Spring 2016

Chap.2 Assemblers

Jongsun Choi

jongsun.choi@ssu.ac.kr

Goal of this Chapter

- Discuss the design and implementation of Assemblers

- Any assembler must perform
  - Translation of assembly codes to machine language
  - Assignment of machine addresses to symbolic labels

Chap.2 Assemblers

1. Basic Assembler Functions
2. Machine-Dependent Assembler functions
3. Machine-Independent Assembler functions
4. Assembler Design Options

Assemblers

2.1 Basic Assembler Functions

Jongsun Choi
jongsun.choi@ssu.ac.kr

Basic Assembler Functions

- SIC assembler language program (Fig. 2.1)

- Assembler Directives
  - **START**: specifies name and starting address for the program
  - **END**: indicates the end of the source program and (optionally) specifies the first executable instruction in the program
  - **BYTE**: generates character a hexadecimal constant, occupying as many as bytes as needed to represent the constant
  - **WORD**: generates one-word integer constant
  - **RESB**: reserves the indicated number of bytes for a data area
  - **RESW**: reserves the indicated number of words for a data area

Example of a SIC program (Fig 2.1)

```
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
```

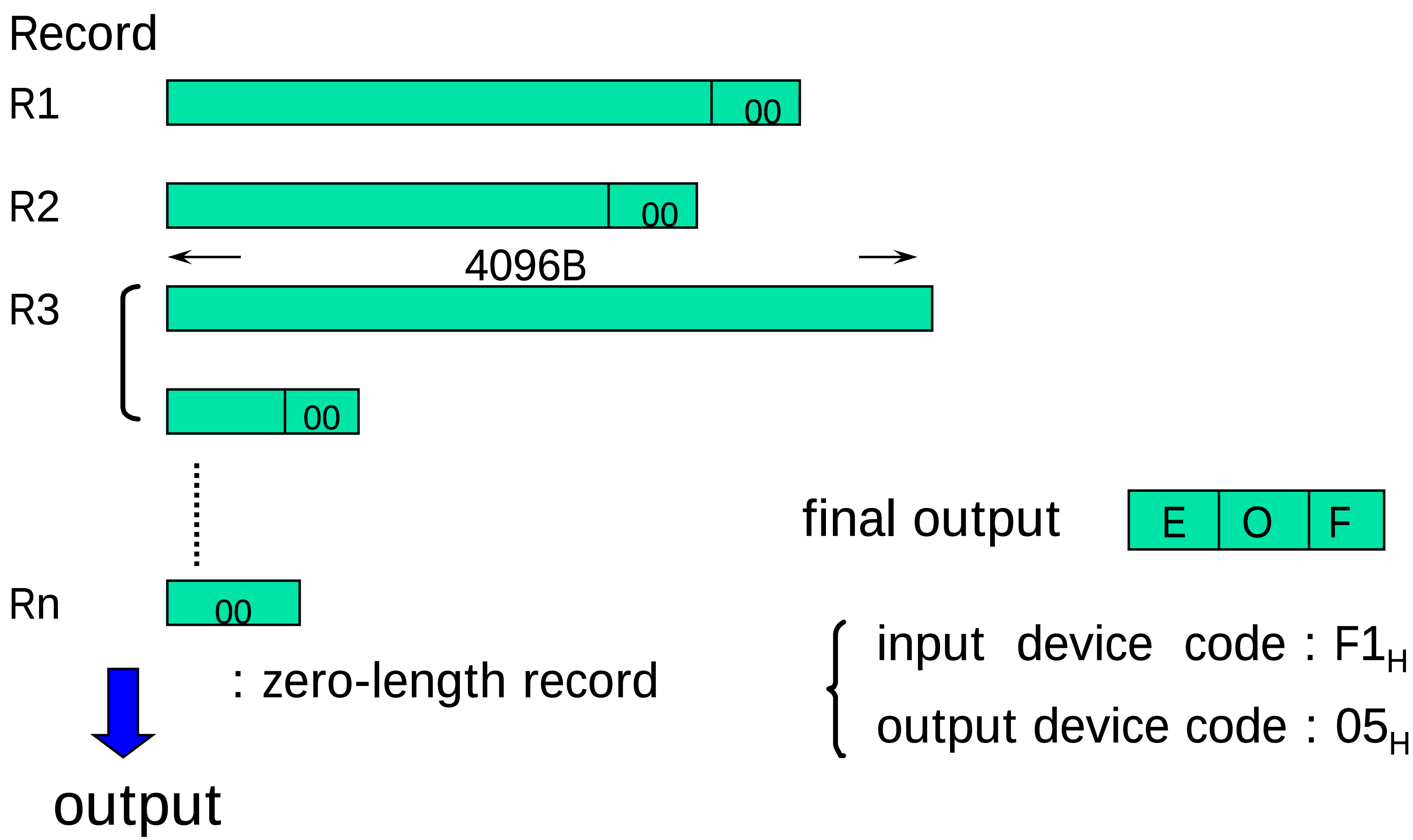
Example of a SIC program (Fig 2.1)

```
.
.       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
```

Example of a SIC program (Fig 2.1)

```
.
.       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
```

Program Logic (1/2)



Program logic (2/2)

- 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

- Subroutines (JSUB, RSUB)
  - RDREC, WRREC
  - Save link (L) register first before nested jump

A Simple SIC Assembler

(1) Convert mnemonic 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

A Simple SIC Assembler

- Example of Instruction Assemble

STCH BUFFER, X **549039**

8	1	15
opcode	x	address

$(1010100)_2$ 1 $(001)_2$ $(0000)_2$ $(0011)_2$ $(1001)_2$

$(54)_{16}$ $(9)_{16}$ $(039)_{16}$

A Simple SIC Assembler

- (1), (3), (4), and (5) can be easily accomplished
- (2)

```
10  1000  FIRST  STL  RETADR
```

→ *Forward reference*
  - ✓ reference to a label that is defined later in the program
  - ✓ not know the address that will be assigned to RETADR

→ *2 pass assembler*
  - ✓ it scans source program in twice

- 2 pass assembler
  - First pass
    - Scan the source program for label definitions and assign addresses
  - Second pass
    - Perform most of actual translation previously described

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

A Simple SIC Assembler

- Assembler directives (or pseudo-instructions)
  - Not translated into machine instructions
  - Provide instructions to the assembler itself
  - E.g.
    - BYTE, WORD
      - ✓ direct the assembler to generate constants as part of the object program
    - RESB, RESW
      - ✓ instruct the assembler to reserve memory locations without generating data values
    - START, END
      - ✓ specify the starting/end address for the program

Object Code of SIC Program (Fig 2.2)

```
1000   COPY     START   1000
1000   FIRST    STL     RETADR          141033
1003   CLOOP    JSUB    RDREC           482039
1006            LDA     LENGTH          001036
1009            COMP    ZERO            281030
100C            JEQ     ENDFIL          301015
100F            JSUB    WRREC           482061
1012            J       CLOOP           3C1003
1015   ENDFIL   LDA     EOF             00102A
1018            STA     BUFFER          0C1039
101B            LDA     THREE           00102D
101E            STA     LENGTH          0C1036
1021            JSUB    WRREC           482061
1024            LDL     RETADR          081033
1027            RSUB                    4C0000
102A   EOF      BYTE    C'EOF'          454F46
102D   THREE    WORD    3               000003
1030   ZERO     WORD    0               000000
1033   RETADR   RESW    1
1036   LENGTH   RESW    1
1039   BUFFER   RESB    4096
```

Object Code of SIC Program (Fig 2.2)

※ Addresses 1033-2038 is simply reserved by the loader for use by the program during execution

```
                .
                .       SUBROUTINE TO READ RECORD INTO BUFFER
                .
        2039    RDREC   LDX     ZERO            041030
        203C            LDA     ZERO            001030
        203F    RLOOP   TD      INPUT           E0205D
        2042            JEQ     RLOOP           30203F
        2045            RD      INPUT           D8205D
        2048            COMP    ZERO            281030
        204B            JEQ     EXIT            302057
        204E            STCH    BUFFER,X        549039
        2051            TIX     MAXLEN          2C205E
        2054            JLT     RLOOP           38203F
        2057    EXIT    STX     LENGTH          101036
        205A            RSUB                    4C0000
        205D    INPUT   BYTE    X'F1'           F1
        205E    MAXLEN  WORD    4096            001000
```

Object Code of SIC Program (Fig 2.2)

```
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
```

Object Program Format

- Contains three of records
  - Header
    - contain the program name, starting address, and length
  - Text
    - contain translated instructions and data of the program
  - End
    - mark the end of the object program and specify the address in the program where execution is to begin

Object Program Format

- 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 in Hex (2 column per byte)

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

Object Program (Fig 2.3)

```
HCOPY  00100000107A
 ^     ^     ^

T0010001E1410334820390010362810303010154820613C100300102A0C103900102D
 ^     ^ ^     ^     ^     ^     ^     ^     ^     ^     ^     ^

T00101E150C1036482061081033 4C0000454F46000003000000
 ^     ^ ^     ^     ^     ^     ^     ^     ^

T0020391E041030001030E0205D30203FD8205D2810303020575490392C205E38203F
 ^     ^ ^     ^     ^     ^     ^     ^     ^     ^     ^     ^

T0020571C1010364C0000F1001000041030E02079302064509039DC20792C1036
 ^     ^ ^     ^     ^ ^     ^     ^     ^     ^     ^     ^

T002073073820644C000005
 ^     ^ ^     ^     ^

E001000
 ^
```

Two passes assembler

- **Pass 1** (define symbols)
  1. Assign addresses to all statements (generate LOC)
  2. Save the values (addresses) assigned to all labels for use in Pass 2
  3. Perform some processing of assembler directives (address assignment, such as the length of data areas)

- **Pass 2** (assemble instructions and generate object program)
  1. Assemble instructions (translating operation codes & 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

Assembler Tables and Logic (I)

- 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.
    - FIRST→1000, COPY→1000, …

Assembler Tables and Logic (II)

- Location Counter (LOCCTR)
  - A variable used to help in the assignment of addresses
  - Initialized to the beginning address
    specified in the START statement
  - After each source statement is processed,
    the length of assembled instruction or data area to be generated
    is added to LOCCTR
  - Whenever we reach a label in the source program,
    the current value of LOCCTR gives the address
    to be associated with that label

Assembler Tables and Logic (III)

Operation Code Table (OPTAB)

- Contain the mnemonic operation code & its machine language equivalent.
  - May also contain information about instruction format and length
- Used to look up & validate mnemonic operation codes (Pass 1) and translate them to machine language (Pass 2)
  - In SIC, both processes could be done together
  - In SIC/XE, search OPTAB to find the instruction length (Pass 1), determine instruction format to use in assembling the instruction (Pass 2)
- Usually organized as a hash table & a static table
  - Mnemonic operation code as a key
  - Provide fast retrieval with a minimum searching
  - A static table in most cases
    - ✓ Entries are not normally added to or deleted from it

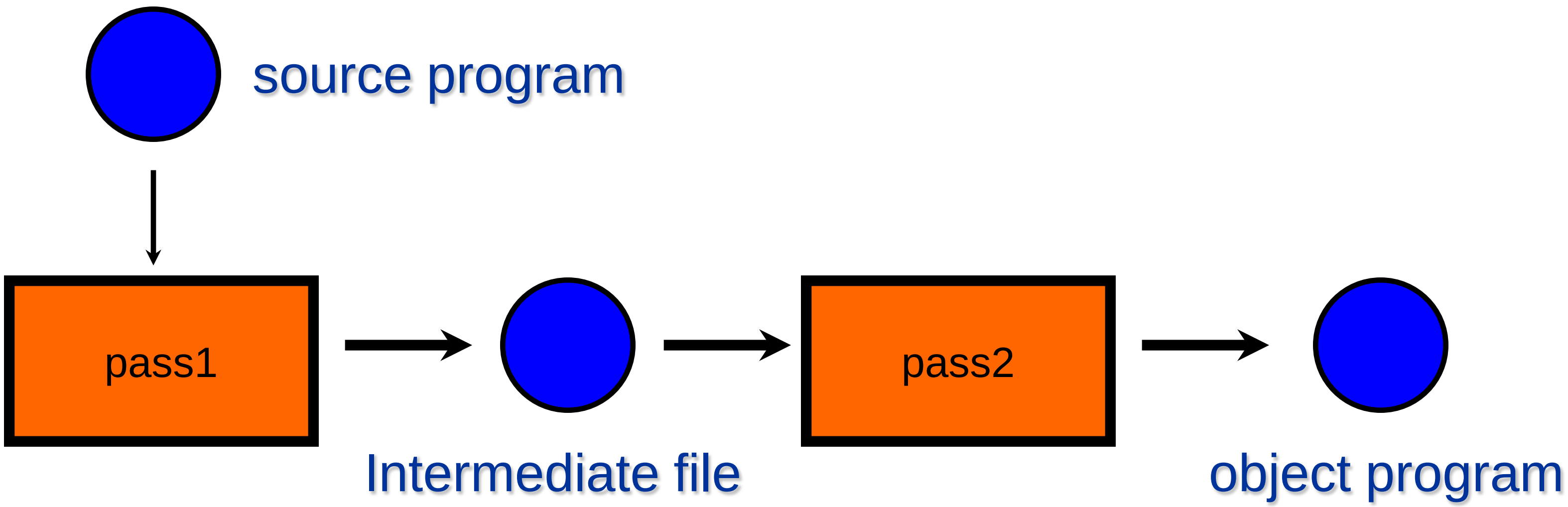
Assembler Tables and Logic (IV)

- Symbol Table (SYMTAB)
  - Used to store values (addresses) assigned to labels
  - Contain the name and value (addresses) for each label, together with flags for error conditions, also information about the data area or instruction labeled
    - for example, its the type or length
  - Pass 1 : labels are entered with their assigned addresses (from LOCCTR)
  - Pass 2 : symbols are used to look up in SYMTAB to obtain the addresses to be inserted
  - Usually organized as a hash table
    - For efficiency of insertion & retrieval
    - Heavily used throughout the assembly

Assembler Tables and Logic (V)

- Intermediate file
  - to communicate between the two passes
  - written by Pass 1, and used as the input to Pass 2
  - containing each source statement
    together with its assigned address, error indicators, etc

Logic Diagram



(containing each source statement together with its assigned address, error indicators, etc.)

Algorithm for Pass 1 (Fig 2.4(a)-I)

```
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}
```

Algorithm for Pass 1 (Fig 2.4(a)-II)

```
                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}
```

Output of Pass 1

- SYMTAB

FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	102A
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039

RDREC	2039
RLOOP	203F
EXIT	2057
INPUT	205D
MAXLEN	205E
WRREC	2061
WLOOP	2064
OUTPUT	2079

Program Length = 207A - 1000 = 107A

Algorithm for Pass 2 (Fig 2.4(b)-I)

```
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
```

Algorithm for Pass 2 (Fig 2.4(b)-II)

```
                              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}
```

Output of Pass 2

```
HCOPY  00100000107A
 ^     ^     ^

T0010001E141033482039001036281030301015482061 3C100300102A0C103900102D
 ^     ^ ^     ^     ^     ^     ^     ^     ^     ^     ^     ^

T00101E150C1036482061081033 4C0000454F46000003000000
 ^     ^ ^     ^     ^     ^     ^     ^     ^

T0020391E041030001030E0205D30203FD8205D2810303020575490392C205E38203F
 ^     ^ ^     ^     ^     ^     ^     ^     ^     ^     ^     ^

T0020571C1010364C0000F1001000041030E0207930206450903 9DC20792C1036
 ^     ^ ^     ^     ^ ^     ^     ^     ^     ^     ^     ^

T002073073820644C000005
 ^     ^ ^     ^     ^

E001000
 ^
```

Assemblers

2.2 Machine-Dependent Assembler Functions

Jongsun Choi
jongsun.choi@ssu.ac.kr

Machine-Dependent Assembler Features

- Consider the design and implementation of an assembler for SIC/XE
- New features
  - Addressing mode
    - Indirect addressing (@)
    - Immediate addressing (#)
    - Base relative addressing (new assembler directive *BASE*)
  - 4-byte extended format (+)
  - Register-register instructions (COMPR A,S)
  - Additional instructions (CLEAR, ...)

Example of a SIC/XE program (Fig 2.5 – I)

```
  5    COPY    START   0              COPY FILE FROM IN 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
```

Example of a SIC/XE program (Fig 2.5 – II)

```
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
```

Example of a SIC/XE program (Fig 2.5 – III)

```
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
```

Object Code of Fig 2.5 (Fig 2.6 - I)

```
  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
```

Object Code of Fig 2.5 (Fig 2.6 - II)

```
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
```

Object Code of Fig 2.5 (Fig 2.6 - III)

```
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
```

Assembler directives

- ```
  5    COPY    START    0
  ```
  - A beginning program address of 0
  - Indicates a relocatable program
- ```
  13   BASE    LENGTH
  ```
  - The base register will contain the address of LENGTH
- ```
       NOBASE
  ```
  - The contents of the base register can no longer be relied upon for addressing
  ```

Register-Register Instructions

- `125    RDREC    CLEAR    X`
  - Add the new mnemonic operation code in OPTAB
  - Add the register names (A, X, etc) and their values (0, 1, etc) in SYMTAB

Register-Memory Instructions

- Most of the register-memory instructions
  - are assembled using
    either program-counter relative
    or base relative addressing
  - Attempt order of translation
    1. Program-counter relative
    2. Base relative addressing
    3. Generate an error message
  - If want to use extended instruction
    - Must specified prefix +

Addressing mode example (I)

- **10 0000 FIRST STL RETADR**
  - OPTAB : **STL 14**
  - SYMTAB : **RETADR 0030** (Pass 1)
  - (SIC) **140030**

Addressing mode example (I) (cont)

- (SIC/XE) PC or base relative addressing mode?
  - PC (program counter) : `0003`
    - ✓ displacement : `0030 - 0003 = 002D`
      - can be stored in 12 bits
    - ✓ PC relative addressing (`p = 1`)
    - ✓ simple addressing (`n = i = 1`)
    - ✓ not indexed addressing (`x = 0`)

op	n	i	x	b	p	e	displacement	
0001 01	1	1	0	0	1	0	0000 0010 1101	17202D

Base-Relative Addressing Mode

- BASE register and directive:

  12 LDB #LENGTH

  13 BASE LENGTH

  - Base register is under the control of programmer
  - BASE directive tells assembler that LENGHTH is base address; NOBASE releases the binding

  160 104E STCH BUFFER, X 57C003

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

  - Displacement= BUFFER–B = 0036–0033 = 3
  - Compare lines 20 and 175 (PC vs Base addressing)

Immediate Address Translation

12 0003 LDB #LENGTH 69202D

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

12 0003 LDB #LENGTH 690033

OPCODE	n	i	x	b	p	e	Address
0110 10	0	1	0	0	0	0	$(033)_{16}$

- The immediate operand is the value of the symbol LENGTH, which is the address assigned to LENGTH
- LENGTH=0033=PC+displacement=0006+02D

Addressing mode example (II)

- **40 0017 J CLOOP**

  - OPTAB : **J 3C**
  - SYMTAB : **CLOOP 0006** (Pass 1)
  - (SIC) **3C0006**
  - (SIC/XE) PC or base relative addressing mode?
    - PC (program counter) : `001A`
      - ✓ displacement : `0006 - 001A = FEC`
        - can be stored in 12 bits
      - ✓ PC relative addressing (`p = 1`)
      - ✓ simple addressing (`n = i = 1`)
      - ✓ not indexed addressing (`x = 0`)

Addressing mode example (III)

- `160    104E     STCH     BUFFER,X`

    - OPTAB : **STCH** **54**
    - SYMTAB : **BUFFER** **0036** (Pass 1)

    - (SIC) **548036**

Addressing mode example (III) (cont)

- (SIC/XE) PC or base relative addressing mode?
  - PC (program counter) : **`1051`**
    - ✓ displacement : **`0036 - 1051 = -101B`**
    - ✓ can not be stored in 12 bits
    - ✓ not PC relative addressing (`p = 0`)
  - B (base) : **`0033`**
    - ✓ displacement : **`0036 - 0033 = 0003`**
    - ✓ can be stored in 12 bits
    - ✓ base relative addressing (b = 1)
    - ✓ simple addressing (n = i = 1)
    - ✓ indexed addressing (x = 1)

op	n	i	x	b	p	e	displacement
0101 01	1	1	1	1	0	0	0000 0000 0011

Addressing mode example (IV)

- **175 1056 EXIT STX LENGTH**

  - OPTAB : **STX 10**
  - SYMTAB : **LENGTH 0033** (Pass 1)
  - (SIC) **100033**

Addressing mode example (IV) (cont)

- (SIC/XE) PC or base relative addressing mode?
  - PC (program counter) : **`1059`**
    - ✓ displacement : `0033 - 1059 = -1026 = EFDA`
      - can not be stored in 12 bits
    - ✓ not PC relative addressing (`p = 0`)
  - B (base) : **`0033`**
    - ✓ displacement : `0033-0033 = 0000`
      - can be stored in 12 bits
    - ✓ base relative addressing (`b = 1`)
    - ✓ simple addressing (`n = i = 1`)
    - ✓ not indexed addressing (`x = 0`)

op	n	i	x	b	p	e	displacement
0001 00	1	1	0	1	0	0	0000 0000 0000

Addressing mode example (V)

- **55   0020   LDA   #3**

  - OPTAB : **LDA   00**
  - SYMTAB :
  - (SIC) ??????
  - (SIC/XE)
    - displacement → 003
    - immediate addressing (n = 0, i = 1)
    - not indexed addressing (x = 0)

Addressing mode example (VI)

- **`133   103C   +LDT   #4096   75101000`**

  - OPTAB : **LDT 74**
  - SYMTAB :
  - (SIC) ??????
  - (SIC/XE)
    - displacement → 1000
      - can be stored in 20 bits
    - extended format (e = 1, b = p = 0)
    - Addressing (n = 0, i = 1)
    - not indexed addressing (x = 0)

Addressing mode example (VII)

- **12 0003 LDB #LENGTH 69202D**

- OPTAB : **LDB** **68**
- SYMTAB : **LENGTH** **0033** (Pass 1)

- (SIC) ??????

- (SIC/XE) PC or base relative addressing mode ?
  - PC (program counter) : `0006`
  - displacement : `0033 - 0006 = 002D`
    - ✓ can be stored in 12 bits
    - ✓ PC relative addressing (`p = 1`)
  - immediate addressing (`n = 0, i = 1`)
  - not indexed addressing (`x = 0`)

Addressing mode example (VIII)

- `70   002A      J     @RETADR      3E2003`

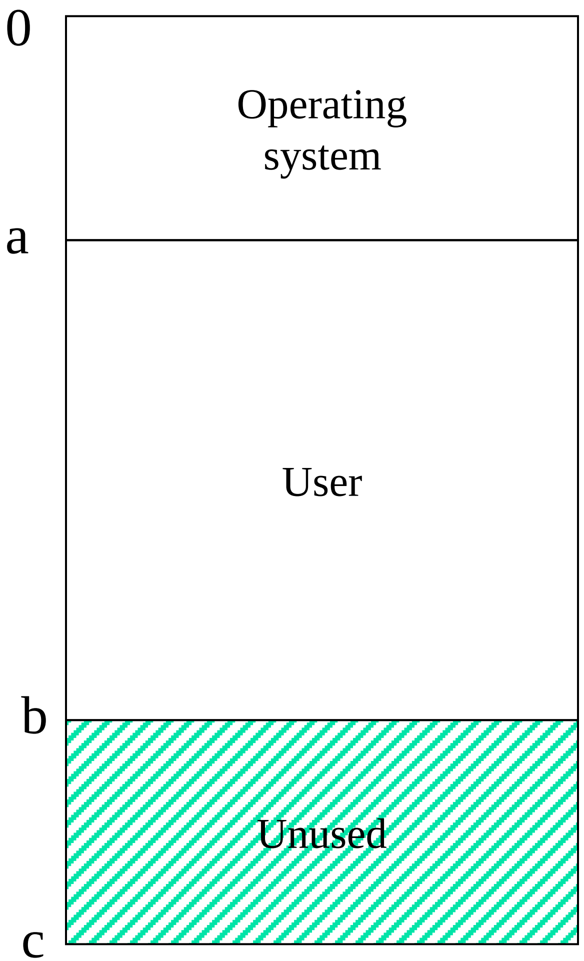
  - OPTAB : **J** **3C**
  - SYMTAB : **RETADR** **0030** (Pass 1)

  - (SIC) ??????

  - (SIC/XE) PC or base relative addressing mode ?
    - PC(program counter) : `002D`
    - displacement : `0030 - 002D = 0003`
      - ✓ can be stored in 12 bits
      - ✓ PC relative addressing (`p = 1`)
    - indirect addressing (`n = 1, i = 0`)
    - not indexed addressing (`x = 0`)

Program Relocation

- Single user contiguous storage allocation
  - earliest computer system
    allowed only a single person
    at a time to use the system
- Memory map



Multiprogramming

- Several jobs are in main memory at once
- The both input/output and CPU calculations can occur simultaneously
- Increases CPU utilization and system throughput

Fixed Partition Multiprogramming

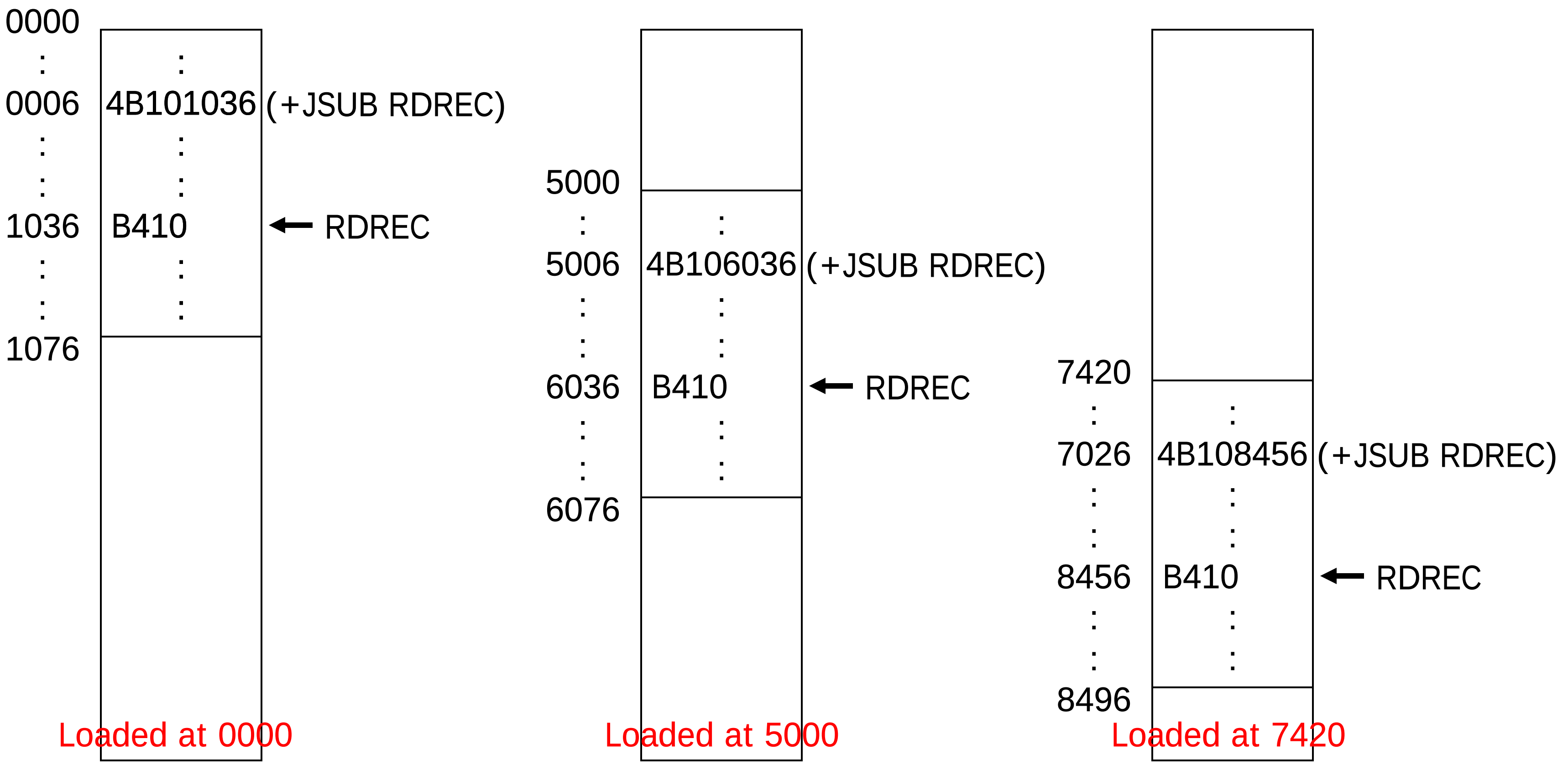
- Absolute Translation and Loading
  - Main memory was divided into a number of fixed-size partitions
  - Each partition could hold a single job
  - Jobs were translated with absolute assembler to run only in a specific partition

Absolute Program (of Fig 2.2)

- `55 101B LDA THREE 00102D`
  - This program must be loaded at address 1000
  - 00102D: Load the content of memory address 102D to register A
  - Suppose the program is loaded at address 5000 instead of address 102D
    - we need to make some change in the address portion of this instruction

Example of Program Relocation (1/3)

- `15   0006 CLOOP   +JSUB   RDREC   4B101036`



Example of Program Relocation (2/3)

- Example Fig. 2.2
  - *__Absolute program__*, starting address ~~1000~~ → 2000

1000	COPY	START	~~1000~~ → 2000	
1000	FIRST	STL	RETADR	141033
1003	CLOOP	JSUB	RDREC	482039
1006		LDA	LENGTH	001036
1009		COMP	ZERO	281030
100C		JEQ	ENDFIL	301015
100F		JSUB	WREC	482061
1012		J	CLOOP	3C1003
1015	ENDFIL	LDA	EOF	00102A
1018		STA	BUFFER	0C1039
101B		LDA	THREE	00102D
101E		STA	LENGTH	0C1036
1021		JSUB	WREC	482061
1024		LDL	RETADR	081033
1027		RSUB		4C0000
102A	EOF	BYTE	C'EOF'	454E46
102D	THREE	WORD	3	000003
1030	ZERO	WORD	0	000000
1033	RETADR	RESW	1	
1036	LENGTH	RESW	1	
1039	BUFFER	RESB	4096	

Example of Program Relocation (3/3)

- Example Fig. 2.6:
  - Except for absolute address, rest of the instructions need not be modified
    - not a memory address (immediate addressing)
    - PC-relative, Base-relative
  - Parts requiring modification at load time are those with absolute addresses

5	0000	COPY	START	~~0~~ → 1000	
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	0036	BUFFER	RESB	4096	

How to change the address part of direct addressing mode instruction ?

- `15 0006   CLOOP   +JSUB   RDREC   4B101036`
    - RDREC is always 1036 bytes past the starting address of the program

- Solutions
    1. When the assembler generates object code for the JSUB instruction, it will insert the address of RDREC relative to the start of the program
    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

    - Modification Record

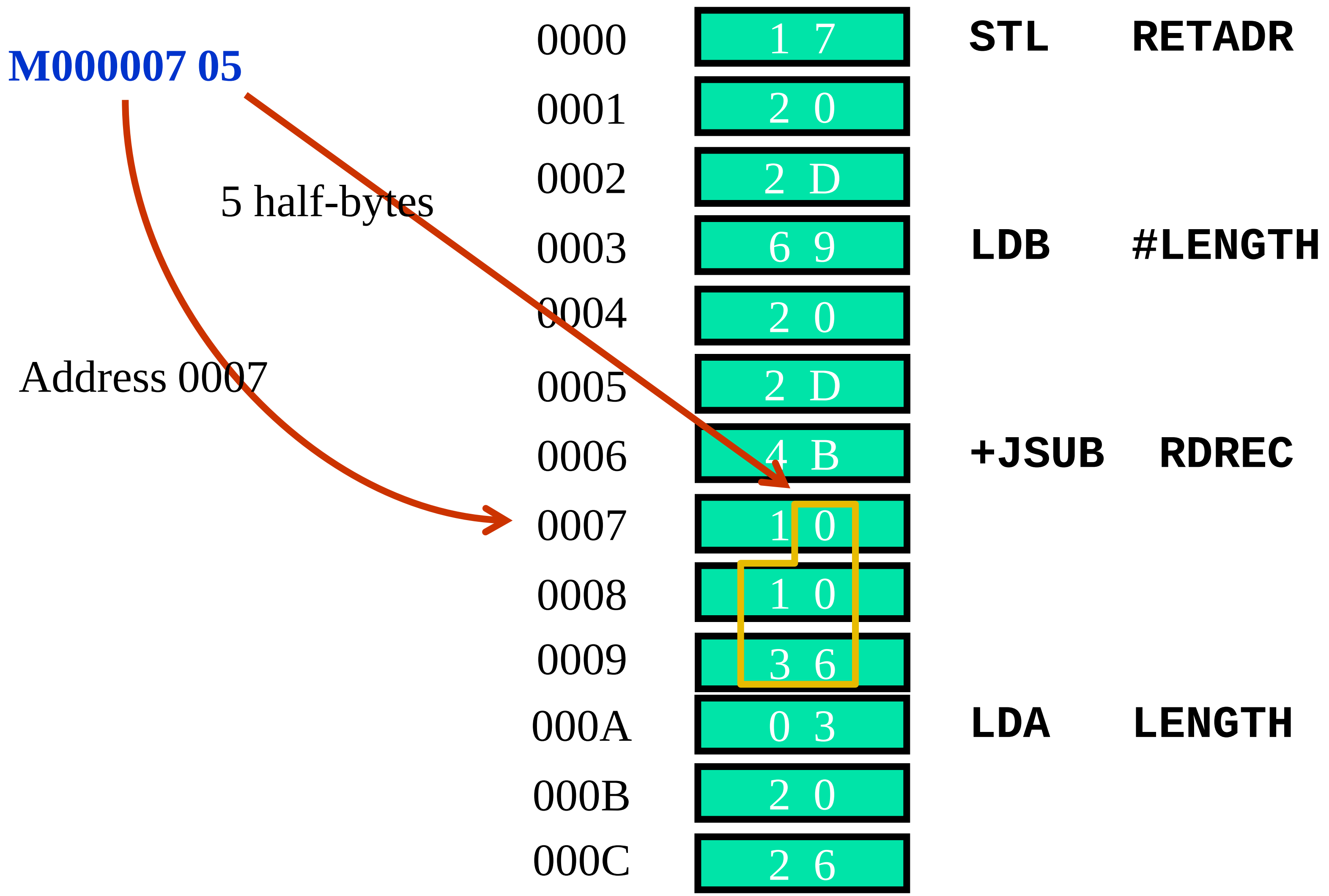
Modification Record

- Modification Record
  - an object program that contains the information (modification record) necessary to perform instruction modification
  - the information is produced by assembler
    - The command for the loader must be a part of the object program

- Format
  - Col. 1 M
  - Col. 2~7 Starting location of the address field to be modified, relative to the beginning of the program (hexadecimal)
  - Col. 8~9 Length of the address field to be modified, in half byte

    if the value of this field it is odd number,
    it is assumed to begin in the middle of the first byte at the starting location

Modification Record



Object Program (Fig 2.8)

```
HCOPY  000000001077
 ^     ^     ^

T0000001D17202D69202D4B1010360320262900003320074B10105D3F2FEC032010
 ^     ^ ^     ^     ^       ^     ^     ^     ^       ^     ^

T00001D130F20160100030F200D4B10105D3E2003454F46
 ^     ^ ^     ^     ^     ^       ^     ^

T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
 ^     ^ ^   ^   ^   ^       ^     ^     ^     ^   ^     ^     ^

T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
 ^     ^ ^     ^     ^     ^ ^   ^     ^     ^     ^     ^     ^

T001070073B2FEF4F000005
 ^     ^ ^     ^     ^

M00000705
 ^     ^

M00001405
 ^     ^

M00002705
 ^     ^

E000000
 ^
```

Example

```
12    0003            LDB    #LENGTH        69202D
13                    BASE    LENGTH
15    0006    CLOOP  +JSUB    RDREC         4B101036
20    000A            LDA     LENGTH        032026
25    000D            COMP   #0             290000
```

- M00000705
  - the beginning address of the program is to be added to a field that begins at address 000007 and is 5 (odd) half-bytes in length
  - 4B101036 → 4B1 01036
    - 4B1 + 01036 (beginning address of the program)

Assemblers

2.3 Machine-Independent Assembler Features

Jongsun Choi
jongsun.choi@ssu.ac.kr

Machine-Independent Assembler Features

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

Literals

- Literals
  - Write the value of a constant operand as part of the instruction that uses it
  - Why use?
    - This avoids having to define the constant elsewhere in the program and make up a label for it
  - ‘literally’ in the instruction
- Usage
  - Identified with the prefix ‘=’

Ex)

```
ENDFIL      LDA    =C'EOF'
WLOOP       TD     =X'05'
```

Example program (Fig 2.9 – I)

```
  5   COPY    START   0               COPY FILE FROM IN 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    =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
```

Example program (Fig 2.9 – II)

```
.
.         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
```

Example program (Fig 2.9 – III)

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

Object Code of Fig 2.9 (Fig 2.10 - I)

```
  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    =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
```

Object Code of Fig 2.9 (Fig 2.10 - II)

```
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
```

Object Code of Fig 2.9 (Fig 2.10 - III)

```
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

- Literal vs. Immediate operand
  - Immediate operand
    - The operand value is assembled as part of the machine instruction

```
55 0020                LDA    #3          010003
```

  - Literal
    - The assembler generates the specified value as a constant at some other memory location
    - The address of the generated constant is used as the target address for the machine instruction

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

  - Compare (Fig. 2.6)

```
45 001A  ENDFIL     LDA      EOF         032010
80 002D  EOF        BYTE     C'EOF'      454F46
```

Literals

- Literal pools
  - All of the literal operands used in a program are gathered together into one or more places
  - Normally be placed at the end of the program
  - Include the assigned addresses & the generated data values
  - LTORG
    - When assembler encounters a LTORG statement, it creates a literal pool that contains all of the literal operands used since the previous LTORG
    - If not using 'LTORG in line 93?
      - ✓ position & distance
    - Reason: keep literal operand close to the instruction

Literals

- Duplicate literals

```
   215      1062   WLOOP TD      =X'05'
   230      106B         WD      =X'05'
```

- Assembler should recognize duplicate literals and store only one copy of specified data value
  - Easiest way to recognize
    : By comparing defining character string, e.g. =X'05'
  - Sometimes a slight additional saving is possible if we look at the generated data value instead of the defining expression.
  - By comparing the generated data value, e.g. =C'EOF' and =X'454F46', but benefits are usually not great enough to justify the additional complexity in the assembler

Literals - Implementation

- Use LITTAB
  - Contains the literal name, the operand value and length, and the address assigned to the operand (operand address)
- Pass 1
  - Build LITTAB with literal name, operand value/length
  - Searches LITTAB for the specified literal name (or value). If it is not present, the literal is added to LITTAB
  - When Pass 1 encounters a LTORG statement or the end of the program, the assembler makes a scan of the literal table.
    - At this time each literal currently in the table is assigned an address
    - The location counter(LOCCTR) is updated to reflect the number of bytes occupied by each literal

Literals

- Pass 2
  - Operand address for use in generating object code is obtained by searching LITTAB for each literal operand encountered
  - The data are inserted at the appropriate places
    in the object program exactly
    as if these values had been generated
    by BYTE or WORD statements

Symbol-Defining Statements

- EQU (equate)
  - Users can define labels on instructions or data area
    - The value of a label is the address assigned to the statement
  - The programmer can define symbols and specify their values
  - Symbolic names can be used for improved readability in place of numeric values
  - Usage

```
Symbol     EQU     Value
```

    - Symbol is entered into **SYMTAB**
    - **value** can be constants, other symbols, expressions
    - EQU doesn't keep memory area
  - Ex-1)

```
                         +LDT   #4096
                              ⇓
          MAXLEN     EQU    4096
133                  +LDT   #MAXLEN
```

Maximum-length record we could read with subroutine RDREC

Symbol-Defining Statements

- EQU (equate)
  - Ex-2) constants

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

  - Ex-3) Symbols

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

  - Ex-4) Expression

```
MAXLEN  EQU   BUFEND-BUFFER
```

Symbol-Defining Statements

- ORG (origin)
  - Indirectly assign values to symbol
  - Usage

```
ORG        value
```

    - Value
      - ✓ a constant or an expression involving constants & previously defined symbols
  - When assembler encounter this statement, reset its location counter(LOCCTR) to the specified value
  - The ORG statement affects the values of all labels defined until the next ORG

- Ex)

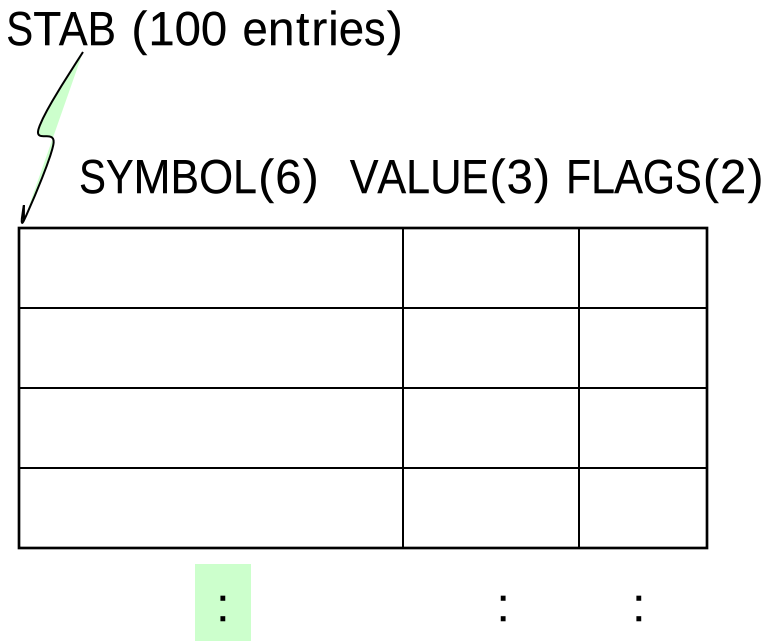
```
STAB      RESB   1100
SYMBOL    EQU    STAB
VALUE     EQU    STAB+6
FLAGS     EQU    STAB+9

          LDA    VALUE,X
```

⇓

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

          LDA    VALUE,X
```



Symbol-Defining Statements

- Restriction
  - All symbols used on the right-hand side of the statement must have been defined previously in the program
  - Ex1)

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

$\Downarrow$

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

Symbol-Defining Statements

- Ex2) **Forward-reference problem of the two-pass assembler**

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

- Ex3)

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

Expressions

- Expressions
  - Must be evaluated by the assembler
    to produce a single operand address or value

- Absolute or relative expressions
  - **BUFEND** and **BUFFER** are relative terms,
    representing addresses relative to program beginning
    → change with program start address
  - But, **BUFEND-BUFFER** has absolute value
    - When relative terms are paired with opposite signs,
      the dependency on starting address is canceled out;
      the result is an absolute value

```
107    MAXLEN        EQU    BUFEND-BUFFER
```

Expressions

- Absolute expression
  - Contain only absolute terms
  - Contain relative terms provided the relative terms occur in pairs and the terms in each such pair have opposite signs
  - None of the relative terms may enter into a multiplication or division operation

  - Ex) R - R + A - R + R ==> (R - R) + A (-R + R)

```
107    MAXLEN        EQU    BUFEND-BUFFER
```

Expressions

- Relative expression
  - One in which all of the relative terms
    excepts one can be paired as described previous,
    and the remaining unpaired relative term must have a positive sign
  - No relative terms may enter into a multiplication or division operation

  - Ex) R - R + A - R + R + R ==> (R - R) + A (- R + R) + R

```
BUFEND - BUFFER + LENGTH
```

  - No relative terms may enter into a multiplication or division operation; the followings are errors:

```
BUFEND+BUFFER, 100-BUFFER, 3*BUFFER
```

Expressions

- SYMTAB

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

R ⇨ Relative symbol
A ⇨ Absolute symbol

Program Blocks

- The source programs logically contained subroutines, data areas, etc
  - Assemblers provide features that allow more flexible handling of the source and object programs

- **Program block**
  - segments of code that are rearranged within a single object program unit

- **Control sections**
  - Segments that are translated into independent object program units

Ex of multiple program blocks (Fig 2.11 – I)

```
  5   COPY    START  0              COPY FILE FROM IN 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
              USE    CBLKS
105   BUFFER  RESB   4096           4096-BYTE BUFFER AREA
106   BUFEND  EQU    *
107   MAXLEN  EQU    BUFEND-BUFFER  MAXIMUM RECORD LENGTH
```

Ex of multiple program blocks (Fig 2.11 – II)

```
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
```

Ex of multiple program blocks (Fig 2.11 – III)

```
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
```

Object Code of Fig 2.11 (Fig 2.12 - I)

```
  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
```

Object Code of Fig 2.11 (Fig 2.12 - II)

```
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            E32019
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
```

Object Code of Fig 2.11 (Fig 2.12 - III)

```
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
```

Program Blocks

- 3 Blocks of Fig 2.11
  - (default) block
    - executable instructions of the program
  - CDATA block
    - all data areas that are a few words or less in length
  - CBLKS block
    - all data areas that consist of larger block of memory
- USE
  - Indicates which portions of the source program belong to the various blocks
  - The assembler will logically rearrange these segments to gather together the pieces of each block

Program Blocks

- 3 Blocks: default, CDATA, CBLKS

Line	Source statement		
5	COPY	START	0
10	FIRST	STL	RETADR
15	CLOOP	JSUB	RDREC
20		LDA	LENGTH
25		COMP	#0
30		JEQ	ENDFIL
35		JSUB	WRREC
40		J	CLOOP
45	FIL	LDA	=C'EOF'
50		STA	BUFFER
55		LDA	#3
60		STA	LENGTH
65		JSUB	WRREC
70		J	@RETADR
92		USE	CDATA
95	RETADR	RESW	1
100	LENGTH	RESW	1
103		USE	CBLKS
105	BUFFER	RESB	4096
106	BUFEND	EQU	*
107	MAXLEN	EQU	BUFEND - BUFFER

3 blocks:
Default (0)
CDATA (1)
CBLKS (2)

(Figure 2.11)

Program Blocks

- Pass 1
  - Accomplishes this logical rearrangement of code
    by maintaining a separate location counter for each program block
    - Each location counter is initialized zero
      when the block is first begun
  - Each location counters are saved and restored
    when switching to another block
  - When labels are entered SYMTAB
    the address is assigned relative to the start of the each block
  - At the end of Pass 1,
    the latest value of the location encounter for each block
    indicates the length of that block

Program Blocks

- Pass 2
  - How to get address of each symbol from SYMTAB?
    - Adds the location of the symbol, relative to the start of its block, to the assigned block stating address

- Ex)

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

Program Blocks

- `20      0006   0        LDA      LENGTH    032060`

  - Start address of CDATA → 0066
  - Relative address of LENGTH → 003 within program block 1

    - TA → 0003 + 0066 = 0069
    - Disp = TA – PC (PC-relative addressing mode)
      = 0069 – 0009 = 60

    - Object code → **032060**

Program Blocks

- Why using program block?
  - Reduced addressing problems
    - No longer need to use extended format instructions
    - The base register is no longer necessary
  - Solve the problem of placement of literals
    - Include a LTORG statement in the CDATA block

Program Blocks (Fig 2.11)

```
HCOPY  000000001071
 ^     ^     ^

T0000001E1720634B20210320602900003320064B203B3F2FEE0320550F2056010003
 ^     ^ ^     ^     ^     ^     ^     ^     ^     ^     ^     ^

T00001E090F20484B20293E203F
 ^     ^ ^     ^     ^

T0000271DB410B400B44075101000E32038332FFADB2032A00433200857A02FB850
 ^     ^ ^   ^   ^   ^       ^     ^     ^     ^   ^     ^     ^

T000044093B2FEA13201F4F0000
 ^     ^ ^     ^     ^

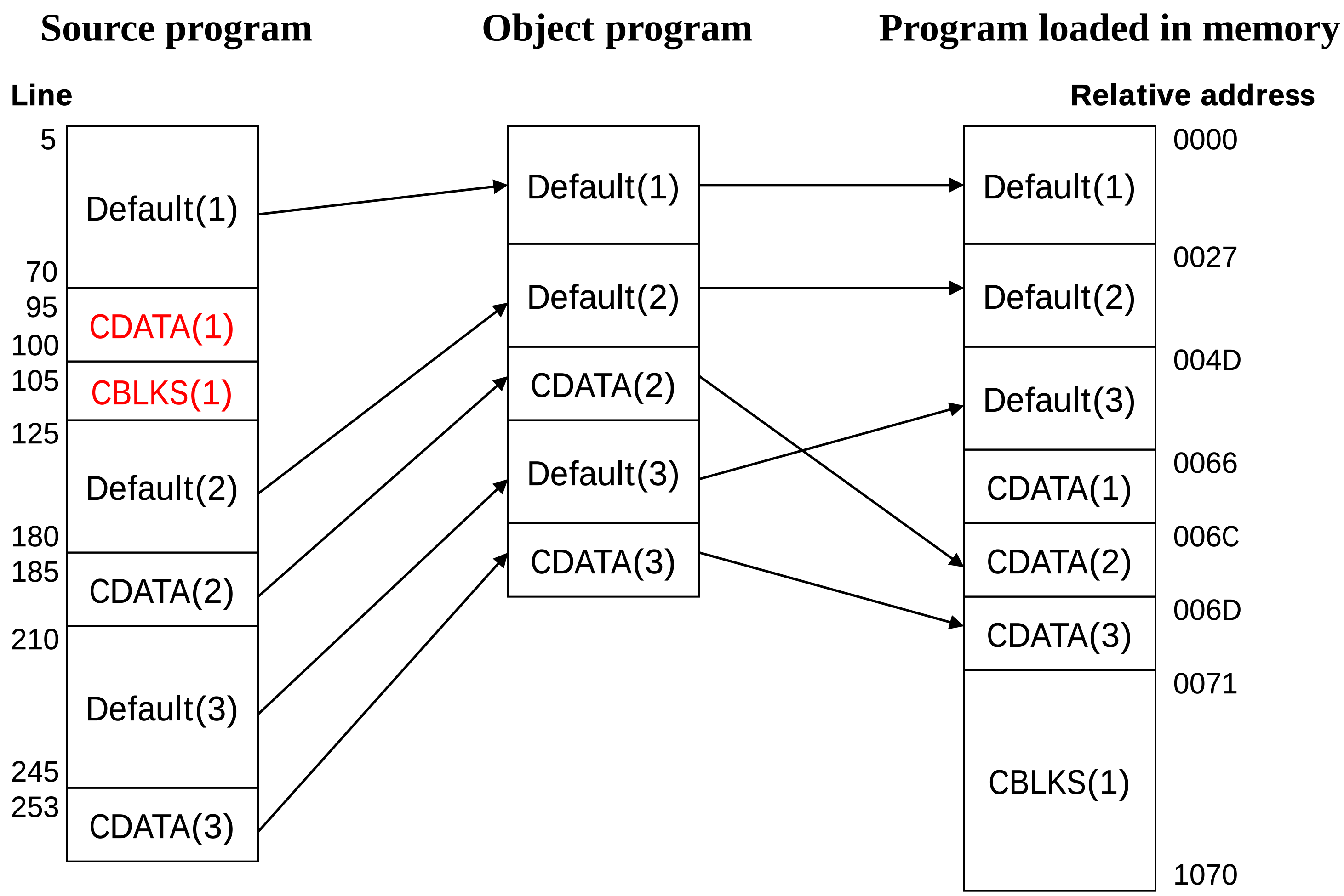
T00006C01F1    0006(LOCCTR) + 0066(Start Address of CDATA)
 ^     ^ ^

T00004D19B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
 ^     ^ ^   ^     ^     ^     ^     ^     ^   ^     ^

T00006D04454F4605  0007(LOCCTR) + 0066(Start Address of CDATA)
 ^     ^ ^     ^

E000000
 ^
```

Program blocks from Fig. 2.11 (Fig 2.14)



CDATA(1), CBLKS(1)

Storage will automatically be reserved for these areas when the program is loaded.

Control Sections and Program Linking

Control Sections

- A part of the program that maintains its identity after assembly
- Each control section can be loaded and relocated independently of the others
- The assembler establishes a separate location counter for each control section

Control Sections and Program Linking

- Directives
  - CSECT
    - Signals the start of a new control section
  - EXTDEF (EXTernal DEFinition)
    - Names external symbol that are defined in this control section and may be used by other sections
  - EXTREF (EXTernal REFerence)
    - Names symbols that are used in this control section and are defined elsewhere

Control Sections and Program Linking (Fig 2.15 – I)

```
  5    COPY    START   0               COPY FILE FROM IN 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   MAXIMUM RECORD LENGTH
```

Control Sections and Program Linking (Fig 2.15 – II)

```
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
```

Object Code of Fig 2.15 (Fig 2.16 - I)

```
  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
```

Object Code of Fig 2.15 (Fig 2.16 - II)

```
109   0000   RDREC   CSECT
110                  .
115                  .        SUBROUTINE TO READ RECORD INTO BUFFER
120                  .
122                          EXTREF   BUFFER,LENGTH,BUFEND
125   0000   RDREC   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
```

Object Code of Fig 2.15 (Fig 2.16 - III)

```
193          WRREC   CSECT
195          .
200          .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205          .
207                  EXTREF  LENGTH,BUFFER
210   0000   WRREC   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
```

Control Sections and Program Linking

- Cf)

15	0003	CLOOP	+JSUB	RDREC	4B100000

- RDREC → External Reference
  - ✓ The assembler has no idea where the CS of RDREC will be loaded
  - ✓ Inserts an address of zero and passes information to the loader

- No predictable operand address
  → therefore relative addressing is not possible

- Must use extended format instruction
  → this is true of any instruction
  whose operand involves an external reference.

Control Sections and Program Linking

- Difference)

```
107  1000  MAXLEN   EQU   BUFEND-BUFFER
190  0028  MAXLEN   WORD  BUFEND-BUFFER  000000
```

- Line 107
  - BUFEND & BUFFER are defined in the same control section
  - The value of the expression can be calculated immediately by the assembler
- Line 190
  - BUFEND & BUFFER are defined in another control section, so values are unknown at assembly time
    - ✓ expression involves external references
  - when the program is loaded, the results is calculated

Control Sections and Program Linking

- Notes
  - The assembler must remember (via entries in SYMTAB)

    in which control section a symbol is defined
  - The assembler must also allow the same symbol to be used in different control sections

Control Sections and Program Linking

- New records for object code
  - Define record
  - Refer record
  - Modification record

- **Define record**
  - Col. 1 D
  - Col. 2-7 Defined external symbol name
  - Col. 8-13 Relative address of symbol within this control section
  - Col. 14-73 Repeat information in Col. 2-13 for other external symbols

New Record types (II)

- Refer record
  - Col. 1 R
  - Col. 2-7 Referred external symbol name
  - Col. 8-73 Names of other external reference symbols

- Modification record *(revised)*
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified
  - Col. 8-9 Length of the field to be modified
  - 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 of Fig 2.15 (Fig 2.17 - I)

```
HCOPY  000000001033
 ^      ^      ^

DBUFFER000033BUFEND001033LENGTH00002D
 ^      ^      ^      ^      ^      ^

RRDREC WRREC
 ^      ^

T0000001D1720274B1000000320232900003320074B1000003F2FEC0320160F2016
 ^      ^ ^      ^        ^      ^      ^      ^        ^      ^      ^

T00001D0D0100030F200A4B1000003E2000
 ^      ^ ^      ^      ^        ^

T00003003454F46
 ^      ^ ^

M00000405+RDREC
 ^      ^ ^

M00001105+WRREC
 ^      ^ ^

M00002405+WRREC
 ^      ^ ^

E000000
 ^
```

Object Program of Fig 2.15 (Fig 2.17 - II)

```
HRDREC 00000000002B
 ^     ^     ^

RBUFFERLENGTHBUFEND
 ^     ^     ^

T0000001DB410B400B44077201FE3201B332FFADB2015A004332009579000000B850
 ^     ^ ^   ^   ^   ^     ^     ^     ^     ^   ^     ^       ^

T00001D0E3B2FE9131000004F0000F1000000
 ^     ^ ^     ^       ^     ^ ^

M00001805+BUFFER
 ^     ^ ^

M00002105+LENGTH
 ^     ^ ^

M00002806+BUFEND
 ^     ^ ^

M00002806-BUFFER
 ^     ^ ^

E
```

Object Program of Fig 2.15 (Fig 2.17-III)

```
HWRREC 00000000001C
 ^     ^     ^

RLENGTHBUFFER
 ^     ^

T0000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
 ^     ^ ^   ^       ^     ^     ^       ^     ^   ^     ^     ^

M00000305+LENGTH
 ^     ^ ^

M00000D05+BUFFER
 ^     ^ ^

E
```

Control Sections and Program Linking

- Ex)

```
M00000405 + RDREC
```

  - The existence of multiple control sections make the handling of expressions slightly more complicated
  - But terms in each pair must be relative within the same control section

Control Sections and Program Linking

- Ex)

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

- BUFEND & BUFFER are defined in another control section
- The assembler generates an initial value of zero for this word
- The last two Modification record
  - ✓ The address of BUFEND be added to this field
  - ✓ The address of BUFFER be subtracted from it
- This computation, performed at load time,
  results in the desired value for the data word
- BUFEND & BUFFER are in the same control section
  - ✓ Their difference is an absolute value

Assemblers

2.4 Assembler Design Options

Jongsun Choi
jongsun.choi@ssu.ac.kr

One-Pass Assemblers

- What is problem?
  - Main problem in trying to assemble a program in one pass involves forward references
  - Many one-pass assemblers prohibit forward references to data items

- Two main types of one-pass assemblers
  - Produce object code directly in memory for immediate execution
  - Produce the usual kind of object program for later execution

One-Pass Assembler

- First type
  - Object code is produced directly in memory for immediate execution
  - No object program is written out, and no loader is needed
  - Called ***Load-and-Go*** assembler
  - Useful in a system that is oriented toward program development & testing
  - Avoid the overhead of an additional pass over the source program

One-Pass Assembler

- Steps
  - Generate object code instructions
    as it scans the source program
  - Generate object code without operand address
    - If an instruction operand is not defined,
      the operand address is omitted when the instruction is assembled
    - Insert the symbol into the SYMTAB with a undefined flag
  - Add the address of the operand to a list of forward references
    associated with the symbol table entry
  - When the definition for a symbol is encountered,
    the Forward reference list for that symbol is scanned,
    and the proper address is inserted into instructions

Sample for a One-pass Assembler (Fig 2.18-I)

```
 0    1000   COPY    START   1000
 1    1000   EOF     BYTE    C'EOF'          454F46
 2    1003   THREE   WORD    3               000003
 3    1006   ZERO    WORD    0               000000
 4    1009   RETADR  RESW    1
 5    100C   LENGTH  RESW    1
 6    100F   BUFFER  RESB    4096
 9           .
10    200F   FIRST   STL     RETADR          141009
15    2012   CLOOP   JSUB    RDREC           48203D
20    2015           LDA     LENGTH          00100C
25    2018           COMP    ZERO            281006
30    201B           JEQ     ENDFIL          302024
35    201E           JSUB    WRREC           482062
40    2021           J       CLOOP           3C2012
45    2024   ENDFIL  LDA     EOF             001000
50    2027           STA     BUFFER          0C100F
55    202A           LDA     THREE           001003
60    202D           STA     LENGTH          0C100C
65    2030           JSUB    WRREC           482062
70    2033           LDL     RETADR          081009
75    2036           RSUB                    4C0000
```

Sample for a One-pass Assembler (Fig 2.18-II)

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

Sample for a One-pass Assembler (Fig 2.18-III)

```
195                 .
200                 .       SUBROUTINE TO WRITE RECORD FROM BUFFER
205                 .
206     2061        OUTPUT  BYTE       X'05'                05
207                 .
210     2062        WRREC   LDX        ZERO                 041006
215     2065        WLOOP   TD         OUTPUT               E02061
220     2068                JEQ        WLOOP                302065
225     206B                LDCH       BUFFER,X             509039
230     206E                WD         OUTPUT               DC2061
235     2071                TIX        LENGTH               2C100C
240     2074                JLT        WLOOP                382065
245     2077                RSUB                            4C0000
255                         END        FIRST
```

Object code & Symbol table entries

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		
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		

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

Object code & Symbol table entries

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		

Symbol	Value	
LENGTH	100C	
RDREC	203D	
THREE	1003	
ZERO	1006	
WRREC	*	→ 201F → 2031 0
EOF	1000	
ENDFIL	2024	
RETADR	1009	
BUFFER	100F	
CLOOP	2012	
FIRST	200F	
MAXLEN	203A	
INPUT	2039	
EXIT	*	→ 2050 0
RLOOP	2043	

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

One-Pass Assembler

- Second type
  - Produces object program as output
  - Used on system
    where external working-storage devices are not available
- Steps
  - Generate object code,
    where the unknown operand address is 0000
  - Forward references are entered into lists as before
  - Generate, for each symbol in the list, text record
    with the correct operand address

Object Program (Fig 2.20)

```
HCOPY  00100000107A
 ^     ^     ^
T00100009454F46000003000000
 ^     ^ ^      ^     ^
T00200F15141009480000000100C2810063000004800003C2012
 ^     ^ ^      ^     ^     ^     ^     ^     ^
T00201C022024
 ^     ^ ^
T00202419001000C100F0010030C100C4800000810094C0000F10010000
 ^     ^ ^      ^     ^     ^     ^     ^     ^     ^ ^
T00201302203D
 ^     ^ ^
T00203D1E041006001006E02039302043D8203928100630000054900F2C203A382043
 ^     ^ ^      ^     ^     ^     ^     ^     ^     ^     ^     ^
T00205002205B
 ^     ^ ^
T00205B0710100C4C000005
 ^     ^ ^      ^     ^
T00201F022062
 ^     ^ ^
T00203102062
 ^     ^ ^
T0020621804100 6E02061302065 50900FDC20612C100C3820654C0000
 ^     ^ ^      ^     ^     ^     ^     ^     ^     ^
E00200F
 ^
```

Object code of lines 10-40

Address of ENDFIL

Multi-Pass Assembler

- Restrictions in EQU and ORG statements
  - Any symbol used in the right-hand side should be defined previously in the source program
  - This restriction is normally applied to all assembler directives
  - Ex)

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

- The general solution
  - Multi-pass assemblers allow forward references in EQU and ORG statements
  - a multi-pass assembler that can make as many passes as are needed to process the definitions of symbols

Multi-Pass Assembler

- Steps
  - Store the label into SYMTAB with some information if it is not in SYMTAB
  - Store the forward reference symbol in SYMTAB if it is not in SYMTAB
  - Enter the label into the list associated with the forward reference symbol
  - Update the information of the symbol in SYMTAB

Multi-Pass Assembler

- Example (1/3)

HALFSZ EQU MAXLEN/2

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

Fig 2.21 (a)

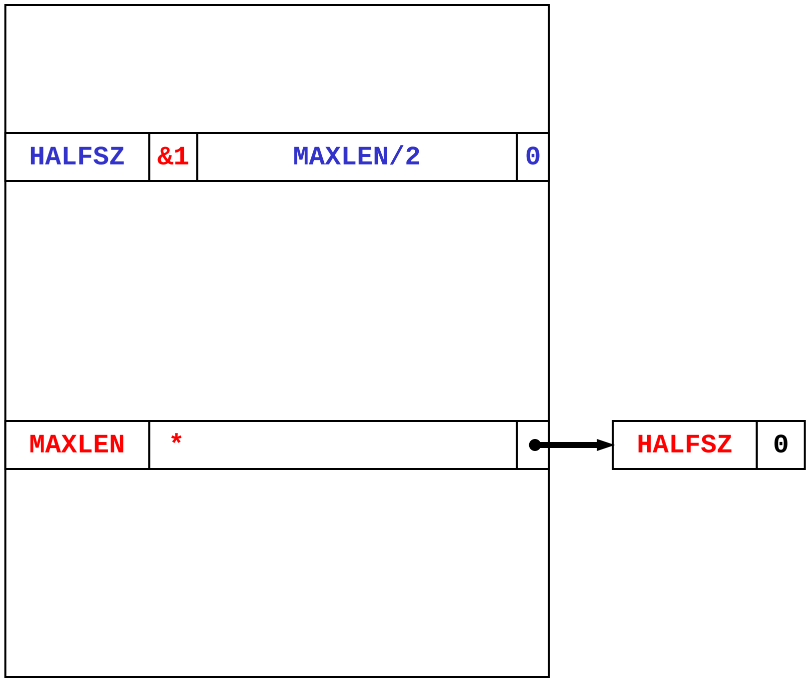


Fig 2.21 (b)

Multi-Pass Assembler

- Example (2/3)

```
MAXLEN   EQU   BUFEND-BUFFER          PREVBT   EQU   BUFFER-1
```

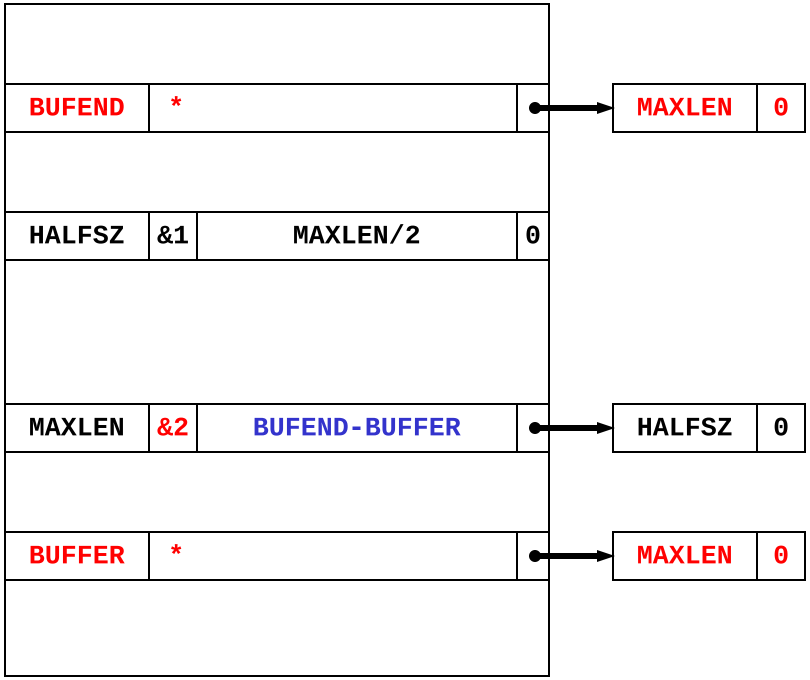


Fig 2.21 (c)

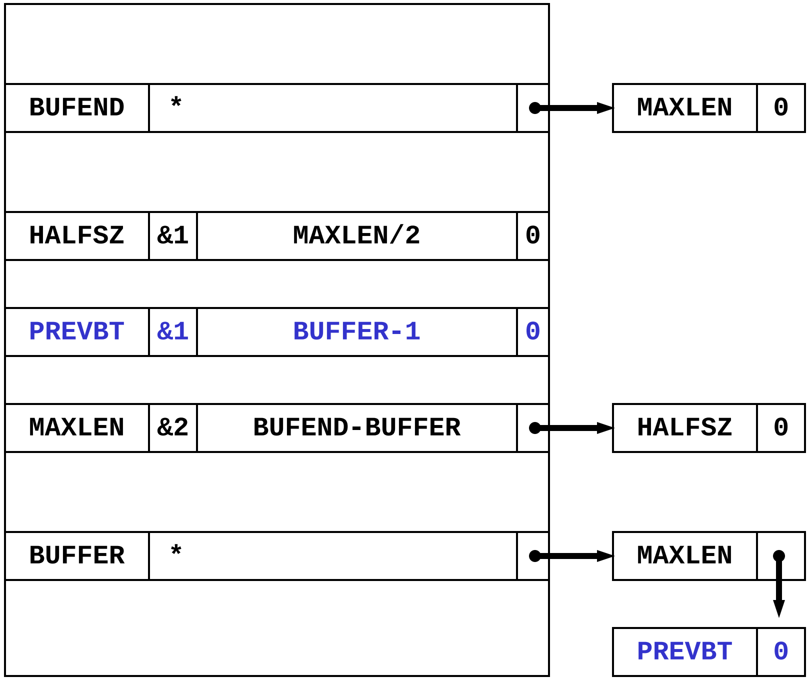


Fig 2.21 (d)

Multi-Pass Assembler

- Example (3/3)

```
BUFFER   RESB    4096                    BUFEND   EQU      *
```

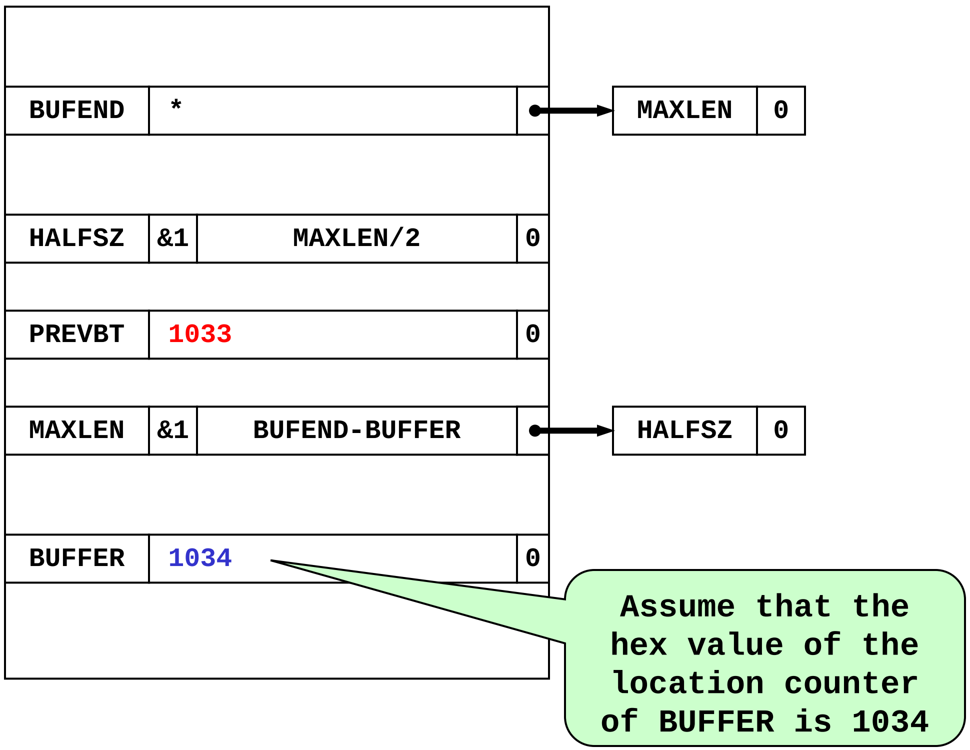


Fig 2.21 (e)

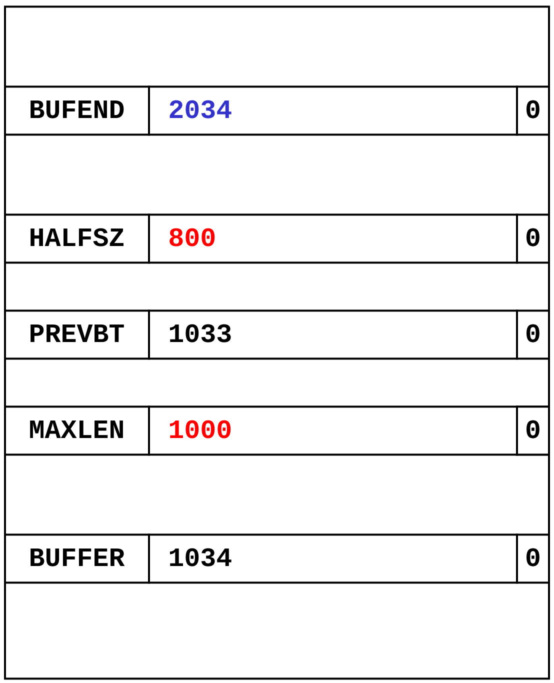


Fig 2.21 (f)