System Programming

(Chapter 2: Assembler)

김지환교수

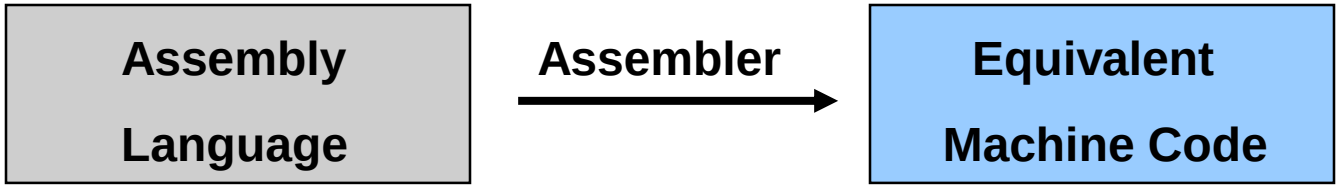
Office: AS관 713

Tel: 705-8924

Email: kimjihwan@sogang.ac.kr

What is Assembler ?

Fundamental Functions of an Assembler



- Translate mnemonic operation codes to their machine language equivalents.
- Assign machine addresses to symbol labels.

Other than the basic functions, designing of an assembler depends heavily upon the machine architecture (e.g., instruction format, addressing mode, etc.).

Outline of Chapter 2

Section 2.1 : Fundamental operations of an assembler using SIC.

Section 2.2 : Typical extensions via hardware considerations using SIC/XE.

Section 2.3 : Most commonly encountered machine-independent features.

Section 2.4 : Important design options of an assembler.

Information on the SIC Assembly Language

Mnemonic instructions : Section 1.3.1 and Appendix A.

Indexed addressing is indicated by adding the modifier " ,X " following the operand (line 160 of the program in next slide).

Lines beginning with " . " contain comments only.

<u>Assembler directives</u> (pseudo-instructions) : These statements are not translated into machine codes. Instead, they provide instructions to the assembler itself.

START : Specify name and starting address for the program.

END : Indicate the end of the source program and (optionally) specify the first executable instruction in the program.

BYTE : Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.

WORD : Generate one-word integer constant.

RESB : Reserve the indicated number of bytes for a data area.

RESW : Reserve the indicated number of words for a data area.

Example of a SIC Assembly Language Program

Source statement

```
  5    COPY      START   1000        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    ZERO
 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     THREE       SET LENGTH = 3
 60              STA     LENGTH
 65              JSUB    WRREC       WRITE EOF
 70              LDL     RETADR      GET RETURN ADDRESS
 75              RSUB                RETURN TO CALLER
 80    EOF       BYTE    C'EOF'
 85    THREE     WORD    3
 90    ZERO      WORD    0
 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     LDX     ZERO        CLEAR LOOP COUNTER
130              LDA     ZERO        CLEAR A TO ZERO
135    RLOOP     TD      INPUT       TEST INPUT DEVICE
140              JEQ     RLOOP       LOOP UNTIL READY
145              RD      INPUT       READ CHARACTER INTO REGISTER A
150              COMP    ZERO        TEST FOR END OF RECORD (X'00')
155              JEQ     EXIT        EXIT LOOP IF EOR
160              STCH    BUFFER,X    STORE CHARACTER IN BUFFER
165              TIX     MAXLEN      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
190    MAXLEN    WORD    4096
195
200              SUBROUTINE TO WRITE RECORD FROM BUFFER
205
210    WRREC     LDX     ZERO        CLEAR LOOP COUNTER
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              TIX     LENGTH      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
```

Assembly Language Program with Object Code

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	
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		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
210	2061	WRREC	LDX	ZERO	041030
215	2064	WLOOP	TD	OUTPUT	E02079
220	2067		JEQ	WLOOP	302064
225	206A		LDCH	BUFFER,X	509039
230	206D		WD	OUTPUT	DC2079
235	2070		TIX	LENGTH	2C1036
240	2073		JLT	WLOOP	382064
245	2076		RSUB		4C0000
250	2079	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Figure 2.2 Program from Fig. 2.1 with object code.

A Simple SIC Assembler

Steps for translating source program to object code (not necessarily in the order given) :

1. Convert mnemonic operation codes to their machine language equivalents – e.g., translate STL to 14 (line 10).
2. Convert symbolic operands to their equivalent machine addresses – e.g., translate RETADR to 1033 (line 10).
3. Build the machine instructions in the proper format.
4. Convert the data constants specified in the source program into their internal machine representations – e.g., translate EOF to 454F46 (line 80).
5. Write the object program and the assembly listing.

- **All of these functions except number 2 can easily be accomplished by sequential processing of the source program. However, the translation of addresses such as line 10 presents a problem (forward reference) – need 2 passes.**

Object Program Format (from Slide 5)

```
HCOPY  001000001107A
T001000 1E 141033 482039 001036 281030 301015 482061 3C1003 00102A 0C1039 00102D
T00101E 15 0C1036 482061 081033 4C0000 454F46 000003 000000
T002039 1E 041030 001030 E0205D 30203F D8205D 281030 302057 549039 2C205E 38203F
T002057 1C 101036 4C0000 F1 001000 041030 E02079 302064 509039 DC2079 2C1036
T002073 07 382064 4C0000 05
E001000
```

Header Record:

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

Text Record:

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

End Record:

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

Two Passes of SIC Assembler

Pass 1

1. Assign addresses to all statements in the program.
2. Save the values (addresses) assigned to all labels for use in Pass2.
3. Perform some processing of assembler directives. (This includes processing that affects address assignment, such as determining the length of data areas defined by BYTE, RESW, etc)

Pass 2

1. Assemble instructions (translating operation codes and looking up addresses).
2. Generate data values defined by BYTE, WORD, etc.
3. Perform processing of assembler directives not done during Pass 1.
4. Write the object program and the assembly listing.

Data Structures for Assembler (1)

LOCCTR (Location Counter)

Variable that is used to help in the assignment of addresses.

Initialized to the beginning address specified in the START statement.

The length of the assembled instruction is added to LOCCTR.

Current value of LOCCTR gives the address to be associated with the label.

OPTAB (Operation Code Table)

Contains mnemonic operation codes and their machine language equivalents.

Also contains information about instruction format and length.

In Pass 1, OPTAB is used to look up and validate operation codes.

In Pass 2, it is used to translate the operation codes to machine language.

OPTAB is usually organized as a hash table (key: mnemonic operation code).

OPTAB is a static table – entries are not normally added to or deleted from it.

Data Structures for Assembler (2)

SYMTAB (Symbol Table)

Store values (addresses) assigned to labels.

Contains the flags to indicate error conditions (e.g., labels defined in two different places, etc.).

May also contain other information about the data area or instruction labeled.

In Pass 1, labels are entered into SYMTAB as they are encountered in the source program, along with their assigned addresses (from LOCCTR).

In Pass 2, symbols used as operands are looked up in SYMTAB to obtain the addresses to be inserted in the assembled instructions.

SYMTAB is usually organized as a hash table for efficiency of insertion and retrieval.

Hashing function should be designed carefully to take into account the facts that entries are not rarely deleted from this table and programmers often select many labels that have similar characteristics (e.g., labels starting with the same characters - LOOP1, LOOP2, LOOPA, or with the same length – A, X, Y, Z).

Assembler Algorithm

Pass 1 of the assembler usually write an intermediate file that contains each source statement together with its assigned address, error indicators, etc.

This file is the input to the Pass 2.

Pointers into OPTAB and SYMTAB may be retained for each operation code and symbol used that allows us to avoid the need to repeat many of the table searching operations.

Slides 12 and 13 show the logic flow of the two passed of our assembler. (Text p. 53 and p. 54)

You are strongly urged to follow through the logic in these algorithms, applying them by hand to the program in Slide 4 to produce the object program of Slide 7.

Algorithm for Pass 1 of Assembler

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

Algorithm for Pass 2 of Assembler

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

Example of a SIC/XE Program

```
Line          Source statement

              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
```

Example Program with Object Code

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13			BASE	LENGTH	
15	0006	CLOOP	+JSUB	RDREC	4B101036
20	000A		LDA	LENGTH	032026
25	000D		COMP	#0	290000
30	0010		JEQ	ENDFIL	332007
35	0013		+JSUB	WRREC	4B10105D
40	0017		J	CLOOP	3F2FEC
45	001A	ENDFIL	LDA	EOF	032010
50	001D		STA	BUFFER	0F2016
55	0020		LDA	#3	010003
60	0023		STA	LENGTH	0F200D
65	0026		+JSUB	WRREC	4B10105D
70	002A		J	@RETADR	3E2003
80	002D	EOF	BYTE	C'EOF'	454F46
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
110					
115			SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
125	1036	RDREC	CLEAR	X	B410
130	1038		CLEAR	A	B400
132	103A		CLEAR	S	B440
133	103C		+LDT	#4096	75101000
135	1040	RLOOP	TD	INPUT	E32019
140	1043		JEQ	RLOOP	332FFA
145	1046		RD	INPUT	DB2013
150	1049		COMPR	A,S	A004
155	104B		JEQ	EXIT	332008
160	104E		STCH	BUFFER,X	57C003
165	1051		TIXR	T	B850
170	1053		JLT	RLOOP	3B2FEA
175	1056	EXIT	STX	LENGTH	134000
180	1059		RSUB		4F0000
185	105C	INPUT	BYTE	X'F1'	F1
195					
200			SUBROUTINE TO WRITE RECORD FROM BUFFER		
205					
210	105D	WRREC	CLEAR	X	B410
212	105F		LDT	LENGTH	774000
215	1062	WLOOP	TD	OUTPUT	E32011
220	1065		JEQ	WLOOP	332FFA
225	1068		LDCH	BUFFER,X	53C003
230	106B		WD	OUTPUT	DF2008
235	106E		TIXR	T	B850
240	1070		JLT	WLOOP	3B2FEF
245	1073		RSUB		4F0000
250	1076	OUTPUT	BYTE	X'05'	05
255			END	FIRST	

Review of SIC/XE Addressing Modes

The addressing modes supported by the SIC/XE can be largely classified by two issues: how to calculate TA (Target Address) and how to interpret the calculated TA.

How to calculate TA (Use b, p, x bits)

Direct addressing mode : TA = address => *b = p = 0/1*

PC relative addressing mode : TA = (PC) + address => *b = 0, p = 1*

Base relative addressing mode : TA = (B) + address => *b = 1, p = 0*

How to interpret TA (Use i, n bits)

Simple addressing mode : (TA) => *i = 1, n = 1*

Indirect addressing mode : ((TA)) => *i = 0, n = 1*

Immediate addressing mode : TA => *i = 1, n = 0*

Features of SIC/XE Assembler

Indirect addressing is indicated by adding prefix @ to the operand (line 70).

Immediate operands are denoted with the prefix # (line 25, 55, 133).

Instructions that refer to memory are normally assembled using either the PC relative or the base relative mode.

If the displacement required for both PC relative and base relative addressing are too large to fit into a 3-byte instruction, then the 4-byte extended format (format 4) must be used.

The extended instruction format is specified with the prefix + added to the operation code (line 15, 35, 65).

For speed-up, use register-to-register instructions, and immediate/indirect addressing as much as possible.

Assembling SIC/XE Instructions (1)

START statement now specifies a beginning address of 0 : re-locatable program => will be covered in the next section.

Translation of register-to-register instructions such as CLEAR (line 125) and COMPR (line 150) presents no new problems.

The conversion of register mnemonics to numbers can be done with a separate table; or SYMAB may be preloaded with the register names (A, X, etc) and their values (0, 1, etc).

Most of the register-to-memory instructions are assembled using either PC relative or base relative addressing.

If the resulting displacement is larger than the values in 12-bit field (For PC relative: -2048 <= disp <= 2047, for base relative: 0 <= disp <= 4095), the 4-byte format 4 (20 bit field) is used.

It is the programmer's responsibility that he/she should put + to represent format 4 instructions.

e.g.,) 15 0006 CLOOP +JSUB RDREC 4B101036

Assembling SIC/XE Instructions (2)

PC relative addressing

e.g. 1) 10 0000 FIRST STL RETADR 17202D

(PC) = 0003 (The PC is advanced ***after*** each instruction is fetched and ***before*** it is executed. Thus PC will contain the address of the ***next*** instruction)

Address of RETADR = 0030. Therefore, displacement = 30 – 3 = 2D.

Simple addressing : i = 1, n = 1.

PC relative addressing : p = 1, b = 0, x = 0 (no indexing).

e.g. 2) 40 0017 J CLOOP 3F2FEC

(PC) = 001A

Address of CLOOP = 0006. Therefore displacement = 6 – 1A = -14.

Negative values will be represented by 2's complement: -14 => FEC.

Assembling SIC/XE Instructions (3)

Base relative addressing : similar to PC relative but differs in that the assembler does not know the content of the B register and it is the responsibility of the programmer to tell the assembler what the base register will contain during execution (via BASE assembler directive – line 13)

e.g. 1)

```
12    0003    LDB     #LENGTH      69202D
13            BASE    LENGTH
```

If we are to use register B for another purpose, we have to use the assembler directive NOBASE to inform the assembler that the contents of the base register can no longer be relied upon for addressing.

e.g. 2)

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

(B) = 0033 (the address of LENGTH)

Address of BUFFER = 0036. Therefore displacement = 36 – 33 = 3.

x = 1, b = 1, p = 0, i = 1, n = 1 : Simple Base relative addressing.

Assembling SIC/XE Instructions (4)

Choice between PC relative addressing and base relative addressing mode : arbitrary.

Try PC relative first, try base relative next. If it is not possible to represent the displacement with 12 bits, use format 4.

e.g. 1) 20 000A LDA LENGTH 032026 : PC relative

e.g. 2) 175 1056 EXIT STX LENGTH 134000 : Base relative

Immediate addressing

e.g. 1) 55 0020 LDA #3 010003

i = 1, other bits = 0.

e.g. 2) 133 103C +LDT #4096 75101000

4096 can't be represented with 12 bits. Use format 4.

e.g. 3) 12 0003 LDB #LENGTH 69202D

PC relative + immediate addressing.

Assembling SIC/XE Instructions (5)

Indirect addressing

e.g.) 70 002A J @RETADDR 3E2003

PC relative + indirect addressing.

(PC) = 002D

Address of RETADR = 30. Therefore, displacement = 30 – 2D = 3.

Program Relocation

It is often desirable to have more than one program at a time sharing the memory and other resources of the machine. (Multi-programming) => The actual starting address of the program is not known until load time.

e.g.) 5 1000 COPY START 1000 <= should be loaded at 1000

55 101B LDA THREE 00102D

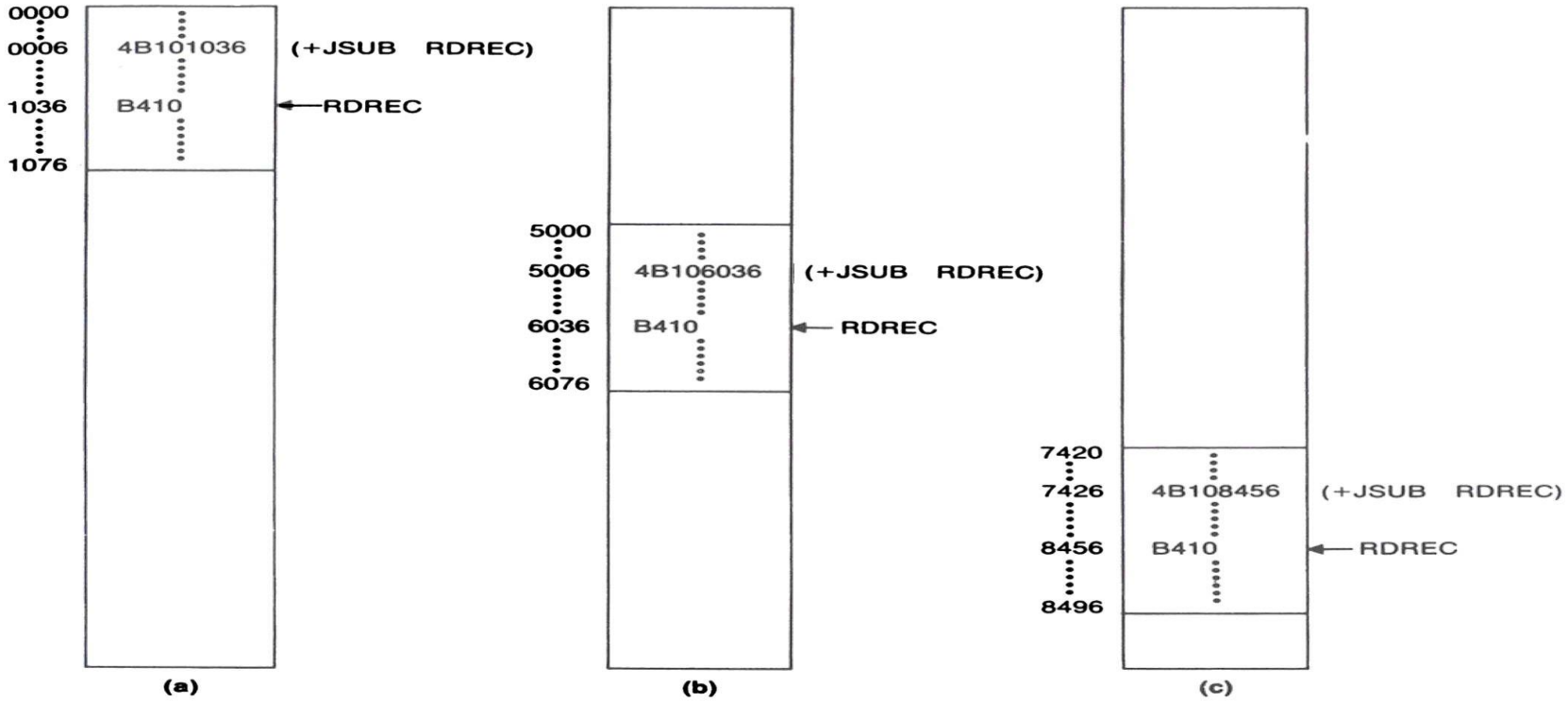
- If this program is loaded at 2000 and memory address 102D should be modified.

- Since the assembler does not know the actual location where the program will be loaded, it identifies those parts of the object program and let the loader know.
- An object program that contains the information necessary to perform this modification is called a *re-locatable* program.

Example of a Program Relocation

Fig 2.5 and Fig 2.6

```
15    CLOOP  +JSUB    RDREC   4B101036
...
125   RDREC  CLEAR   X        B410
```



Modification Record (1)

Relocation can be solved in the following way:

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* (starting address should be 0 – START 0 instruction).

The assembler 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. => Modification record.

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 (hex).

Modification Record (2)

The length is stored in half-bytes because the address field to be modified may not occupy an integral number of bytes. (e.g., 20 bit address -> 5 half bytes)

The starting location is the location of the byte containing the leftmost bits of the address field to be modified.

If the length contains an odd number of half bytes, it is assumed to begin in the middle of the first byte at the starting location.

```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200 74B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T001036 1DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T001053 1D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T00107007 3B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

Rules for Modification

The only parts of the program that require modification at load time are those that specify direct (as opposed to relative) addresses.

For this SIC/XE program, the only such direct addresses are found in extended format (4 bytes) instructions.

This is an advantage of relative addressing.

Program with Additional Features (Fig 2.9)

```
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
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    #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
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      =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
```

Example Program with Object Code (Fig 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					
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					
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

Literals (1) (강의제외 참조용)

Using Constants (Review)

Immediate addressing

e.g. 1) LDA #3

BYTE directive : Generate *character* or *hexadecimal* constant.

e.g. 1) LABEL1 BYTE C'EOF'

e.g. 2) LABEL2 BYTE X'05'

WORD directive: Generate *one-word integer* constant.

e.g. 1) LABEL1 WORD 3

- **Literal : a constant operand used as a part of the instruction. Identified with the prefix =, followed by a specification of the literal value (same notation as in the BYTE statement).**

e.g. 1) 45	001A	ENDFILL	LDA	=C'EOF'	032010
e.g. 2) 215	1062	WLOOP	TD	=X'05'	E32011

Literals (2) (강의제외 참조용)

Difference between an immediate operand and a literal.

Immediate addressing : The operand value is assembled as part of the machine instruction.

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 target address for the machine instruction. => same as the constant definition with a label attached to it (Compare line 45 and 215 in Fig 2.10 and Fig 2.6)

- **Where are the literals placed ? => *literal pools* (all of the literal operands used in a program are gathered together into one or more literal pools)**
- **Location of the literal pools**
  - At the end of the program (following the END statement). – Check Fig 2.10
  - After the LTORG assembler directive. Create a literal pool that contains all of the literal operands used since the previous LTORG (or the beginning of the program). – Check line 93 in Fig 2.10.

Literals (3) (강의제외 참조용)

Why LTORG ?

Avoid placing literal operand too far away from the instruction so that we can use PC relative addressing mode.

Avoid having to use extended format instructions when referring to the literals.

- **Most assemblers recognize duplicate literals :**
  - Comparison of the character strings defining them. (line 215 and 230 in Fig 2.10)
  - Additional saving is possible if we look at the generated data value (e.g., =C'EOF' and =X'454F46' specify identical operand values). If the benefits realized are not enough to justify the additional complexity, store both literals.
- **Literals whose value depends upon their location in the program (often denoted by the symbol *).**
  - e.g.) BASE *
    LDB =*
  - Should be careful when detecting duplicate literals. If this literal appears on line 13 in Fig 10, it will specify an operand with value 0003. If the same literal appears on line 55, it will specify an operand with value 0020.

Handing Literal Operands in the Assembler(강의제외 참조용)

Basic Data Structure : a literal table called LITTAB.

Contains the literal name, the operand value and length, and the address assigned to the operand when it is placed in a literal pool.

LITTAB is often organized as a hash table, using the literal name or value as the key.

Steps for handling literal operands.

Pass 1 :

Search LITTAB for the specified literal name.

If the literal is already there, no action is needed; If not, the literal is added to LITTAB (leaving the address unassigned).

When pass 1 encounters a LTORG or the end of the program, assign addresses to the literals currently in the table. The location counter is updated.

Pass 2 :

Search LITTAB for each literal operand encountered and insert the values at the appropriate places in the object program.

If a literal value represents an address, the assembler also generates the appropriate Modification Record.

EQU (Equate) Assembler Directive (1)

EQU (Equate) allows programmer to define symbols and specify their values.

Syntax : *symbol* EQU *value*

The symbol is also entered into the SYMTAB.

The value may be given as a constant or as any expression involving constants and previously defined symbols.

One common use of EQU is to establish symbolic names that can be used for improved readability. (Compare with #define in C)

(e.g.)

```
        +LDT   #4096     can be written as

MAXLEN  EQU    4096
        +LDT   #MAXLEN
```

EQU (Equate) Assembler Directive (2)

Another common use of EQU is in defining names for registers if the assembler expects register numbers instead of names.

(e.g.)

```
        RMO     0, 1 can be written as RMO   A, X   if

A       EQU     0
X       EQU     1
L       EQU     2
  ……                    are defined.
```

Case for a machine with general purpose registers (e.g., R0, R1, R2, …)

(e.g.)

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

Restrictions Common to EQU

In the case of EQU,

All symbols used on the right-hand side of the statement must have been defined previously in the program.

(e.g.) The sequence

```
ALPHA   RESW  1
BETA    EQU   ALPHA     ---  is valid, however

BETA    EQU   ALPH
ALPHA   RESW  1         ---  is not valid.
```

ORG (Origin) Assembler Directive (1) (강의제외 참조용)

ORG (Origin) is used to indirectly assign values to symbols.

Syntax : ORG value

The assembler resets its location counter (LOCCTR) to the specified value => This will affect the values of all labels defined until the next ORG.

The value is a constant or an expression involving constants and previously defined symbols.

Consider the case where we are defining a symbol table with the following structure:

STAB

(100 entries)

Symbol	Value	Flags
6 byte symbol	1 word	2 bytes

... . .

```
STAB    RESB    1100
```

ORG (Origin) Assembler Directive (2) (강의제외 참조용)

Two ways to efficiently access the symbol table (STAB).

Method 1: Use indexed addressing (X register) with EQU directive.

(e.g.) Define

```
SYMBOL    EQU    STAB
VALUE     EQU    STAB+6
FLAGS     EQU    STAB+9
```

And access each field (e.g., VALUE field) using "LDA VALUE, X", where X will contains the offset of the desired entry from the beginning of the table.

Method 2: Use ORG directive.

(e.g.) Define

```
STAB      RESB   1100
          ORG    STAB          : Reset LOCCTR to STAB
SYMBOL    RESB   6             : 1st 6 bytes are reserved for SYMBOL
VALUE     RESW   1             : 2nd 3 bytes are reserved for VALUE
FLAGS     RESB   2             : 3rd 2 bytes are reserved for FLAGS
          ORG    STAB+1100     : Set LOCCTR back to its previous value
                               : STAB+1100 can be omitted (simply ORG)
```

Expressions (1)

Definitions

Expression : Terms with operators such as +, -, *, /.

Terms : Constants, user-defined symbols, or special term (e.g., * - value of current LOCCTR)

Examples from Fig 2.10

(e.g. 1) 106	BUFEND	EQU	*
(e.g. 2) 107	MAXLEN	EQU	BUFEND-BUFFER

The values of terms and expressions are either relative or absolute.

Terms

Constant : absolute term.

Symbols on instructions and data areas : relative terms.

Symbol whose value is given by EQU : either absolute term or relative term.

Expressions (2)

Expressions

An expression that contains only absolute terms : absolute expression.

- (e.g.) 10-3, DATA-10 (where DATA is declared as a constant).

Absolute expressions may also contain relative terms if the relative terms occur in pairs and the terms in each such pair have opposite signs.

- (e.g.) 10 + BUFEND - BUFFER

A relative term is the one in which all of the relative terms except one can be paired as described above; the remaining unpaired relative term must have a positive sign.

- (e.g.) BUFEND – BUFFER + LENGTH

No relative terms may enter into a multiplication or division operation.

Expressions that do not meet the conditions given above generate errors.

- (e.g.) BUFEND + BUFFER, 100 – BUFFER, or 3 * BUFFER

A relative term represents some location within the program. When relative terms are paired with opposite signs, the dependency on the program starting address is canceled out -> absolute value.

Expressions (3)

To determine the type of an expression, we must keep track of the types for al symbols.

For this purpose, we need a flag in the symbol table to indicate the type of value (absolute or relative) in addition to the value itself.

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

With this information, the assembler can easily determine the type of each expression and generate Modification records for relative values.

Program Blocks

So far, all assembled programs were treated as a unit (single block of object code) although they contained subroutines, data areas, etc. => Machine instructions and data appeared in the same order as they were written in the source program.

Some assemblers allow the generated machine instructions and data to appear in the object program in a different order by creating several independent parts in the program.

Program Blocks : Segments of code that are rearranged within a single object program unit.

Syntax : USE *Block Name*

Blocks with same names are treated as same blocks. If the name is omitted, it is the unnamed block.

Example of program blocks : Fig. 2.11 and 2.12 (line 92, 103, 123, 183) shows 3 blocks, *Unnamed*, *CDATA*, *CBLKS*.

An Example with Multiple Program Blocks

```
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
  110
  115           SUBROUTINE TO READ RECORD INTO BUFFER
  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
  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
```

An Example with Object Code

```
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
```

Pass 1 for a Program with Program Blocks

Pass 1

A separate location counter (LOCCTR) for each program block is maintained (Fig 2.12).

The LOCCTR for a block is initialized to 0 when the block is first begun (Line 92 in Fig 2.12).

The current LOCCTR value is saved when switching to another block, and the saved value is restored when resuming a previous block (Line 123 in Fig 2.12).

The address of each label is relative to the start of each block.

The latest LOCCTR value for each block indicates the length of that block.

At the end of Pass 1, the assembler constructs a table as shown below :

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

Pass 2 for a Program with Program Blocks

Pass 2

The assembler needs the address for each symbol relative to the start of the object program (not the start of each block).

The assembler simply adds the location of the symbol, relative to the start of each block, to the assigned block starting address.

(e.g. 1)

Line	Loc	Block		Source Statement		Object Code
20	0006	0		LDA	LENGTH	032060
…..						
100	0003	1	LENGH	RESW	1	

- Relative address of LENGTH relative to block 1 (CDATA) = 0003
- The starting address for CDATA is 0066.
- Target address of LENGTH = 0003 + 0066 = 0069.
- Use PC-relative addressing (PC = 009).
- Displacement = 0069 – 0009 = 0060.

Advantages of using Program Blocks

Reduce the addressing problems.

No need to use extended format (line 15, 35, 65).

No need to use base relative addressing (no LDB and BASE statements).

Reduce the problems of placing literals.

Include a LTORG statement in the CDATA block to be sure that the literals are placed ahead of any large data areas.

The use of program blocks is one way of satisfying both machine consideration and human factors :

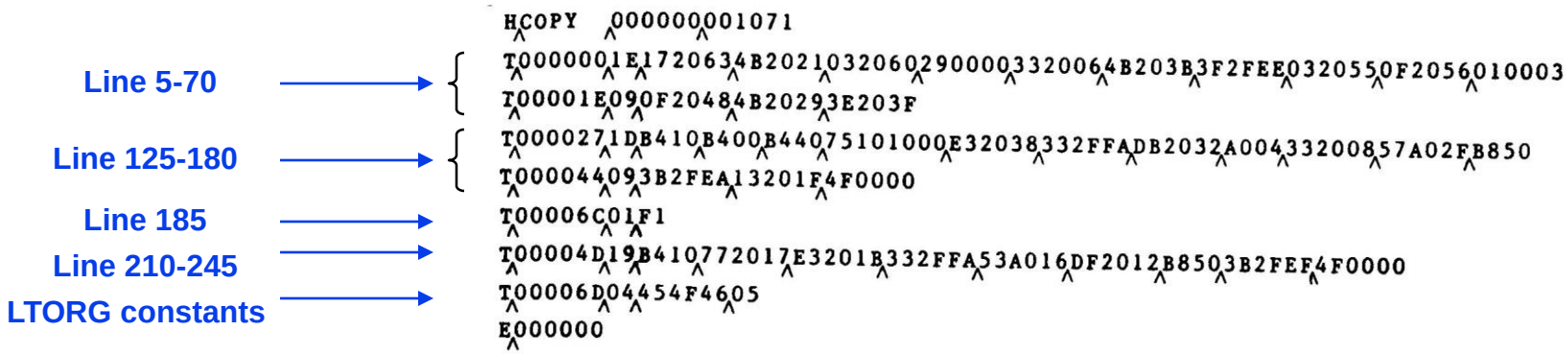
Machine consideration: Part of the object program appear in memory in a particular order.

Human factors: Improve readability by putting data areas close to the source statements that reference them -> Need long jump and extended addressing.

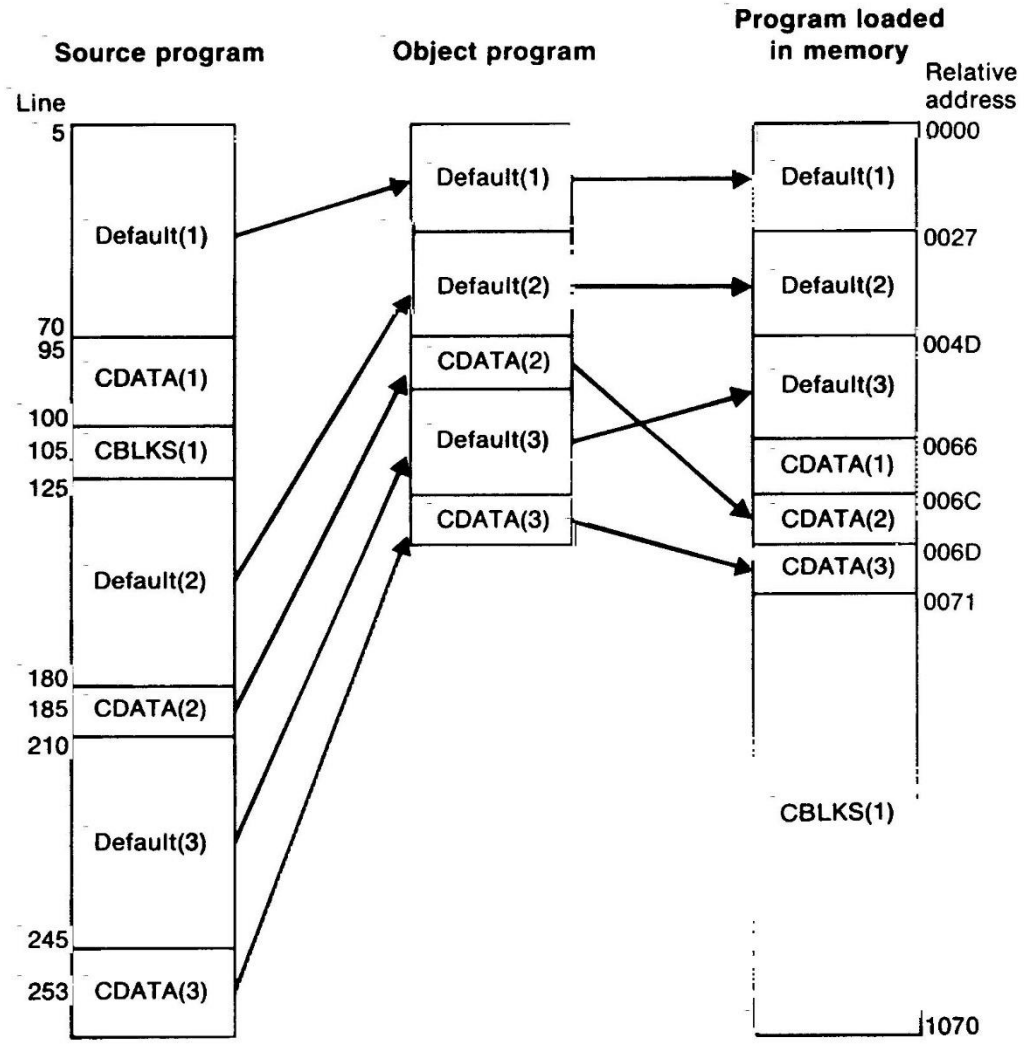
Object Program for Fig 2.11

It is not necessary to physically rearrange the generated code in the object program to place the pieces of each program block together.

The assembler simply writes the object code as it is generated during pass 2 and insert the proper load address in each Text record => The loader will rearrange them.



Assembly and Loading Process



Control Sections

A part of the program that can be assembled, loaded and relocated independently of the others.

Different control sections are most often used for subroutines or other logical subdivisions of a program.

Syntax : *Section Name* CSECT

The assembler establishes a separate location counter (beginning at 0) for each control section.

The instructions in one control section may need to refer to instructions or data located in another section.

EXTDEF (External Definition): for external symbols that are defined in this control section and may be used by other sections.

EXTREF (External Reference): for symbols that are used in this control section and are defined elsewhere.

Control section names do not need to be named in an EXTDEF statement.

Same symbols can be used in different control sections (line 107 and line 190).

Example of Control Sections (Fig 2.15)

```
Line        Source statement
   5  COPY     START   0                   COPY FILE FROM INPUT TO OUTPUT
   6           EXTDEF  BUFFER,BUFEND,LENGTH
   7           EXTREF  RDREC,WRREC
  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
  95  RETADR   RESW    1
 100  LENGTH   RESW    1                   LENGTH OF RECORD
 103           LTORG
 105  BUFFER   RESB    4096                4096-BYTE BUFFER AREA
 106  BUFEND   EQU     *
 107  MAXLEN   EQU     BUFEND-BUFFER

 109  RDREC    CSECT
 110           .
 115           .       SUBROUTINE TO READ RECORD INTO BUFFER
 120           .
 122           EXTREF  BUFFER,LENGTH,BUFEND
 125           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
 185  INPUT    BYTE    X'F1'               CODE FOR INPUT DEVICE
 190  MAXLEN   WORD    BUFEND-BUFFER

 193  WRREC    CSECT
 195           .
 200           .       SUBROUTINE TO WRITE RECORD FROM BUFFER
 205           .
 207           EXTREF  LENGTH,BUFFER
 210           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
```

Fig 2.15 with Object Code (Fig 2.16)

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
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

Handling External References by the Assembler

Since the instructions in one control section may need to refer to instructions or data located in another section, the assembler generates information so that the linker can perform the required linking.

(e.g. 1)

```
15   0003    CLOOP    +JSUB    RDREC          4B100000
```

RDREC is named in the EXTREF statement.

Since the assembler has no idea where the control section containing RDREC will be loaded, it generates an extended format instruction with an address of zero.

(e.g. 2)

```
160  0017             +STCH    BUFFER, X      57900000
```

(e.g. 3) – Check the difference between line 107 and line 190.

```
190  0028   MAXLEN    WORD     BUFEND-BUFFER  000000
```

Define Record and Refer Record

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 (hex).
Col. 14-73	Repeat information in Col. 2-13 for other external symbols.

Refer Record:

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

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 (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.

Object Program for Fig 2.15

```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001D172027 4B1000003202329000033200 74B100000 3F2FEC032016 0F2016
T00001D0D0100030F200A4B1000003E2000
T000030034 54F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000


HRDREC 000000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A004332009579000000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E


HWRREC 000000000001C
RLENGTHBUFFER
T0000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```