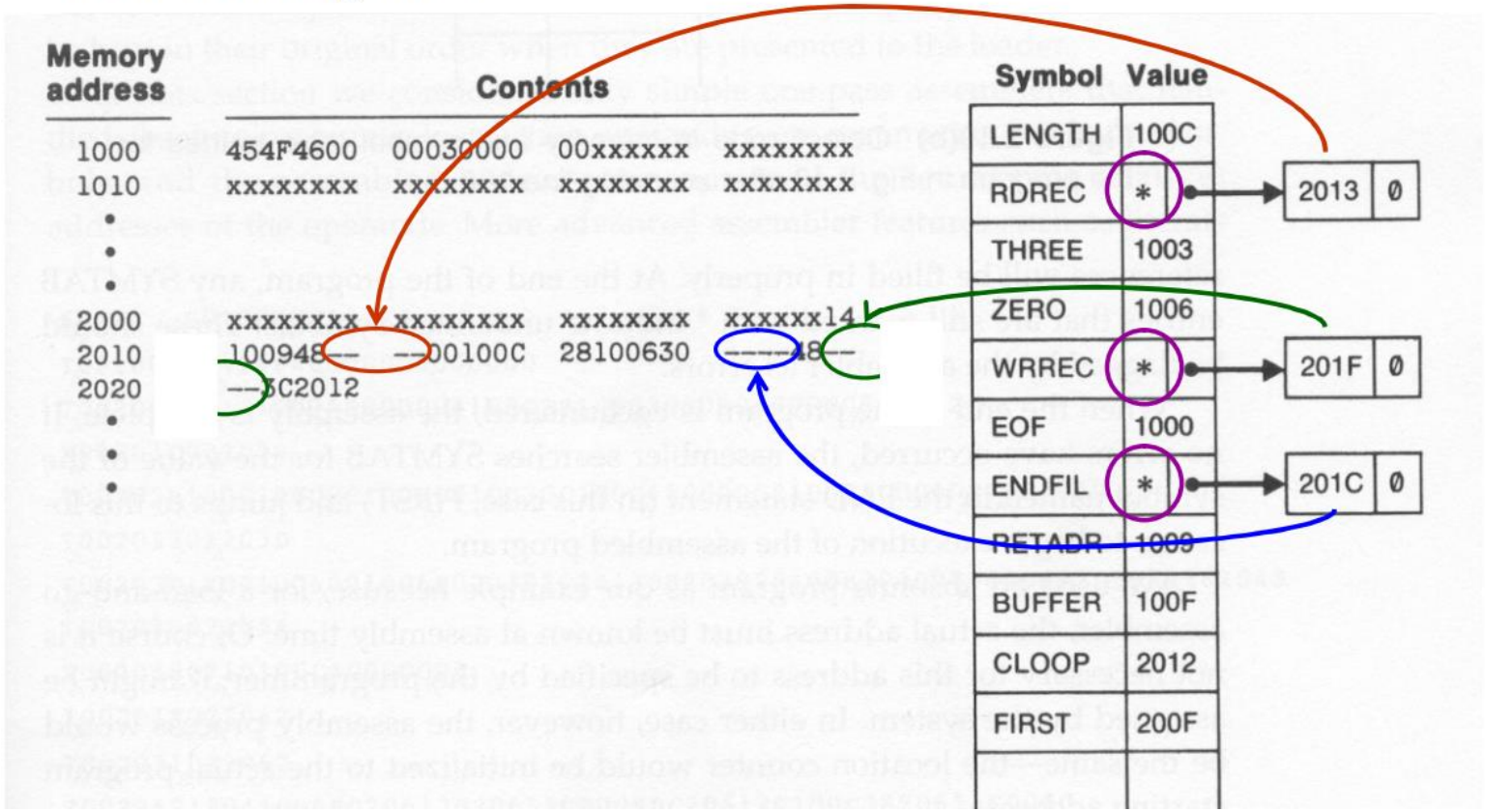
Figure 2.18 Sample program for a one-pass assembler.

Example

- Fig. 2.19 (a)
  - Show the object code in memory and symbol table entries after scanning line 40
  - Line 15: forward reference (RDREC)
    - Object code is marked ----
    - Value in symbol table is marked as * (undefined)
    - Insert *the address of operand* (2013) in a list associated with RDREC
  - Line 30 and Line 35: follow the same procedure

Object Code in Memory and SYMTAB

After scanning line 40

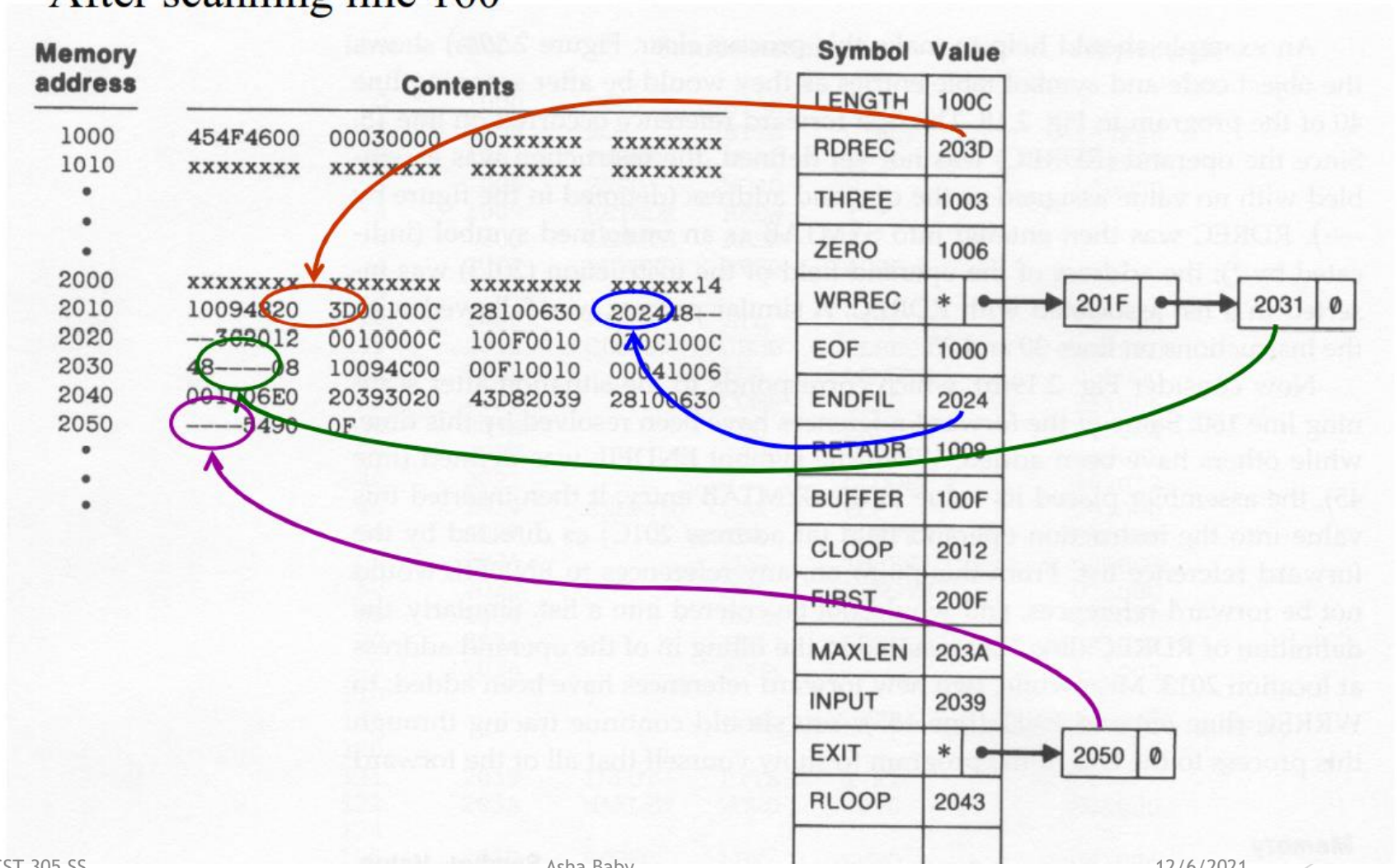


Example (Cont.)

- Fig. 2.19 (b)
  - Show the object code in memory and symbol table entries after scanning line 160
  - Line 45: ENDFIL was defined
    - Assembler place its value in the SYMTAB entry
    - Insert this value into the address (at 201C) as directed by the forward reference list
  - Line 125: RDREC was defined
    - Follow the same procedure
  - Line 65 and 155
    - Two new forward reference (WRREC and EXIT)

Object Code in Memory and SYMTAB

After scanning line 160



Object Code in Memory and SYMTAB Entries for Fig 2.18 (Fig. 2.19b)

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	10094820	3D00100C	28100630	202448
2020	--3C2012	0010000C	100F0010	030C100C
2030	48-----08	10094C00	00F10010	00041006
2040	00100650	20393020	43D82039	28100630
2050	----5490	0F		
•				
•				
•				

Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	* → 201F → 2031 ∅
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	* → 2050 ∅
RLOOP	2043

Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

One-Pass Assembler Producing Object Code

- Forward reference are entered into the symbol table's list as before
  - If the operand contains an undefined symbol, use 0 as the address and write the Text record to the object program.
- However, when definition of a symbol is encountered, the assembler must generate *another Text record* with the *correct operand address*.
- When the program is loaded, this address will be inserted into the instruction by *loader*.
- The object program records must be kept in their original order when they are presented to the loader

Example

- In Fig. 2.20
  - Second Text record contains the object code generated from lines 10 through 40
    - The operand addressed for the instruction on line 15, 30, 35 have been generated as 0000
  - When the definition of ENDFIL is encountered
    - Generate the third Text record
      - Specify the value 2024 (the address of ENDFIL) is to be loaded at location 201C (the operand field of JEQ in line 30)
      - Thus, the value 2024 will replace the 0000 previously loaded

Object Program from one-pass assembler for Fig 2.18 (Fig 2.20)

201C

```
H^COPY  ^001000^00107A
T^001000^09^454F46^000003^000000
T^00200F^15^141009^480000^00100C^281006^300000^480000^3C2012
T^00201C^02^2024
T^002024^19^001000^0C100F^001003^0C100C^480000^081009^4C0000^F1^001000
T^002013^02^203D
T^00203D^1E^041006^001006^E02039^302043^D82039^281006^300000^54900F^2C203A^382043
T^002050^02^205B
T^00205B^07^10100C^4C0000^05
T^00201F^02^2062
T^002031^02^2062
T^002062^18^041006^E02061^302065^50900F^DC2061^2C100C^382065^4C0000
E^00200F
```

Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.

One pass assembler algorithm

```
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialize LOCCTR as starting address
      read next input line
    end {if START}
  else
    initialize LOCCTR to 0
while OPCODE ≠ 'END' do
    begin
      if there is not a comment line then
        begin
          if there is a symbol in the LABEL field then
            begin
              search SYMTAB for LABEL
                if found then
              begin
                  if symbol value as null
                  set symbol value as LOCCTR and search
                    the linked list with the corresponding
                    operand
                  PTR addresses and generate operand
                    addresses as corresponding symbol
                    values
                  set symbol value as LOCCTR in symbol
                    table and delete the linked list
              end
```

```
    else
      insert (LABEL, LOCCTR) into SYMTAB
end
  search OPTAB for OPCODE
    if found then
      begin
        search SYMTAB for OPERAND address
    if found then
      if symbol value not equal to null then
        store symbol value as OPERAND address
      else
        insert at the end of the linked list
          with a node with address as LOCCTR
      else
        insert (symbol name, null)
```

Figure 2.19(c) Algorithm for One pass assembler.

```
    add 3 to LOCCTR
  end
  else if OPCODE = 'WORD' then
    add 3 to LOCCTR & convert comment to
      object code
  else if OPCODE = 'RESW' then
    add 3 #[OPERAND] to LOCCTR
  else if OPCODE = 'RESB' then
    add #[OPERAND] to LOCCTR
  else if OPCODE = 'BYTE' then
    begin
      find length of constant in bytes
      add length to LOCCTR
      convert constant to object code
    end
if object code will not fit into current
  text record then
  begin
    write text record to object program
    initialize new text record
  end
```

```
            end
         write listing line
         read next input line
      end
   write last Text record to object program
   write End record to object program
   write last listing line
end {Pass 1}
```

Figure 2.19(c) (*cont'd*)

2.4.2 Multi-Pass Assemblers

- Motivation: for a 2-pass assembler, any symbol used on the *right-hand side* should be defined previously.
  - No forward references since symbols' value can't be defined during the first pass
    - E.g.

```
APLHA   EQU   BETA
BETA    EQU   DELTA
DELTA   RESW  1
```

Not allowed !

Multi-Pass Assemblers (Cont.)

- Multi-pass assemblers
  - Eliminate the restriction on EQU and ORG
  - Make as many passes as are needed to process the definitions of symbols.
- Implementation
  - To facilitate symbol evaluation, in SYMTAB, each entry must indicates *which symbols are dependent on the values of it*
  - Each entry keeps a *linking list* to keep track of whose symbols' value depend on an this entry

Example of Multi-pass Assembler Operation (fig 2.21a)

```
HALFSZ   EQU   MAXLEN/2
MAXLEN   EQU   BUFEND-BUFFER
PREVBT   EQU   BUFFER-1
      .
      .
      .
BUFFER   RESB   4096
BUFEND   EQU    *
```

Example of Multi-Pass Assembler Operation (Fig 2.21b)

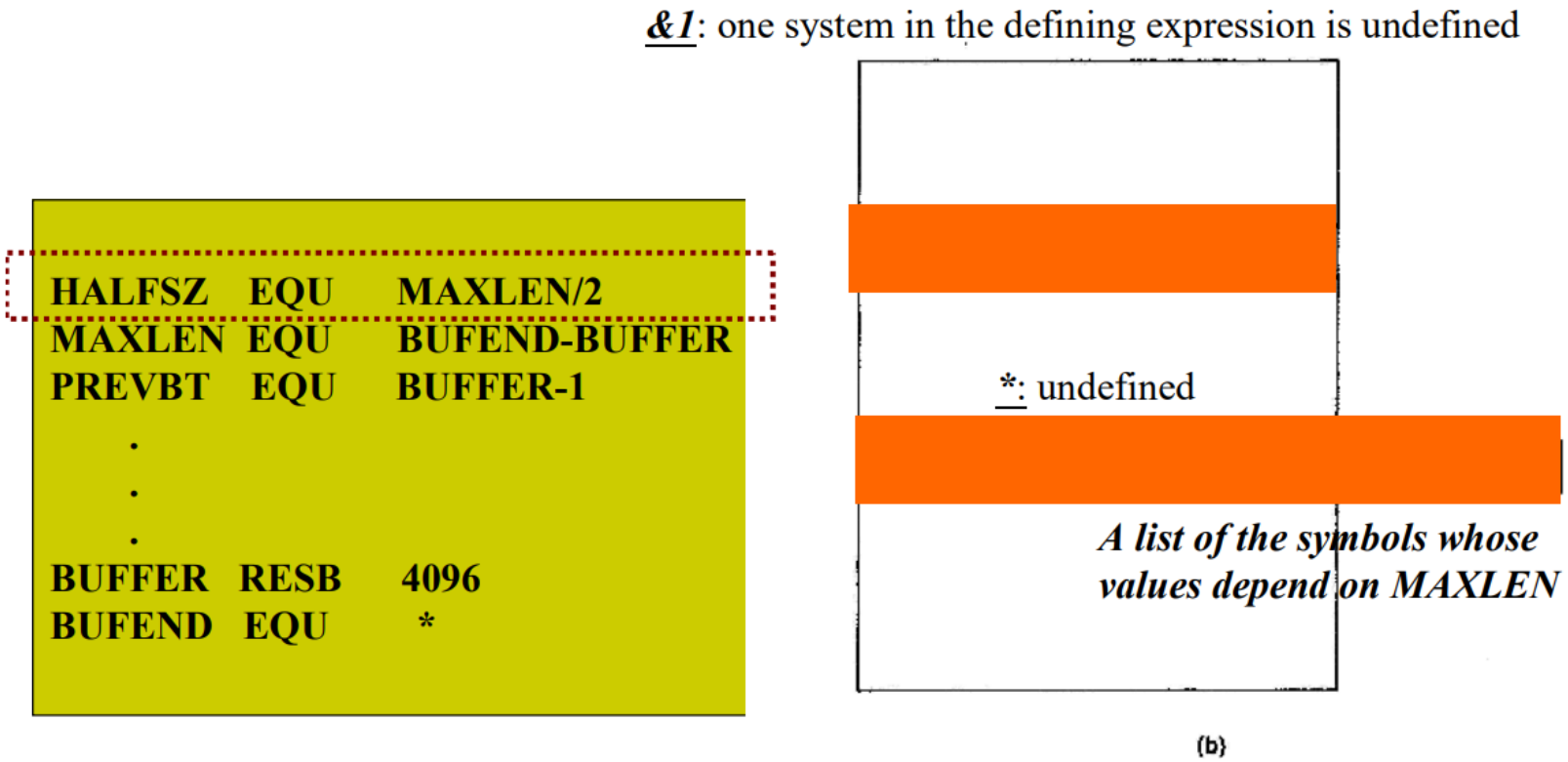
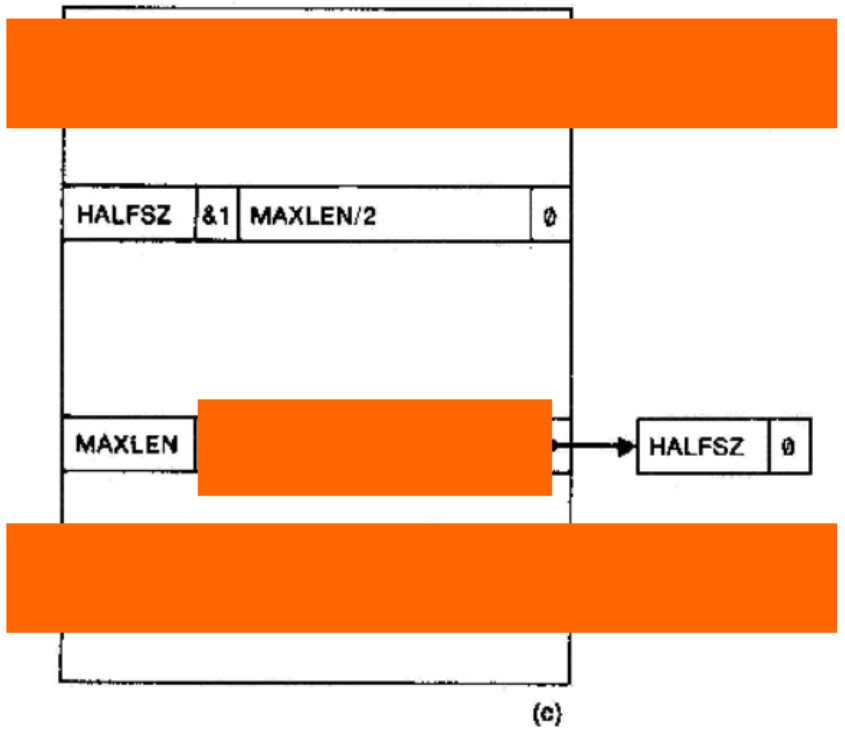


Figure 2.21 Example of multi-pass assembler operation.

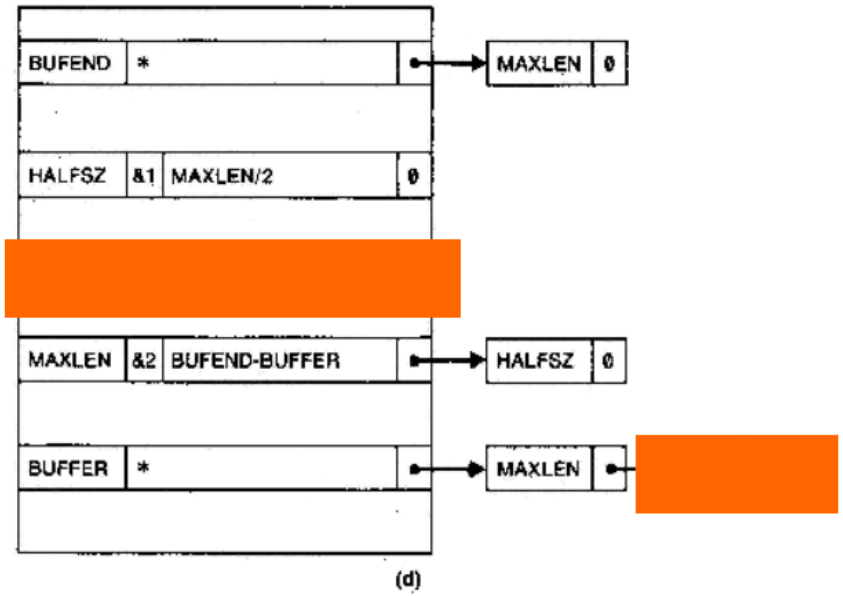
Example of Multi-Pass Assembler Operation (Fig 2.21c)

```
HALFSZ  EQU   MAXLEN/2
MAXLEN  EQU   BUFEND-BUFFER
PREVBT  EQU   BUFFER-1
        .
        .
        .
BUFFER  RESB  4096
BUFEND  EQU   *
```



Example of Multi-pass Assembler Operation (fig 2.21d)

```
HALFSZ   EQU     MAXLEN/2
MAXLEN   EQU     BUFEND-BUFFER
PREVBT   EQU     BUFFER-1
         .
         .
         .
BUFFER   RESB    4096
BUFEND   EQU     *
```

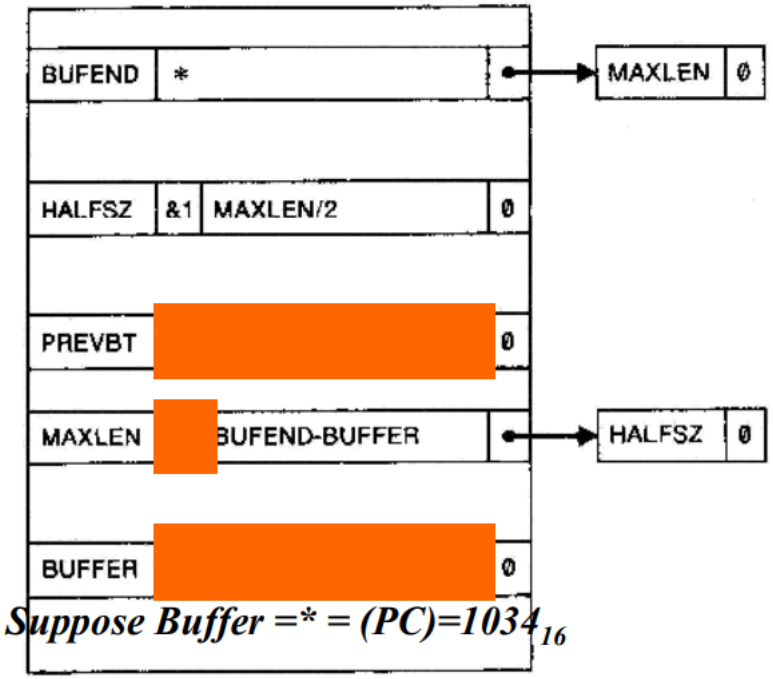


(d)

Figure 2.21 (*cont'd*)

Example of Multi-pass Assembler Operation (fig 2.21e)

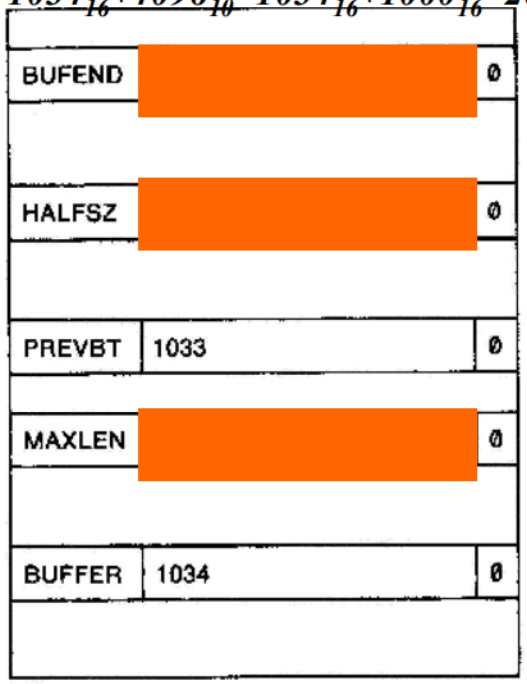
```
HALFSZ   EQU     MAXLEN/2
MAXLEN   EQU     BUFEND-BUFFER
PREVBT   EQU     BUFFER-1
         .
         .
         .
BUFFER   RESB    4096
BUFEND   EQU     *
```



Example of Multi-pass Assembler Operation (Fig 2.21f)

$BUFEND=*(PC)=1034_{16}+4096_{10}=1034_{16}+1000_{16}=2034_{16}$

```
HALFSZ   EQU    MAXLEN/2
MAXLEN   EQU    BUFEND-BUFFER
PREVBT   EQU    BUFFER-1
      .
      .
      .
BUFFER   RESB   4096
BUFEND   EQU    *
```



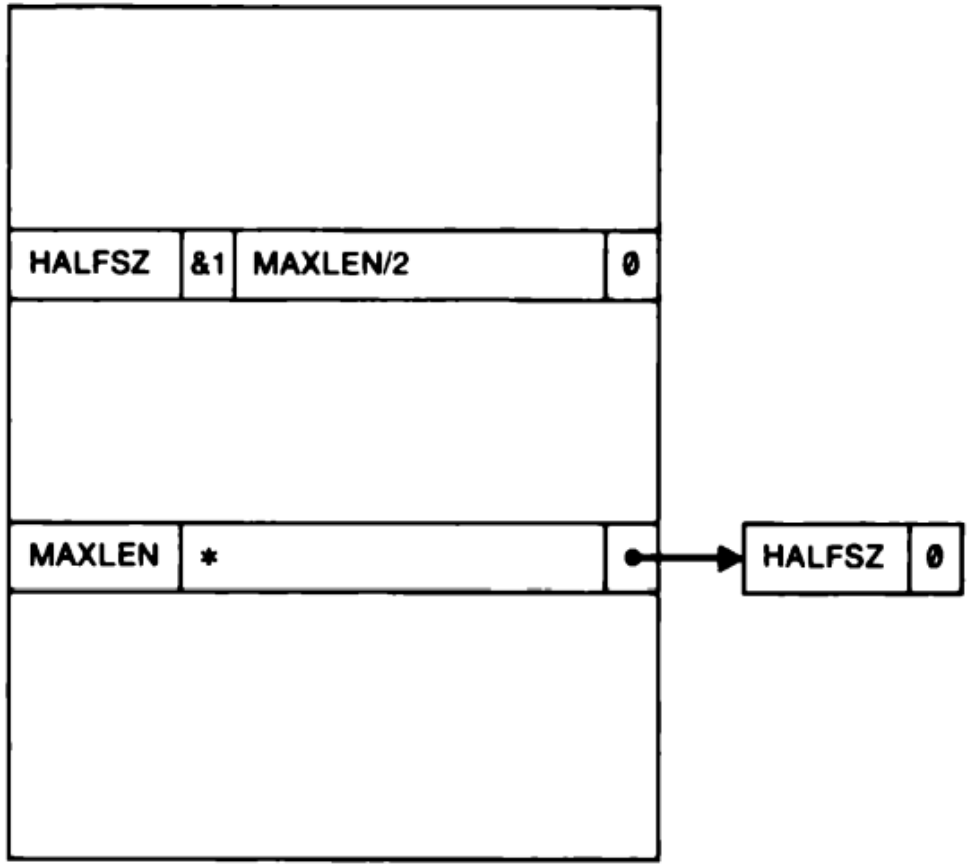
(f)

53

Figure 2.21 (*con'd*)

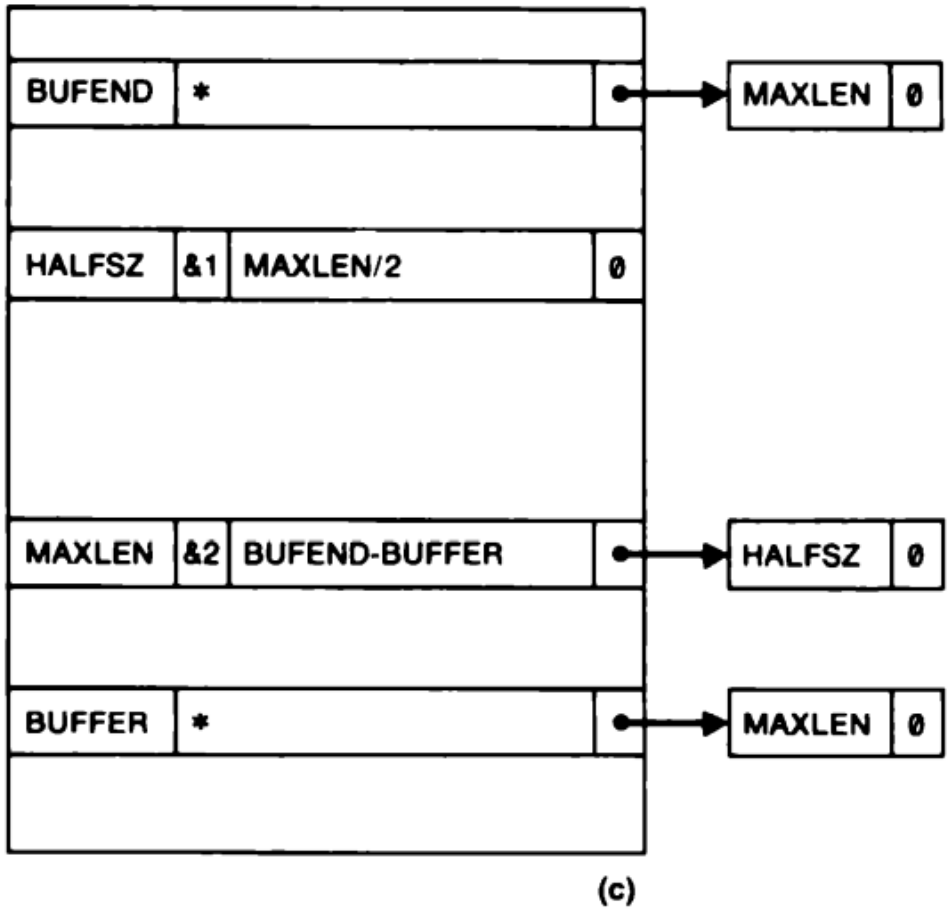
```
1       HALFSZ      EQU     MAXLEN/2
2       MAXLEN      EQU     BUFEND-BUFFER
3       PREVBT      EQU     BUFFER-1
                            .
                            .
                            .
4       BUFFER      RESB    4096
5       BUFEND      EQU     *
```

(a)



(b)

Figure 2.21 Example of multi-pass assembler operation.


BUFEND
*
MAXLEN
0
HALFSZ
&1
MAXLEN/2
0
MAXLEN
&2
BUFEND-BUFFER
HALFSZ
0
BUFFER
*
MAXLEN
0

(c)

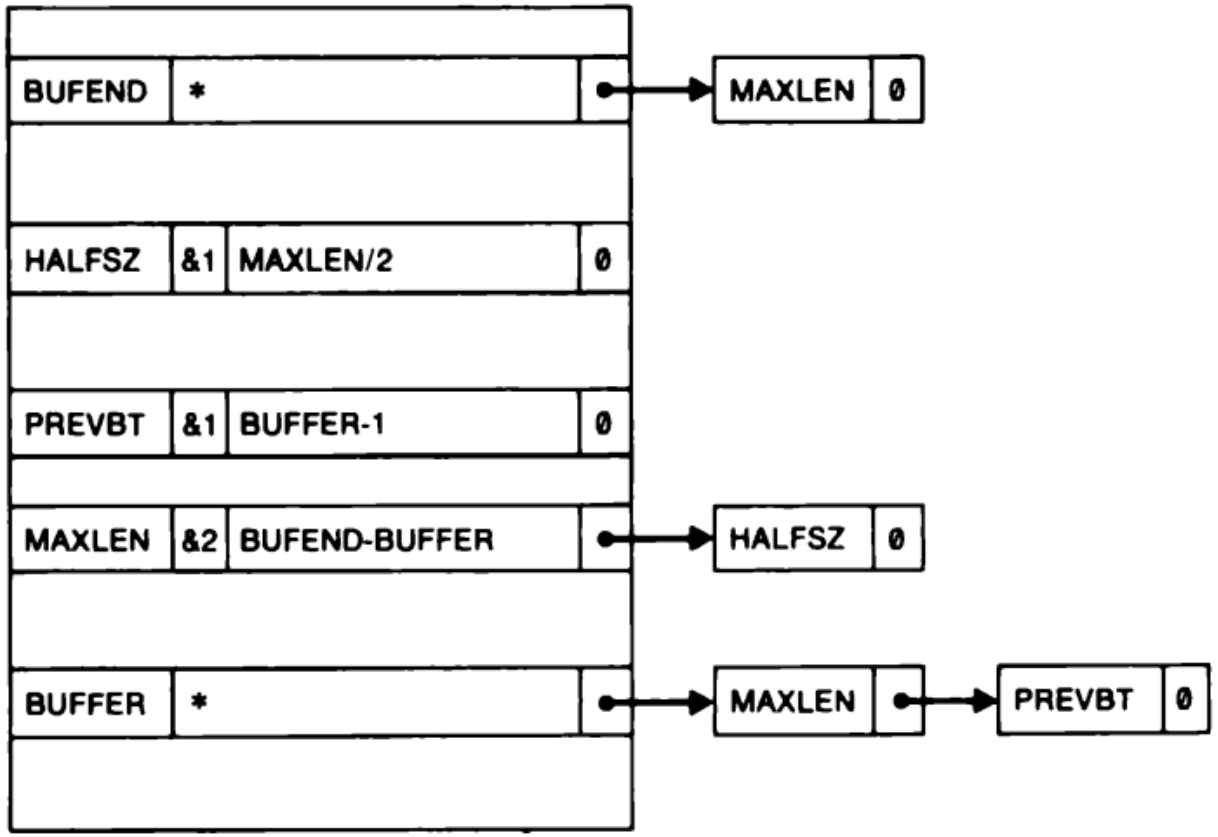
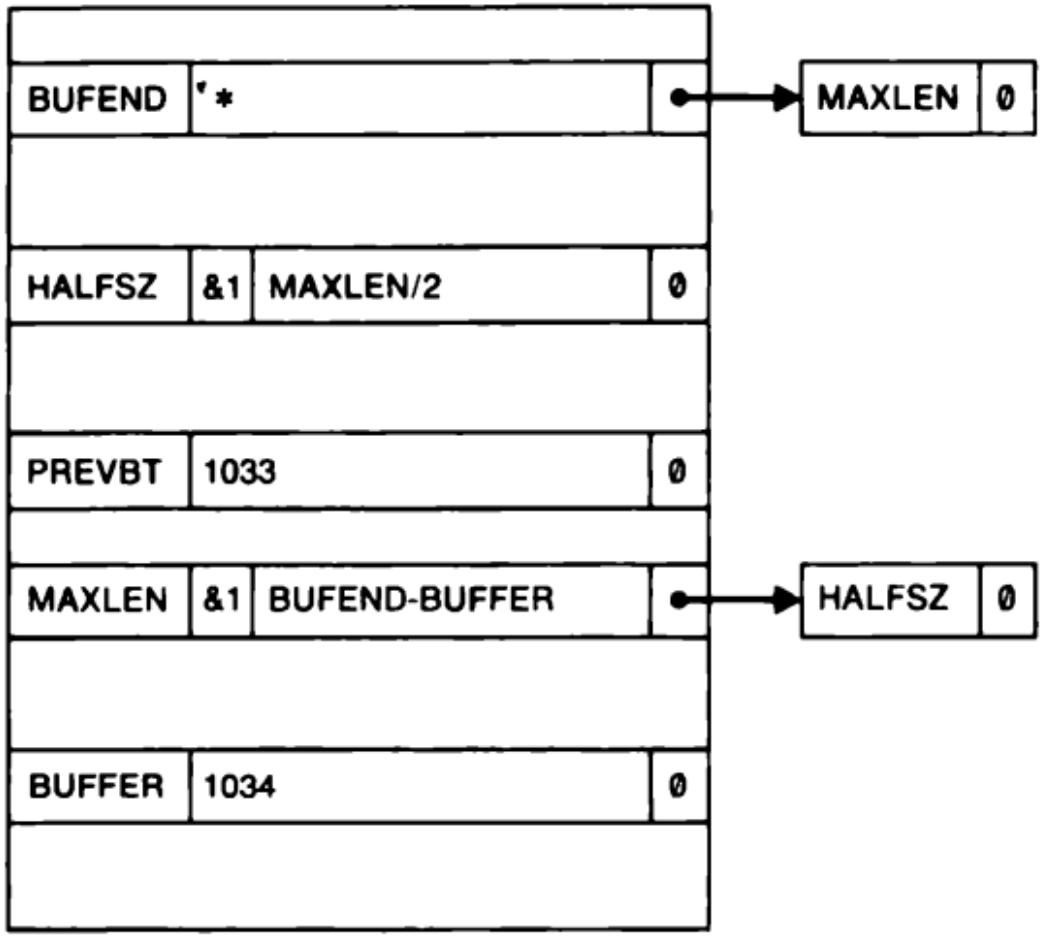
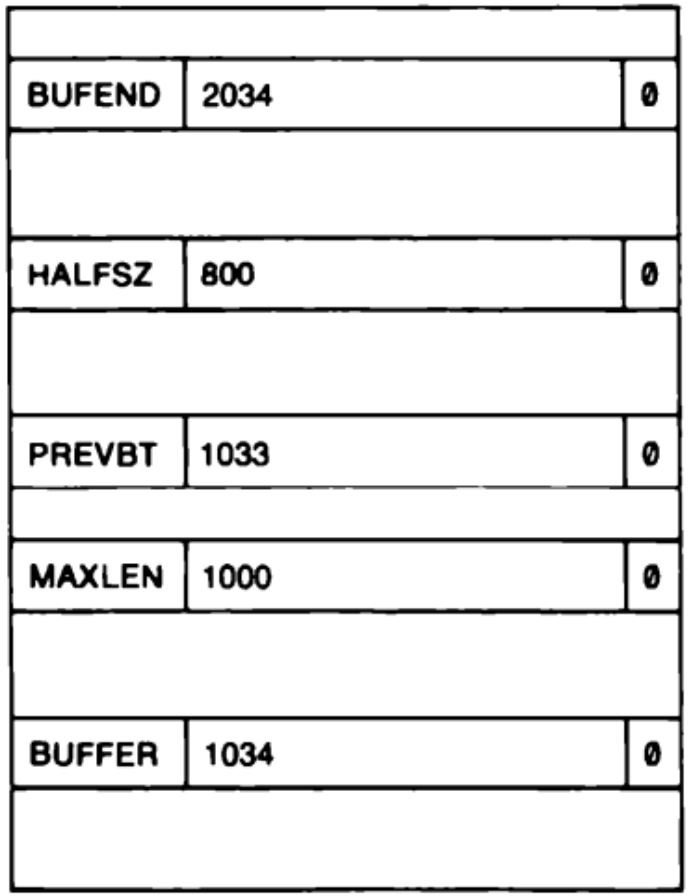


Figure 2.21 (*cont'd*)


BUFEND
'*
0
MAXLEN
0
HALFSZ
&1
MAXLEN/2
0
PREVBT
1033
0
MAXLEN
&1
BUFEND-BUFFER
0
HALFSZ
0
BUFFER
1034
0

(e)



(f)

Figure 2.21 (*con'd*)

2.5 Implementation Examples

- Microsoft MASM Assembler
- Sun Sparc Assembler
- IBM AIX Assembler

2.5.1 Microsoft MASM Assembler

- Microsoft MASM assembler for Pentium and other x86 systems
- Programmer of an x86 system views memory as a collection of segments

Microsoft MASM Assembler (Cont.)

- An MASM assembler language program is written as a collection of segments.
- Each segment is defined as belonging to a particular class: CODE, DATA, CONST, STACK
- Assembler directive: **SEGMENT**
  - Similar to program blocks in SIC
  - All parts of a segment are gathered together by assembler
- Segment registers are automatically set by the system loader when a program is loaded for execution: CS (code), SS (stack), DS (data), ES, FS, GS
- Assembler directive: **ASSUME**
  - By default, assembler assumes all references to data segments use register DS
  - We can change by the assembler directive ASSUME
  - e.g. ASSUME ES:DATASEG2
    - Tell the assembler that register ES indicate the segment DATASEG2
    - Thus, any reference to labels are defined in DATASEG2 will be assembled using register ES
  - Similar to **BASE** directive in SIC/XE
    - BASE tell a SIC/XE assembler the contents of register B
    - ASSUME tell MASM the contents of a segment register

Microsoft MASM Assembler (Cont.)

- Jump instructions are assembled in 2 different ways:
  - Near jump: jump to a target in the same code segment
    - 2- or 3-byte instruction
  - Far jump: jump to a target in a different code segment
    - 5-byte instruction
- Problem: Jump with forward reference
  - By default, MASM assumes that a forward jump is a near jump
  - If it is a far jump, ***programmer*** must tell the assembler
    - E.g. JMP ***FAR PTR*** TARGET
- In x86, the ***length*** of an assembled instruction depends on *the operands that are used*.
  - Operands maybe registers, memory locations, immediate values (1~4 bytes)
  - Thus, Pass 1 in MASM is much complex that in SIC assembler

Microsoft MASM Assembler (Cont.)

- External references between separately assembled modules must be handled by the *linker*
  - MASM directive: **PUBLIC**, **EXTRN**
  - Similar to **EXTDEF**, **EXTREF** in SIC/XE
- The object program from MASM may be in several different formats to allow easy and efficient execution of the program in a variety of operating environments.

University questions..

1. With suitable example, explain the concept of Program Relocation.
2. List out the basic functions of Assemblers with proper examples.
3. What is a Literal? How is a literal handled by an assembler?
4. Write notes on Multi pass assemblers.
5. With example, write notes on Program Blocks.
6. What is a forward reference? How are forward references handled by a single pass assembler?
7. Explain how external references are handled by an assembler.
8. With the aid of an algorithm explain the Second pass of a Two Pass Assembler.

9. What are control sections? What is the advantage of using them?
10. What are the uses of assembler directives EXTDEF and EXTREF?
11. Distinguish between Program Blocks and Control Section
12. How the assembler handles multiple Program blocks?
13. Explain the working of any one type of One pass Assembler
14. What is meant by forward reference? How it is resolved by two pass assembler?
15. Write down the format of Modification record. Describe each field with the help of an example.

16. With the aid of an algorithm explain the Second pass of a Two Pass Assembler.
17. What are control sections? What is the advantage of using them?
18. Given an idle computer with no programs in memory, how do we get things started?
19. What are the uses of assembler directives EXTDEF and EXTREF?
20. Give the algorithm for pass 1 of a two pass SIC assembler.
21. Describe the format of object program generated by the two-pass SIC assembler algorithm.
22. Explain with suitable examples, how the different instruction formats and addressing modes of SIC/XE are handled during assembling.

26. Explain the format and purpose of Define and Refer records in the object program.
27. Write short notes on MASM assembler.
28. Explain the concept of single pass assembler with a suitable example.
29. How are control sections different from program blocks? Explain, with proper examples, the purpose of EXTREF and EXTDEF assembler directives.
30. Explain, with examples, the working of a multi pass assembler.