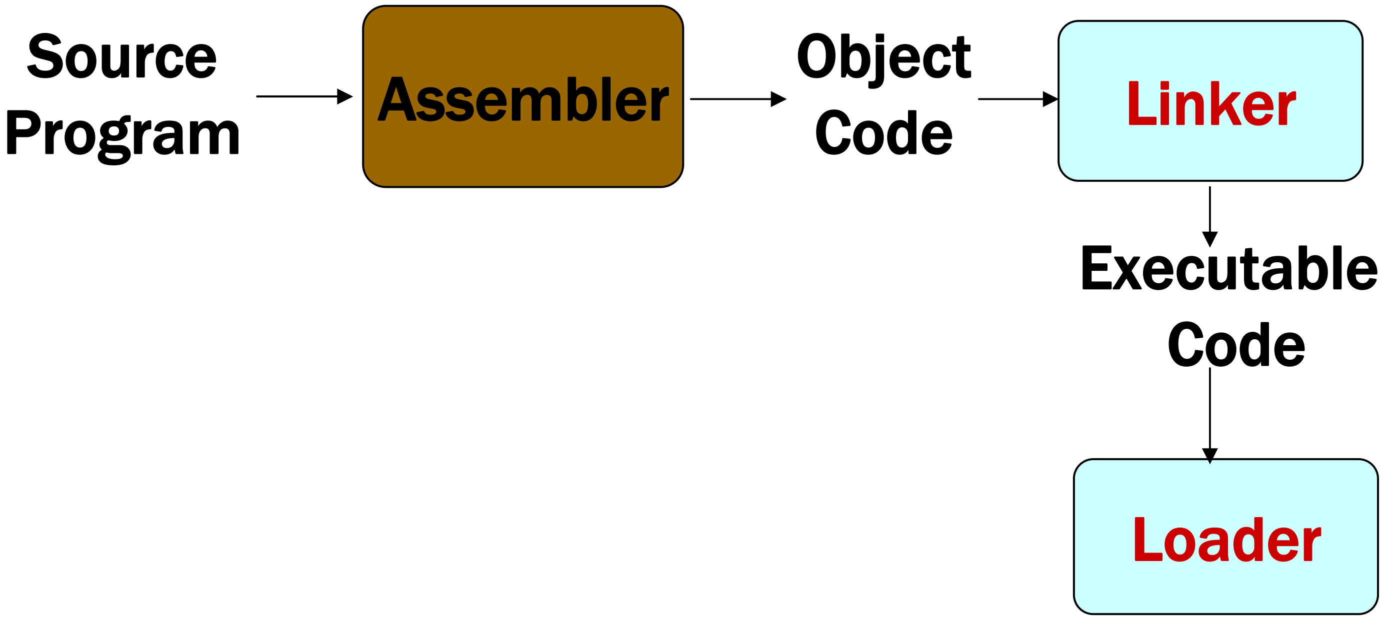
Chapter 2
Assemblers

http://www.intel.com/multi-core/demos.htm



Outline

- 2.1 Basic Assembler Functions
  - A simple SIC assembler
  - Assembler tables and logic
- 2.2 Machine-Dependent Assembler Features
  - Instruction formats and addressing modes
  - Program relocation
- 2.3 Machine-Independent Assembler Features
- 2.4 Assembler Design Options
  - Two-pass
  - One-pass
  - Multi-pass

2.1 Basic Assembler Functions

- Figure 2.1 shows an assembler language program for SIC.
  - The line numbers are for reference only.
  - Indexing addressing is indicated by adding the modifier ",X"
  - Lines beginning with "." contain comments only.
  - 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

Line Source statement

```
COPY     START   1000      COPY FILE FROM INPUT TO OUTPUT
FIRST    STL     RETADR    SAVE RETURN ADDRESS
CLOOP    JSUB    RDREC     READ INPUT RECORD
         LDA     LENGTH    TEST FOR EOF (LENGTH = 0)
         COMP    ZERO
         JEQ     ENDFIL    EXIT IF EOF FOUND
         JSUB    WRREC     WRITE OUTPUT RECORD
         J       CLOOP     LOOP
ENDFIL   LDA     EOF       INSERT END OF FILE MARKER
         STA     BUFFER
         LDA     THREE     SET LENGTH = 3
         STA     LENGTH
         JSUB    WRREC     WRITE EOF
         LDL     RETADR    GET RETURN ADDRESS
         RSUB              RETURN TO CALLER
EOF      BYTE    C'EOF'
THREE    WORD    3
ZERO     WORD    0
RETADR   RESW    1
LENGTH   RESW    1         LENGTH OF RECORD
BUFFER   RESB    4096      4096-BYTE BUFFER AREA
```

```
.
.           SUBROUTINE TO READ RECORD INTO BUFFER
.
RDREC       LDX       ZERO              CLEAR LOOP COUNTER
            LDA       ZERO              CLEAR A TO ZERO
RLOOP       TD        INPUT             TEST INPUT DEVICE
            JEQ       RLOOP             LOOP UNTIL READY
            RD        INPUT             READ CHARACTER INTO REGISTER A
            COMP      ZERO              TEST FOR END OF RECORD (X'00')
            JEQ       EXIT              EXIT LOOP IF EOR
            STCH      BUFFER,X          STORE CHARACTER IN BUFFER
            TIX       MAXLEN            LOOP UNLESS MAX LENGTH
            JLT       RLOOP               HAS BEEN REACHED
EXIT        STX       LENGTH            SAVE RECORD LENGTH
            RSUB                        RETURN TO CALLER
INPUT       BYTE      X'F1'             CODE FOR INPUT DEVICE
MAXLEN      WORD      4096
```

```
        .          SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
        WRREC      LDX       ZERO          CLEAR LOOP COUNTER
        WLOOP      TD        OUTPUT        TEST OUTPUT DEVICE
                   JEQ       WLOOP         LOOP UNTIL READY
                   LDCH      BUFFER,X      GET CHARACTER FROM BUFFER
                   WD        OUTPUT        WRITE CHARACTER
                   TIX       LENGTH        LOOP UNTIL ALL CHARACTERS
                   JLT       WLOOP           HAVE BEEN WRITTEN
                   RSUB                    RETURN TO CALLER
        OUTPUT     BYTE      X'05'         CODE FOR OUTPUT DEVICE
                   END       FIRST
```

Figure 2.1 Example of a SIC assembler language program.

2.1 Basic Assembler Functions

- Assembler *directives* (pseudo-instructions)
  - START, END, BYTE, WORD, RESB, RESW.
  - These statements are not translated into machine instructions.
  - Instead, they provide instructions to the assembler itself.

2.1 Basic Assembler Functions

- 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 (00_{16})
  - Buffer length is 4096 Bytes
  - The end of the file is indicated by a zero-length record
  - When the end of file is detected, the program writes EOF on the output device and terminates by RSUB.
- Subroutines (JSUB, RSUB)
  - RDREC, WRREC
  - Save link (L) register first before nested jump

2.1.1 A simple SIC Assembler

- Figure 2.2 shows the generated object code for each statement.
  - Loc gives the machine address in Hex.
  - Assume the program starting at address 1000.
- Translation functions
  - Translate STL to 14.
  - Translate RETADR to 1033.
  - Build the machine instructions in the proper format (,X).
  - Translate EOF to 454F46.
  - Write the object program and assembly listing.

Line	Loc	Source statement			Object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454F46
85	102D	THREE	WORD	3	000003
90	1030	ZERO	WORD	0	000000
95	1033	RETADR	RESW	1	
100	1036	LENGTH	RESW	1	
105	1039	BUFFER	RESB	4096	

Line	Loc	Source statement			Object code
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	2039	RDREC	LDX	ZERO	041030
130	203C		LDA	ZERO	001030
135	203F	RLOOP	TD	INPUT	E0205D
140	2042		JEQ	RLOOP	30203F
145	2045		RD	INPUT	D8205D
150	2048		COMP	ZERO	281030
155	204B		JEQ	EXIT	302057
160	204E		STCH	BUFFER,X	549039
165	2051		TIX	MAXLEN	2C205E
170	2054		JLT	RLOOP	38203F
175	2057	EXIT	STX	LENGTH	101036
180	205A		RSUB		4C0000
185	205D	INPUT	BYTE	X'F1'	F1
190	205E	MAXLEN	WORD	4096	001000
195		.			

Loc	Source statement			Object code
	.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
	.			
2061	WRREC	LDX	ZERO	041030
2064	WLOOP	TD	OUTPUT	E02079
2067		JEQ	WLOOP	302064
206A		LDCH	BUFFER,X	509039
206D		WD	OUTPUT	DC2079
2070		TIX	LENGTH	2C1036
2073		JLT	WLOOP	382064
2076		RSUB		4C0000
2079	OUTPUT	BYTE	X'05'	05
		END	FIRST	

Figure 2.2 Program from Fig. 2.1 with object code.

2.1.1 A simple SIC Assembler

- A forward reference
  - 10 1000 FIRST STL RETADR 141033
  - A reference to a label (RETADR) that is defined later in the program
  - Most assemblers make two passes over the source program
- Most assemblers make two passes over source program.
  - Pass 1 scans the source for label definitions and assigns address (Loc).
  - Pass 2 performs most of the actual translation.

2.1.1 A simple SIC Assembler

- Example of Instruction Assemble
  - Forward reference
  - STCH BUFFER, X

549039

8	1	15
opcode	x	address
		m

$(54)_{16}$ $1 \; (001)_{2}$ $(039)_{16}$

2.1.1 A simple SIC Assembler

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

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

2.1.1 A simple SIC Assembler

- The object program (OP) will be loaded into memory for execution.
- Three types of records
  - Header: program name, starting address, length.
  - Text: starting address, length, object code.
  - End: address of first executable instruction.

Header record:

Col. 1	H
Col. 2–7	Program name
Col. 8–13	Starting address of object program (hexadecimal)
Col. 14–19	Length of object program in bytes (hexadecimal)

2.1.1 A simple SIC Assembler

Text record:

Col. 1	T
Col. 2–7	Starting address for object code in this record(hexadecimal)
Col. 8–9	Length of object code in this record in bytes (hexadecimal)
Col. 10–69	Object code, represented in hexadecimal (2 columns per byte of object code)

End record:

Col. 1	E
Col. 2–7	Address of first executable instruction in object program (hexadecimal)

2.1.1 A simple SIC Assembler

- The symbol ^ is used to separate fields.
  - Figure 2.3

    1E(H)=30(D)=16(D)+14(D)

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

Figure 2.3 Object program corresponding to Fig. 2.2.

Object code

```
141033
482039
001036
281030
301015
482061
3C1003
00102A
0C1039
00102D
0C1036
482061
081033
4C0000
454F46
000003
000000
```

2.1.1 A simple SIC Assembler

- Assembler's Functions
  - Convert mnemonic operation codes to their machine language equivalents
    - STL to 14
  - Convert symbolic operands (referred label) to their equivalent machine addresses
    - RETADR to 1033
  - Build the machine instructions in the proper format
  - Convert the data constants to internal machine representations
  - Write the object program and the assembly listing

2.1.1 A simple SIC Assembler

- The functions of the two passes assembler.
- Pass 1 (define symbol)
  - Assign addresses to all statements (generate LOC).
  - Check the correctness of Instruction (check with OP table).
  - Save the values (address) assigned to all labels into SYMBOL table for Pass 2.
  - Perform some processing of assembler directives.
- Pass 2
  - Assemble instructions (op code from OP table, address from SYMBOL table).
  - Generate data values defined by BYTE, WORD.
  - Perform processing of assembler directives not done during Pass 1.
  - Write the OP (Fig. 2.3) and the assembly listing (Fig. 2.2).

2.1.2 Assembler Tables and Logic

- Our simple assembler uses two internal tables: The OPTAB and SYMTAB.
  - OPTAB is used to look up mnemonic operation codes and translate them to their machine language equivalents.
    - LDA→00, STL→14, …
  - SYMTAB is used to store values (addresses) assigned to labels.
    - COPY→1000, FIRST→1000 …
- Location Counter LOCCTR
  - LOCCTR is a variable for assignment addresses.
  - LOCCTR is initialized to address specified in START.
  - When reach a label, the current value of LOCCTR gives the address to be associated with that label.

2.1.2 Assembler Tables and Logic

- The Operation Code Table (OPTAB)
  - Contain the mnemonic operation & its machine language equivalents (at least).
  - Contain instruction format & length.
  - Pass 1, OPTAB is used to look up and validate operation codes.
  - Pass 2, OPTAB is used to translate the operation codes to machine language.
  - In SIC/XE, assembler search OPTAB in Pass 1 to find the instruction length for incrementing LOCCTR.
  - Organize as a hash table (static table).

2.1.2 Assembler Tables and Logic

- The Symbol Table (SYMTAB)
  - Include the name and value (address) for each label.
  - Include flags to indicate error conditions
  - Contain type, length.
  - Pass 1, labels are entered into SYMTAB, along with assigned addresses (from LOCCTR).
  - Pass 2, symbols used as operands are look up in SYMTAB to obtain the addresses.
  - Organize as a hash table (static table).
  - The entries are rarely deleted from table.

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

2.1.2 Assembler Tables and Logic

- Pass 1 usually writes an intermediate file.
  - Contain source statement together with its assigned address, error indicators.
  - This file is used as input to Pass 2.
- Figure 2.4 shows the two passes of assembler.
  - Format with fields LABEL, OPCODE, and OPERAND.
  - Denote numeric value with the prefix #.

    #[OPERAND]

Pass 1

Pass 1:

```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save #[OPERAND] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end {if START}
   else
      initialize LOCCTR to 0

   write last line to intermediate file
   save (LOCCTR - starting address) as program length
end {Pass 1}
```

```
while OPCODE ≠ 'END' do
    begin
        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) into SYMTAB
                    end {if symbol}
                search OPTAB for OPCODE
                if found then
                    add 3 {instruction length} to LOCCTR
                else if OPCODE = 'WORD' then
                    add 3 to LOCCTR
                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
                    end {if BYTE}
                else
                    set error flag (invalid operation code)
            end {if not a comment}
        write line to intermediate file
        read next input line
    end {while not END}
```

Pass 2

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end {if START}
   write Header record to object program
   initialize first Text record
```

```
   write last Text record to object program
   write End record to object program
   write last listing line
end {Pass 2}
```

```
while OPCODE ≠ 'END' do
    begin
        if this is not a comment line then
            begin
                search OPTAB for OPCODE
                if found then
                    begin
                        if there is a symbol in OPERAND field then
                            begin
                                search SYMTAB for OPERAND
                                if found then
                                    store symbol value as operand address
                                else
                                    begin
                                        store 0 as operand address
                                        set error flag (undefined symbol)
                                    end
                            end {if symbol}
                        else
                            store 0 as operand address
                        assemble the object code instruction
                    end {if opcode found}
                else if OPCODE = 'BYTE' or 'WORD' then
                    convert constant to object code
                if object code will not fit into the current Text record then
                    begin
                        write Text record to object program
                        initialize new Text record
                    end
                add object code to Text record
            end {if not comment}
        write listing line
        read next input line
    end {while not END}
```

else
  if (found symbol==RSUB||
    found symbol== …||
    found symbol==…)
    store 0 as operand address
  else
    store 0 as operand address
    set error flag
assemble the object code inst.

2.2 Machine-Dependent Assembler Features

- Indirect addressing
  - Adding the prefix @ to operand (line 70).
- Immediate operands
  - Adding the prefix # to operand (lines 12, 25, 55, 133).
- Base relative addressing
  - Assembler directive BASE (lines 12 and 13).
- Extended format
  - Adding the prefix + to OP code (lines 15, 35, 65).
- The use of register-register instructions.
  - Faster and don't require another memory reference.

Figure 2.5: First

Line **Source statement**

```
COPY     START   0          COPY FILE FROM INPUT TO OUTPUT
FIRST    STL     RETADR     SAVE RETURN ADDRESS
         LDB     #LENGTH    ESTABLISH BASE REGISTER
         BASE    LENGTH
CLOOP    +JSUB   RDREC      READ INPUT RECORD
         LDA     LENGTH     TEST FOR EOF (LENGTH = 0)
         COMP    #0
         JEQ     ENDFIL     EXIT IF EOF FOUND
         +JSUB   WRREC      WRITE OUTPUT RECORD
         J       CLOOP      LOOP
ENDFIL   LDA     EOF        INSERT END OF FILE MARKER
         STA     BUFFER
         LDA     #3         SET LENGTH = 3
         STA     LENGTH
         +JSUB   WRREC      WRITE EOF
         J       @RETADR    RETURN TO CALLER
EOF      BYTE    C'EOF'
RETADR   RESW    1
LENGTH   RESW    1          LENGTH OF RECORD
BUFFER   RESB    4096       4096-BYTE BUFFER AREA
```

Figure 2.5: RDREC

```
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
```

Figure 2.5: WRREC

```
        .
        .          SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
        WRREC      CLEAR     X                CLEAR LOOP COUNTER
                   LDT       LENGTH
        WLOOP      TD        OUTPUT           TEST OUTPUT DEVICE
                   JEQ       WLOOP            LOOP UNTIL READY
                   LDCH      BUFFER,X         GET CHARACTER FROM BUFFER
                   WD        OUTPUT           WRITE CHARACTER
                   TIXR      T                LOOP UNTIL ALL CHARACTERS
                   JLT       WLOOP              HAVE BEEN WRITTEN
                   RSUB                       RETURN TO CALLER
        OUTPUT     BYTE      X'05'            CODE FOR OUTPUT DEVICE
                   END       FIRST
```

Figure 2.5 Example of a SIC/XE program.

2.2 Machine-Dependent Assembler Features

- SIC/XE
  - PC-relative/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 COMPR
  - larger memory → multi-programming (program allocation)

2.2 Machine-Dependent Assembler Features

- 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 disp (address) field
    - PC-relative: -2048~2047
    - Base-relative: 0~4095
  - Format 4: 20-bit address field (absolute addressing)

2.2.1 Instruction Formats & Addressing Modes

- The START statement
  - Specifies a beginning address of 0.
- Register-register instructions
  - CLEAR & TIXR, COMPR
- Register-memory instructions are using
  - Program-counter (PC) relative addressing
  - The program counter is advanced *after* each instruction is fetched and *before* it is executed.
  - PC will contain the address of the *next* instruction.

```
10    0000        FIRST  STL    RETADR         17202D
```

TA - (PC) = disp = 30H – 3H= 2D

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

```
                 .
                 .           SUBROUTINE TO WRITE RECORD FROM BUFFER
                 .
        105D     WRREC       CLEAR      X                    B410
        105F                 LDT        LENGTH               774000
        1062     WLOOP       TD         OUTPUT               E32011
        1065                 JEQ        WLOOP                332FFA
        1068                 LDCH       BUFFER,X             53C003
        106B                 WD         OUTPUT               DF2008
        106E                 TIXR       T                    B850
        1070                 JLT        WLOOP                3B2FEF
        1073                 RSUB                            4F0000
        1076     OUTPUT      BYTE       X'05'                05
                             END        FIRST
```

Figure 2.6 Program from Fig. 2.5 with object code.

```
+OP, e=1                              Extended
n=1, i=1,       OPcode+3,             Simple
@m, n=1, i=0,   OPcode+2,             Indirect
#C, n=0, i=1,   OPcode+1,             Immediate
xbpe        2: PC-relative
            4: base-relative
            8: index (m,X)
            1: extended
```

2.2.1 Instruction Formats & Addressing Modes

```
40    0017              J     CLOOP          3F2FEC
```

0006 - 001A = disp = -14

- Base (B), LDB #LENGTH, BASE LENGTH

```
160   104E              STCH  BUFFER, X      57C003
```

TA-(B) = 0036 - (B) = disp = 0036-0033 = 0003

- Extended instruction

```
15    0006        CLOOP +JSUB RDREC          4B101036
```

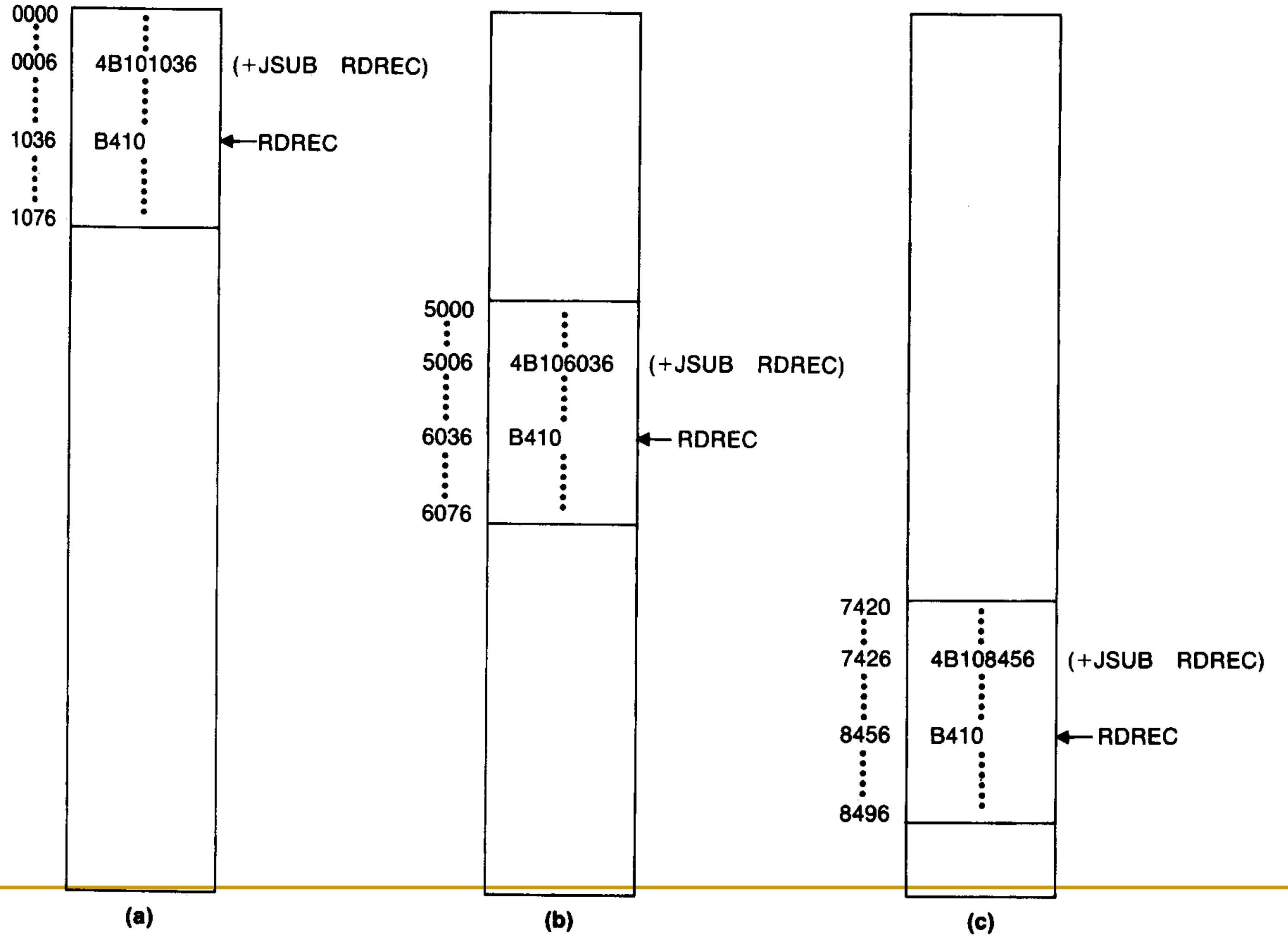
- Immediate instruction

```
55    0020              LDA   #3             010003
133   103C              +LDT  #4096          75101000
```

- PC relative + indirect addressing (line 70)

2.2.2 Program Relocation

- Absolute program, relocatable program



2.2.2 Program Relocation

Note that no matter where the program is loaded, RDREC is always 1036 bytes past the starting address of the program. This means that we can solve the relocation problem in the following way:

1. When the assembler generates the object code for the JSUB instruction we are considering, it will insert the address of RDREC *relative to the start of the program.* (This is the reason we initialized the location counter to 0 for the assembly.)

2. The assembler will also produce a command for the loader, instructing it to *add* the beginning address of the program to the address field in the JSUB instruction at load time.

2.2.2 Program Relocation

- **Modification record (direct addressing)**
  - 1 M
  - 2-7 Starting location of the address field to be modified, relative to the beginning of the program.
  - 8-9 Length of the address field to be modified, in *half bytes*.

M^000007^05

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05
M^000014^05
M^000027^05
E^000000
```

```
M00000705+COPY
M00001405+COPY
M00002705+COPY
```

2.3 Machine-Independent Assembler Features

- Write the value of a constant operand as a part of the instruction that uses it (Fig. 2.9).
- A literal is identified with the prefix =

```
45    001A    ENDFIL    LDA   =C'EOF'    032010
```

  - Specifies a 3-byte operand whose value is the character string EOF.

```
215   1062    WLOOP     TD    =X'05'     E32011
```

  - Specifies a 1-byte literal with the hexadecimal value 05

Line	Source statement			
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

RDREC

```
        .
        .       SUBROUTINE TO READ RECORD INTO BUFFER
        .
        RDREC   CLEAR   X               CLEAR LOOP COUNTER
                CLEAR   A               CLEAR A TO ZERO
                CLEAR   S               CLEAR S TO ZERO
               +LDT     #MAXLEN
        RLOOP   TD      INPUT           TEST INPUT DEVICE
                JEQ     RLOOP           LOOP UNTIL READY
                RD      INPUT           READ CHARACTER INTO REGISTER A
                COMPR   A,S             TEST FOR END OF RECORD (X'00')
                JEQ     EXIT            EXIT LOOP IF EOR
                STCH    BUFFER,X        STORE CHARACTER IN BUFFER
                TIXR    T               LOOP UNLESS MAX LENGTH
                JLT     RLOOP             HAS BEEN REACHED
        EXIT    STX     LENGTH          SAVE RECORD LENGTH
                RSUB                    RETURN TO CALLER
        INPUT   BYTE    X'F1'           CODE FOR INPUT DEVICE
```

WRREC

```
        .
        .            SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
        WRREC        CLEAR     X                 CLEAR LOOP COUNTER
                     LDT       LENGTH
        WLOOP        TD       =X'05'             TEST OUTPUT DEVICE
                     JEQ       WLOOP             LOOP UNTIL READY
                     LDCH      BUFFER,X          GET CHARACTER FROM BUFFER
                     WD       =X'05'             WRITE CHARACTER
                     TIXR      T                 LOOP UNTIL ALL CHARACTERS
                     JLT       WLOOP               HAVE BEEN WRITTEN
                     RSUB                        RETURN TO CALLER
                     END       FIRST
```

Figure 2.9 Program demonstrating additional assembler features.

2.3.1 Literals

- The difference between literal operands and immediate operands
  - =, #
  - Immediate addressing, the operand value is assembled as part of the machine instruction, no memory reference.
  - With a literal, the assembler generates the specified value as a constant at some other memory location. The *address* of this generated constant is used as the TA for the machine instruction, using PC-relative or base-relative addressing with memory reference.
- Literal pools
  - At the end of the program (Fig. 2.10).
  - Assembler directive LTORG, it creates a literal pool that contains all of the literal operands used since the previous LTORG.

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	

RDREC

```
                .
                .       SUBROUTINE TO READ RECORD INTO BUFFER
                .
        1036    RDREC   CLEAR   X               B410
        1038            CLEAR   A               B400
        103A            CLEAR   S               B440
        103C           +LDT     #MAXLEN         75101000
        1040    RLOOP   TD      INPUT           E32019
        1043            JEQ     RLOOP           332FFA
        1046            RD      INPUT           DB2013
        1049            COMPR   A,S             A004
        104B            JEQ     EXIT            332008
        104E            STCH    BUFFER,X        57C003
        1051            TIXR    T               B850
        1053            JLT     RLOOP           3B2FEA
        1056    EXIT    STX     LENGTH          134000
        1059            RSUB                    4F0000
        105C    INPUT   BYTE    X'F1'           F1
```

WRREC

```
                .
                .       SUBROUTINE TO WRITE RECORD FROM BUFFER
                .
        105D    WRREC   CLEAR   X               B410
        105F            LDT     LENGTH          774000
        1062    WLOOP   TD      =X'05'          E32011
        1065            JEQ     WLOOP           332FFA
        1068            LDCH    BUFFER,X        53C003
        106B            WD      =X'05'          DF2008
        106E            TIXR    T               B850
        1070            JLT     WLOOP           3B2FEF
        1073            RSUB                    4F0000
                        END     FIRST
        1076    *      =X'05'                   05
```

Figure 2.10 Program from Fig. 2.9 with object code.

2.3.1 Literals

- When to use LTORG (page 69, 4th paragraph)
  - The literal operand would be placed too far away from the instruction referencing.
  - Cannot use PC-relative addressing or Base-relative addressing to generate Object Program.
- Most assemblers recognize duplicate literals.
  - By comparison of the character strings defining them.
  - =C’EOF’ and =X’454F46’

2.3.1 Literals

- Allow literals that refer to the current value of the location counter.
  - Such literals are sometimes useful for loading base registers.

  LDB =*

  ; register B=beginning address of statement=current LOC

  BASE *

  ; for base relative addressing
- If a literal =* appeared on line 13 or 55
  - Specify an operand with value 0003 (Loc) or 0020 (Loc).

2.3.1 Literals

- Literal table (LITTAB)
  - Contains the literal name (=C'EOF'), the operand value (454F46) and length (3), and the address (002D).
  - Organized as a hash table.
  - Pass 1, the assembler creates or searches LITTAB for the specified literal name.
  - Pass 1 encounters a LTORG statement or the end of the program, the assembler makes a scan of the literal table.
  - Pass 2, the operand address for use in generating OC is obtained by searching LITTAB.

2.3.2 Symbol-Defining Statements

- Allow the programmer to define symbols and specify their values.
  - Assembler directive EQU.
  - Improved readability in place of numeric values.

```
        +LDT  #4096
MAXLEN  EQU   BUFEND-BUFFER (4096)
        +LDT  #MAXLEN
```

- Use EQU in defining mnemonic names for registers.
  - Registers A, X, L can be used by numbers 0, 1, 2.

```
        RMO   0, 1      A    EQU    0
        RMO   A, X      X    EQU    1
                        L    EQU    2
```

2.3.2 Symbol-Defining Statements

- The standard names reflect the usage of the registers.

```
BASE     EQU   R1
COUNT    EQU   R2
INDEX    EQU   R3
```

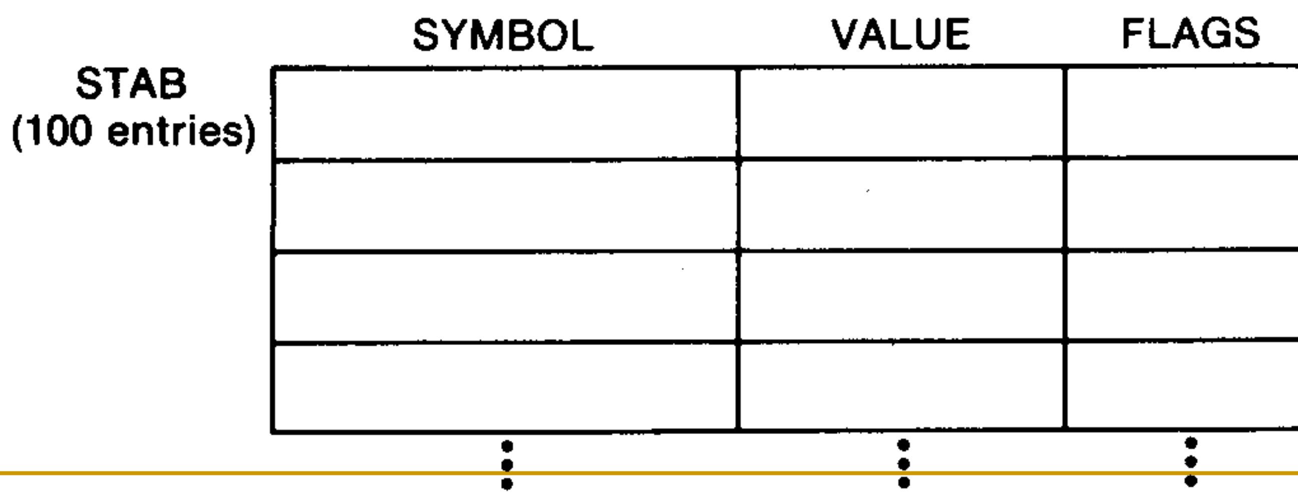
- Assembler directive ORG
  - Use to indirectly assign values to symbols.

    ORG value

  - The assembler resets its LOCCTR to the specified value.
  - ORG can be useful in label definition.

2.3.2 Symbol-Defining Statements

- The location counter is used to control assignment of storage in the object program
  - In most cases, altering its value would result in an incorrect assembly.
- ORG is used
  http://home.educities.edu.tw/wanker742126/index.html
  - SYMBOL is 6-byte, VALUE is 3-byte, and FLAGS is 2-byte.



2.3.2 Symbol-Defining Statements

STAB	SYMBOL	VALUE	FLAGS
(100 entries)	6	3	2

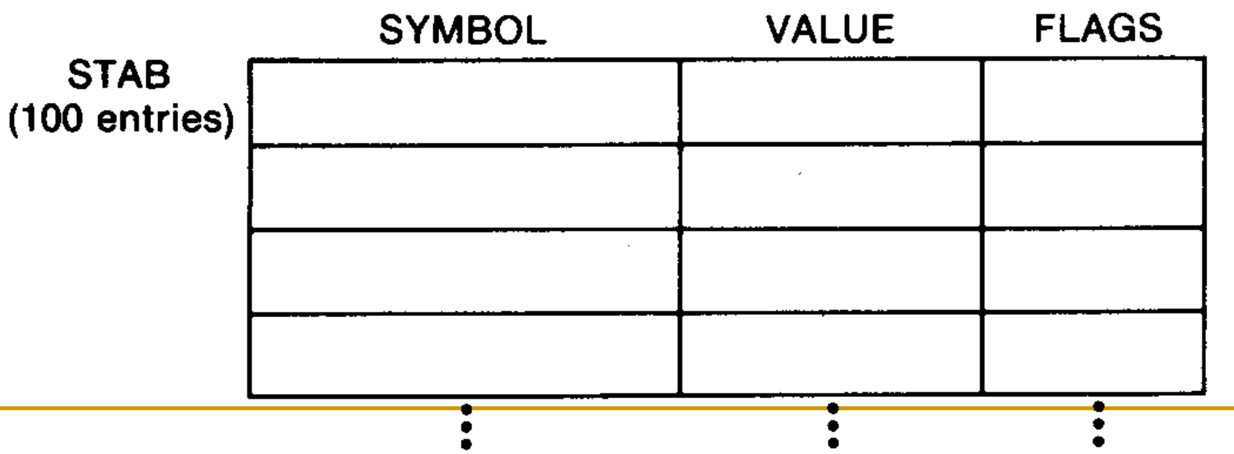
```
LOC
1000    STAB      RESB  1100

1000    SYMBOL    EQU   STAB +0
1006    VALUE     EQU   STAB +6
1009    FLAGS     EQU   STAB +9
```

- Use LDA VALUE, X to fetch the VALUE field form the table entry indicated by the contents of register X.

2.3.2 Symbol-Defining Statements

STAB	SYMBOL	VALUE	FLAGS
(100 entries)	6	3	2

```
1000    STAB      RESB  1100
                  ORG   STAB
1000    SYMBOL    RESB  6
1006    VALUE     RESW  1
1009    FLAGS     RESB  2
                  ORG   STAB+1100
```



2.3.2 Symbol-Defining Statements

- All terms used to specify the value of the new symbol — must have been defined previously in the program.

```
...
BETA    EQU   ALPHA
ALPHA   RESW  1
...
```

Need 2 passes

2.3.2 Symbol-Defining Statements

- All symbols used to specify new location counter value must have been previously defined.

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

- Forward reference

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


Need 3 passes

2.3.3 Expressions

- Allow arithmetic expressions formed
  - Using the operators +, -, ×, /.
  - Division is usually defined to produce an integer result.
  - Expression may be constants, user-defined symbols, or special terms.

```
106    1036       BUFEND    EQU    *
```

  - Gives BUFEND a value that is the address of the next byte after the buffer area.
- Absolute expressions or relative expressions
  - A relative term or expression represents some value (S+r), S: starting address, r: the relative value.

2.3.3 Expressions

107 1000 MAXLEN EQU BUFEND-BUFFER

- Both BUFEND and BUFFER are relative terms.
- The expression represents absolute value: the *difference* between the two addresses.
- Loc =1000 (Hex)
- The value that is associated with the symbol that appears in the source statement.
- BUFEND+BUFFER, 100-BUFFER, 3*BUFFER represent neither absolute values nor locations.

- Symbol tables entries

Symbol	Type	Value
RETADR	R	0030
BUFFER	R	0036
BUFEND	R	1036
MAXLEN	A	1000

2.3.4 Program Blocks

- The source program logically contained main, subroutines, data areas.
  - In a single block of object code.
- More flexible (Different blocks)
  - Generate machine instructions (codes) and data in a different order from the corresponding source statements.
- Program blocks
  - Refer to segments of code that are rearranged within a single object program unit.
- Control sections
  - Refer to segments of code that are translated into independent object program units.

2.3.4 Program Blocks

- Three blocks, Figure 2.11
  - Default (USE), CDATA (USE CDATA), CBLKS (USE CBLKS).
- Assembler directive USE
  - Indicates which portions of the source program blocks.
  - At the beginning of the program, statements are assumed to be part of the default block.
  - Lines 92, 103, 123, 183, 208, 252.
- Each program block may contain several separate segments.
  - The assembler will rearrange these segments to gather together the pieces of each block.

Main

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

RDREC

```
        .
        .               SUBROUTINE TO READ RECORD INTO BUFFER
        .
                USE
        RDREC   CLEAR   X               CLEAR LOOP COUNTER
                CLEAR   A               CLEAR A TO ZERO
                CLEAR   S               CLEAR S TO ZERO
               +LDT     #MAXLEN
        RLOOP   TD      INPUT           TEST INPUT DEVICE
                JEQ     RLOOP           LOOP UNTIL READY
                RD      INPUT           READ CHARACTER INTO REGISTER A
                COMPR   A,S             TEST FOR END OF RECORD (X'00')
                JEQ     EXIT            EXIT LOOP IF EOR
                STCH    BUFFER,X        STORE CHARACTER IN BUFFER
                TIXR    T               LOOP UNLESS MAX LENGTH
                JLT     RLOOP             HAS BEEN REACHED
        EXIT    STX     LENGTH          SAVE RECORD LENGTH
                RSUB                    RETURN TO CALLER
                USE     CDATA
        INPUT   BYTE    X'F1'           CODE FOR INPUT DEVICE
```

WRREC

```
195      .
200      .          SUBROUTINE TO WRITE RECORD FROM BUFFER
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.

2.3.4 Program Blocks

- Pass 1, Figure 2.12
  - The block number is started form 0.
  - A separate location counter for each program block.
  - The location counter for a block is initialized to 0 when the block is first begun.
  - Assign each block a starting address in the object program (location 0).
  - Labels, block name or block number, relative addr.
  - Working table is generated

Block name	Block number	Address	End	Length
default	0	0000	0065	0066 (0~0065)
CDATA	1	0066	0070	000B (0~000A)
CBLKS	2	0071	1070	1000 (0~0FFF)

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			.			
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                      .
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.

2.3.4 Program Blocks

- Pass 2, Figure 2.12
  - The assembler needs the address for each symbol relative to the start of the object program.
  - Loc shows the relative address and block number.
  - Notice that the value of the symbol MAXLEN (line 70) is shown without a block number.

```
20        0006 0     LDA       LENGTH 032060
```

0003(CDATA) +0066 =0069 =TA

using program-counter relative addressing

TA - (PC) =0069-0009 =0060 =disp

2.3.4 Program Blocks

- Separation of the program into blocks.
  - Because the large buffer (CBLKS) is moved to the end of the object program.
  - No longer need extended format, base register, simply a LTORG statement.
  - No need Modification records.
  - Improve program readability.
- Figure 2.13
  - Reflect the starting address of the block as well as the relative location of the code within the block.
- Figure 2.14
  - Loader simply loads the object code from each record at the dictated.
  - CDATA(1) & CBLKS(1) are not actually present in OP.

2.3.4 Program Blocks

```
HCOPY ^000000^001071
T^000000^1E^172063^4B2021^032060^290000^332006^4B203B^3F2FEE^032055^0F2056^010003
T^00001E^09^0F2048^4B2029^3E203F                                          Default 1
T^000027^1D^B410^B400^B440^75101000^E32038^332FFA^DB2032^A004^332008^57A02F^B850
T^000044^09^3B2FEA^13201F^4F0000                                          Default 2
T^00006C^01^F1                                                            CDATA 2
T^00004D^19^B410^772017^E3201B^332FFA^53A016^DF2012^B850^3B2FEF^4F0000    Default 3
T^00006D^04^454F46^05                                                     CDATA 3
E^000000
```

Figure 2.13 Object program corresponding to Fig. 2.11.

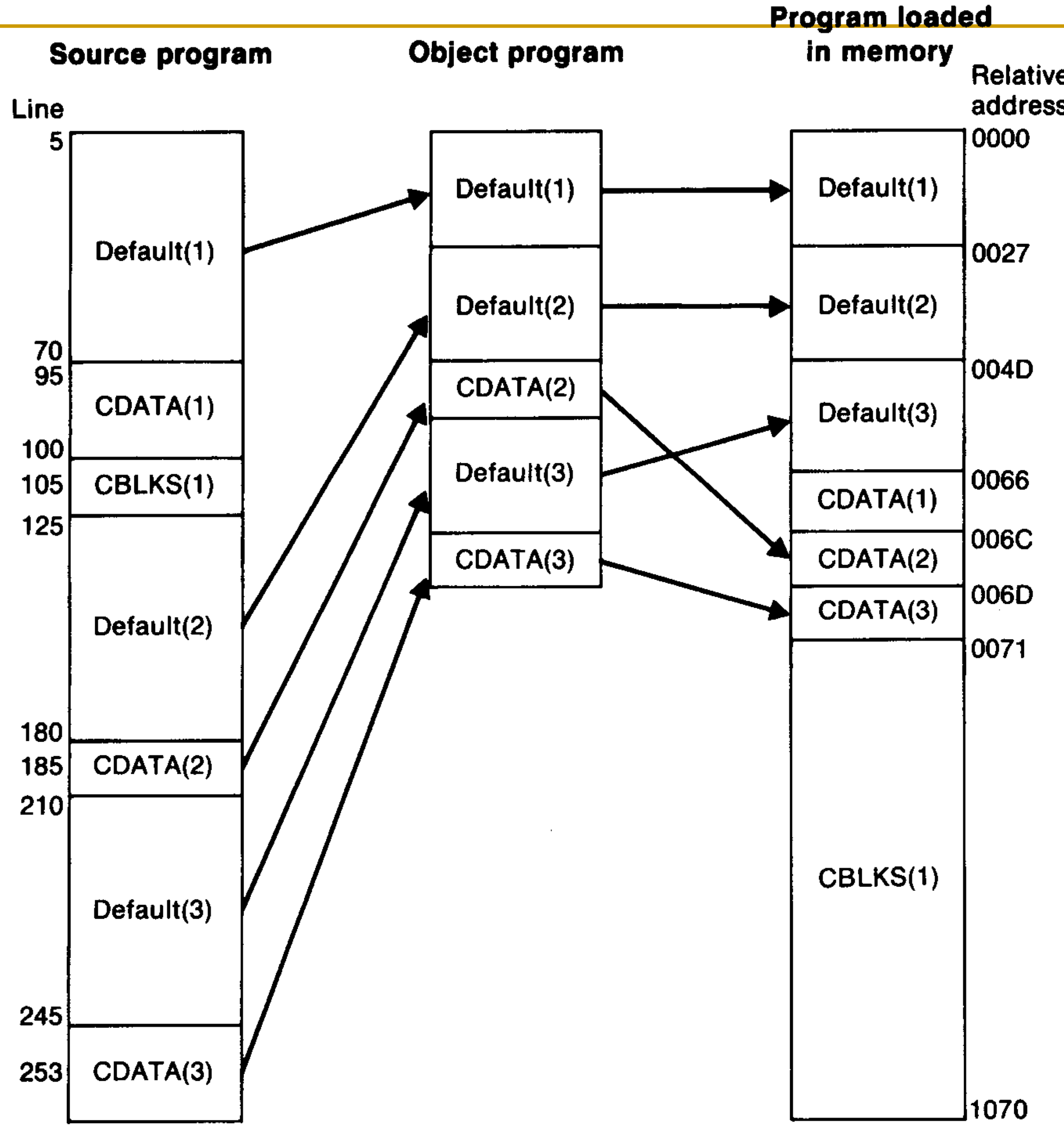


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.

2.3.5 Control Sections & Program Linking

- Control section
  - Handling of programs that consist of multiple control sections.
  - Each control section is a part of the program.
  - Can be assembled, loaded and relocated independently.
  - Different control sections are most often used for subroutines or other logical subdivisions of a program.
  - The programmer can assemble, load, and manipulate each of these control sections separately.
  - More Flexibility then the previous.
  - Linking control sections together.

2.3.5 Control Sections & Program Linking

- External references (external symbol references)
  - Instructions in one control section might need to refer to instructions or data located in another section.
- Figure 2.15, multiple control sections.
  - Three sections, main COPY, RDREC, WRREC.
  - Assembler directive CSECT.
  - Assembler directives EXTDEF and EXTREF for external symbols.
  - The order of symbols is not significant.

```
COPY      START      0
          EXTDEF     BUFFER, BUFEND, LENGTH
          EXTREF     RDREC, WRREC (symbol name)
```

Line Source statement

```
COPY     START    0               COPY FILE FROM INPUT TO OUTPUT
         EXTDEF   BUFFER,BUFEND,LENGTH
         EXTREF   RDREC,WRREC
FIRST    STL      RETADR          SAVE RETURN ADDRESS
CLOOP    +JSUB    RDREC           READ INPUT RECORD
         LDA      LENGTH          TEST FOR EOF (LENGTH = 0)
         COMP     #0
         JEQ      ENDFIL          EXIT IF EOF FOUND
         +JSUB    WRREC           WRITE OUTPUT RECORD
         J        CLOOP           LOOP
ENDFIL   LDA      =C'EOF'         INSERT END OF FILE MARKER
         STA      BUFFER
         LDA      #3              SET LENGTH = 3
         STA      LENGTH
         +JSUB    WRREC           WRITE EOF
         J        @RETADR         RETURN TO CALLER
RETADR   RESW     1
LENGTH   RESW     1               LENGTH OF RECORD
         LTORG
BUFFER   RESB     4096            4096-BYTE BUFFER AREA
BUFEND   EQU      *
MAXLEN   EQU      BUFEND-BUFFER
```

```
RDREC   CSECT
        .
        .       SUBROUTINE TO READ RECORD INTO BUFFER
        .
        EXTREF  BUFFER,LENGTH,BUFEND
        CLEAR   X               CLEAR LOOP COUNTER
        CLEAR   A               CLEAR A TO ZERO
        CLEAR   S               CLEAR S TO ZERO
        LDT     MAXLEN
RLOOP   TD      INPUT           TEST INPUT DEVICE
        JEQ     RLOOP           LOOP UNTIL READY
        RD      INPUT           READ CHARACTER INTO REGISTER A
        COMPR   A,S             TEST FOR END OF RECORD (X'00')
        JEQ     EXIT            EXIT LOOP IF EOR
       +STCH    BUFFER,X        STORE CHARACTER IN BUFFER
        TIXR    T               LOOP UNLESS MAX LENGTH
        JLT     RLOOP             HAS BEEN REACHED
EXIT   +STX     LENGTH          SAVE RECORD LENGTH
        RSUB                    RETURN TO CALLER
INPUT   BYTE    X'F1'           CODE FOR INPUT DEVICE
MAXLEN  WORD    BUFEND-BUFFER
```

```
WRREC    CSECT
.
.        SUBROUTINE TO WRITE RECORD FROM BUFFER
.
         EXTREF   LENGTH,BUFFER
         CLEAR    X                CLEAR LOOP COUNTER
        +LDT      LENGTH
WLOOP    TD      =X'05'            TEST OUTPUT DEVICE
         JEQ      WLOOP            LOOP UNTIL READY
        +LDCH     BUFFER,X         GET CHARACTER FROM BUFFER
         WD      =X'05'            WRITE CHARACTER
         TIXR     T                LOOP UNTIL ALL CHARACTERS
         JLT      WLOOP              HAVE BEEN WRITTEN
         RSUB                      RETURN TO CALLER
         END      FIRST
```

Figure 2.15 Illustration of control sections and program linking.

2.3.5 Control Sections & Program Linking

- Figure 2.16, the generated object code.

```
15      0003  CLOOP +JSUB RDREC        4B100000
160     0017        +STCH BUFFER,X     57900000
```

  - The LOC of all control section is started form 0
  - RDREC is an external reference.
  - The assembler has no idea where the control section containing RDREC will be loaded, so it cannot assemble the address.
  - The proper address to be inserted at load time.
  - Must use extended format instruction for external reference (M records are needed).

```
190     0028  MAXLEN      WORD  BUFEND-BUFFER
```

  - An expression involving two external references.

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	

```
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
```

```
0000    WRREC   CSECT
        .
        .       SUBROUTINE TO WRITE RECORD FROM BUFFER
        .
                EXTREF   LENGTH,BUFFER
0000            CLEAR    X                   B410
0002           +LDT      LENGTH              77100000
0006    WLOOP   TD      =X'05'               E32012
0009            JEQ      WLOOP               332FFA
000C           +LDCH     BUFFER,X            53900000
0010            WD      =X'05'               DF2008
0013            TIXR     T                   B850
0015            JLT      WLOOP               3B2FEE
0018            RSUB                         4F0000
                END      FIRST
001B    *      =X'05'                        05
```

Figure 2.16 Program from Fig. 2.15 with object code.

2.3.5 Control Sections & Program Linking

- The loader will add to this data area with the address of BUFEND and subtract from it the address of BUFFER. (COPY and RDREC for MAXLEN)
- Line 190 and 107, in 107, the symbols BUFEND and BUFFER are defined in the same section.
- The assembler must remember in which control section a symbol is defined.
- The assembler allows the same symbol to be used in different control sections, lines 107 and 190.

- Figure 2.17, two new records.
  - Defined record for EXTDEF, relative address.
  - Refer record for EXTREF.

Define record:

Col. 1	D
Col. 2–7	Name of external symbol defined in this control section
Col. 8–13	Relative address of symbol within this control section (hexadecimal)
Col. 14–73	Repeat information in Col. 2–13 for other external symbols

Refer record:

Col. 1	R
Col. 2–7	Name of external symbol referred to in this control section
Col. 8–73	Names of other external reference symbols

The other information needed for program linking is added to the Modification record type. The new format is as follows.

Modification record (revised):

Col. 1	M
Col. 2–7	Starting address of the field to be modified, relative to the beginning of the control section (hexadecimal)
Col. 8–9	Length of the field to be modified, in half-bytes (hexadecimal)
Col. 10	Modification flag (+ or –)
Col. 11–16	External symbol whose value is to be added to or subtracted from the indicated field

2.3.5 Control Sections & Program Linking

- Modification record
  - M
  - Starting address of the field to be modified, relative to the beginning of the control section (Hex).
  - Length of the field to be modified, in *half-bytes*.
  - Modification flag (+ or -).
  - External symbol.

```
M^000004^05+RDREC
M^000028^06+BUFEND
M^000028^06-BUFFER
```

```
M00000705
M00001405
M00002705
```

to

```
M00000705+COPY
M00001405+COPY
M00002705+COPY
```

- Use Figure 2.8 for program relocation.

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001D1720274B1000000320232900003320074B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T00003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000

HRDREC 00000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E
```

```
H^WRREC ^000000^00001C
R^LENGTH^BUFFER
T^000000^1C^B410^77100000^E32012^332FFA^53900000^DF2008^B850^3B2FEE^4F0000^05
M^000003^05^+LENGTH
M^00000D^05^+BUFFER
E
```

Figure 2.17 Object program corresponding to Fig. 2.15.

2.4 Assembler Design Options
2.4.1 Two-Pass Assembler

- Most assemblers
  - Processing the source program into two passes.
  - The internal tables and subroutines that are used only during Pass 1.
  - The SYMTAB, LITTAB, and OPTAB are used by both passes.
- The main problems to assemble a program in one pass involves forward references.

2.4.2 One-Pass Assemblers

- Eliminate forward references
  - Data items are defined before they are referenced.
  - But, forward references to labels on instructions cannot be eliminated as easily.
  - Prohibit forward references to labels.
- Two types of one-pass assembler. (Fig. 2.18)
  - One type produces object code directly in memory for immediate execution.
  - The other type produces the usual kind of object program for later execution.

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	48
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	30
35	201E		JSUB	WRREC	48
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	48
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000

```
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           30
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
```

```
                  .
                  .       SUBROUTINE TO WRITE RECORD FROM BUFFER
                  .
        2061      OUTPUT  BYTE     X'05'              05
                  .
        2062      WRREC   LDX      ZERO               041006
        2065      WLOOP   TD       OUTPUT             E02061
        2068              JEQ      WLOOP              302065
        206B              LDCH     BUFFER,X           50900F
        206E              WD       OUTPUT             DC2061
        2071              TIX      LENGTH             2C100C
        2074              JLT      WLOOP              382065
        2077              RSUB                        4C0000
                          END      FIRST
```

Figure 2.18 Sample program for a one-pass assembler.

2.4.2 One-Pass Assemblers

- Load-and-go one-pass assembler
  - The assembler avoids the overhead of writing the object program out and reading it back in.
  - The object program is produced in memory, the handling of forward references becomes less difficult.
  - Figure 2.19(a), shows the SYMTAB after scanning line 40 of the program in Figure 2.18.
  - Since RDREC was not yet defined, the instruction was assembled with no value assigned as the operand address (denote by - - - -).

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	100948--	--00100C	28100630	----48--
2020	--3C2012			
•				
•				
•				

Symbol	Value	Reference list
LENGTH	100C	
RDREC	*	2013 | 0
THREE	1003	
ZERO	1006	
WRREC	*	201F | 0
EOF	1000	
ENDFIL	*	201C | 0
RETADR	1009	
BUFFER	100F	
CLOOP	2012	
FIRST	200F	

Figure 2.19(a) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.

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	001006E0	20393020	43D82039	28100630
2050	-----5490	0F		
•				
•				
•				

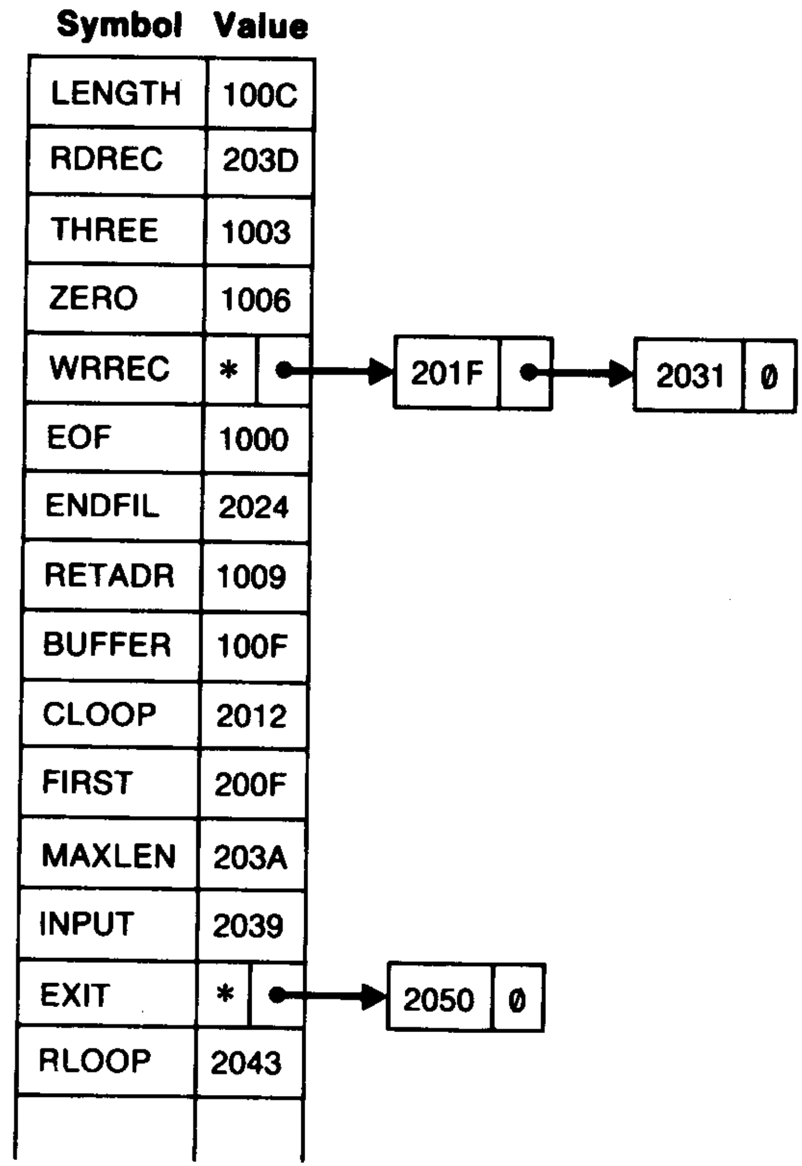


Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

2.4.2 One-Pass Assemblers

- Load-and-go one-pass assembler
  - RDREC was then entered into SYMTAB as an undefined symbol, the address of the operand field of the instruction (2013) was inserted.
  - Figure 2.19(b), when the symbol ENDFIL was defined (line 45), the assembler placed its value in the SYMTAB entry; it then inserted this value into the instruction operand field (201C).
  - At the end of the program, all symbols must be defined without any * in SYMTAB.
  - For a load-and-go assembler, the actual address must be known at assembly time.

2.4.2 One-Pass Assemblers

- Another one-pass assembler by generating OP
  - Generate another Text record with correct operand address.
  - When the program is loaded, this address will be inserted into the instruction by the action of the loader.
  - Figure 2.20, the operand addresses for the instructions on lines 15, 30, and 35 have been generated as 0000.
  - When the definition of ENDFIL is encountered on line 45, the third Text record is generated, the value 2024 is to be loaded at location 201C.
  - The loader completes forward references.

```
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^480000^810094^C0000F^1001000
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
```

ENDFIL

RDREC

EXIT

WRREC

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

2.4.2 One-Pass Assemblers

- In this section, simple one-pass assemblers handled absolute programs (SIC example).

2.4.3 Multi-Pass Assemblers

- Use EQU, any symbol used on the RHS be defined previously in the source.

LOC				Pass1	2	3
1000	LDA	#0		1000	1000	1000
1003	ALPHA	EQU	BETA	????	????	1003
1003	BETA	EQU	DELTA	????	1003	1003
1003	DELTA	RESW	1	1003	1003	1003

  - Need 3 passes!
- Figure 2.21, multi-pass assembler

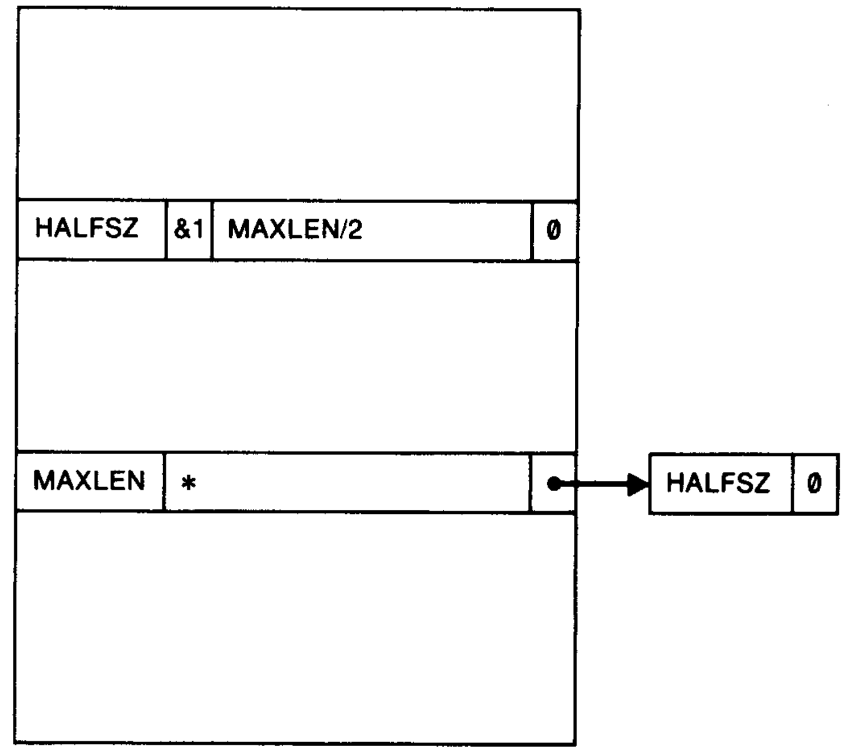
```
1     HALFSZ     EQU     MAXLEN/2
2     MAXLEN     EQU     BUFEND-BUFFER
3     PREVBT     EQU     BUFFER-1
                         .
                         .
                         .
4     BUFFER     RESB    4096
5     BUFEND     EQU     *
```

(a)

2.4.3 Multi-Pass Assemblers

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

(a)

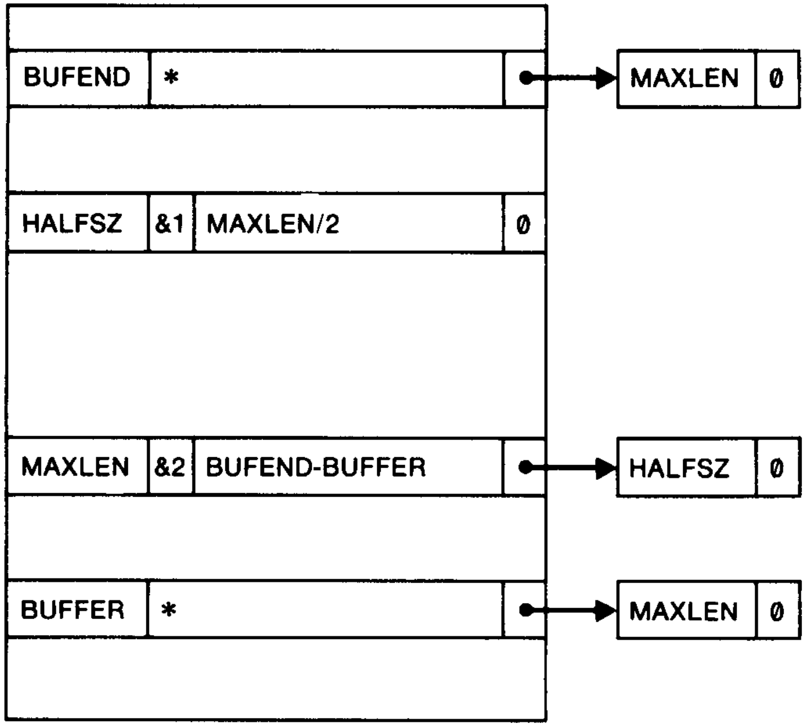


(b)

2.4.3 Multi-Pass Assemblers

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

(a)

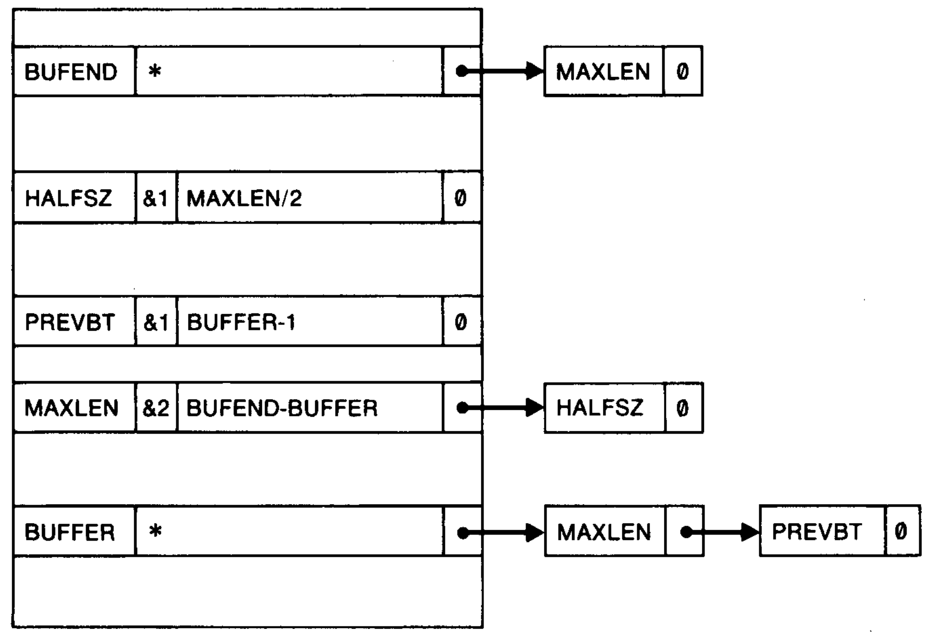


(c)

2.4.3 Multi-Pass Assemblers

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

(a)

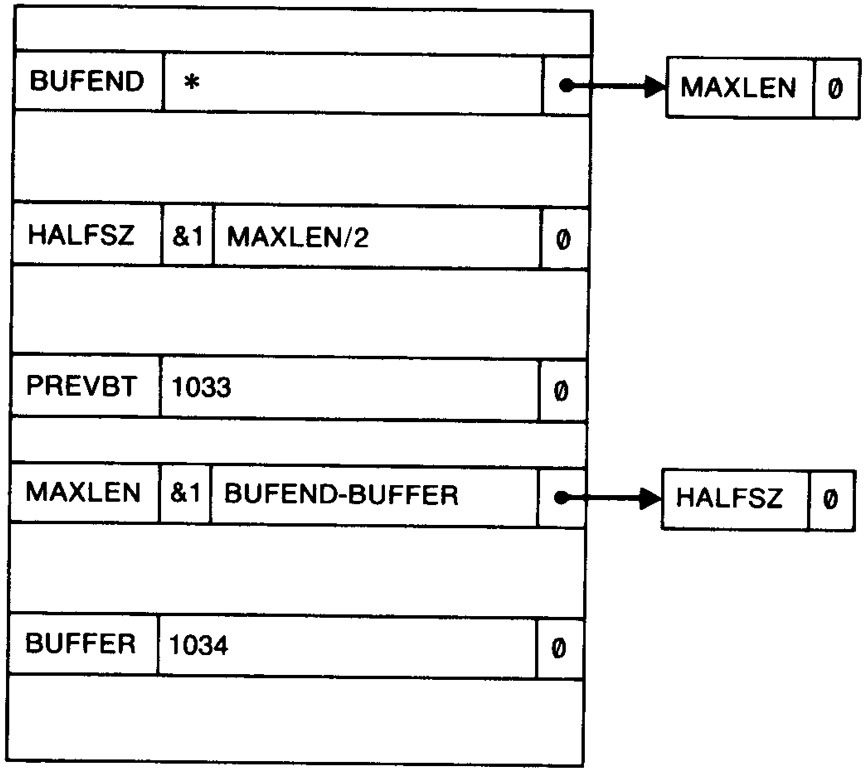


(d)

2.4.3 Multi-Pass Assemblers

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

(a)

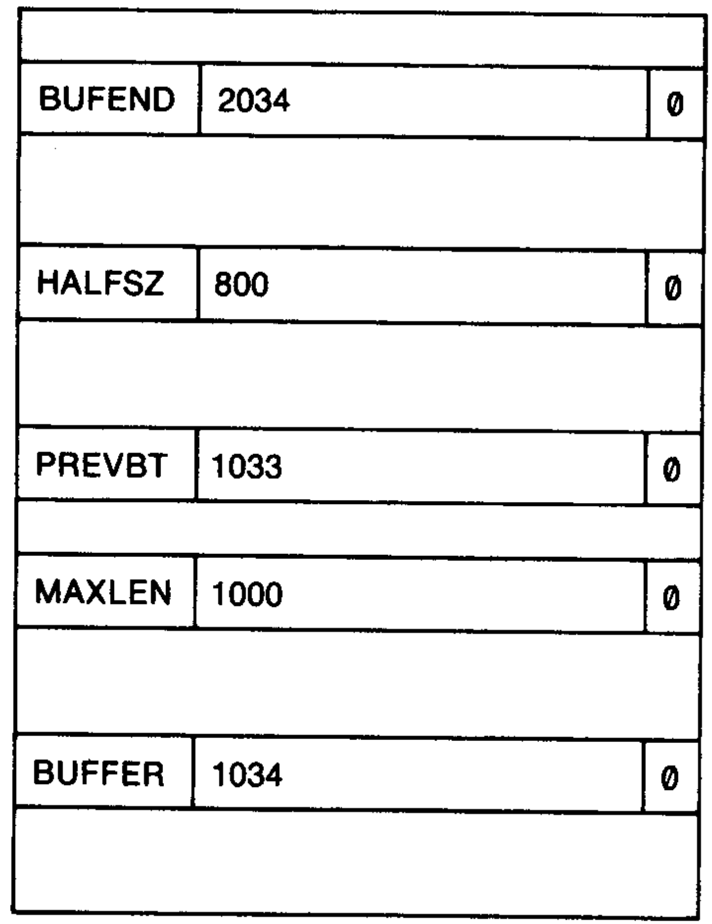


(e)

2.4.3 Multi-Pass Assemblers

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

(a)



(f)