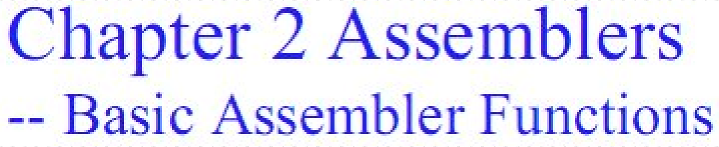
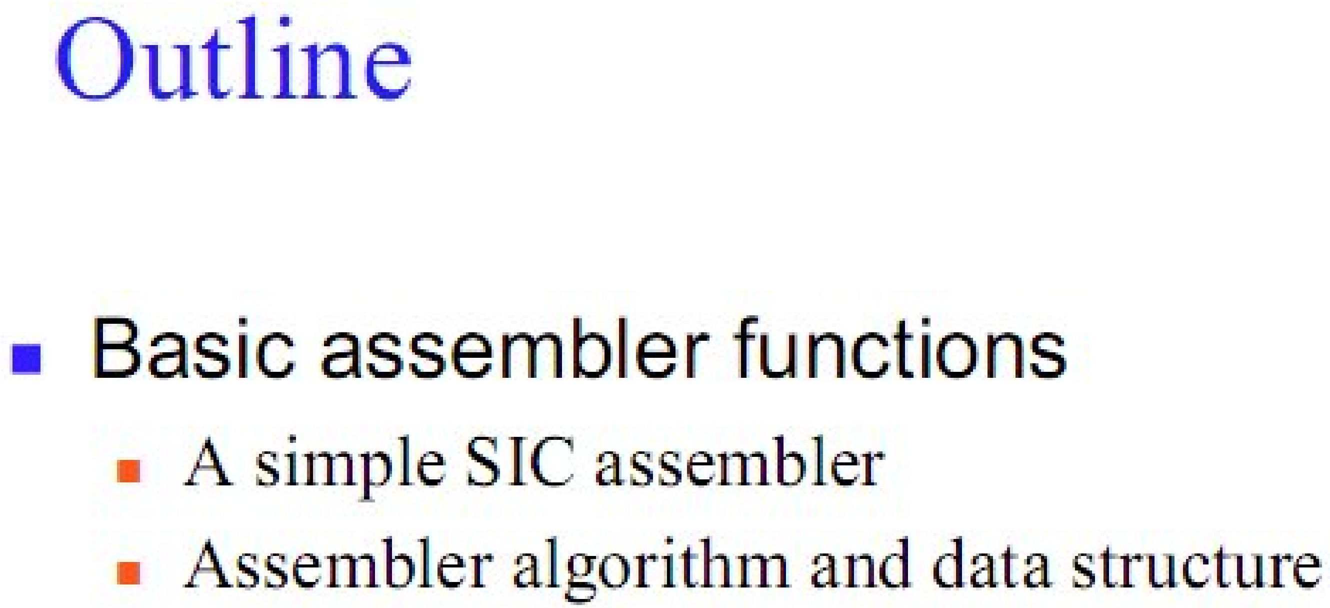
CC410: System Programming
Lecture 4



Chapter 2 Assemblers

-- Basic Assembler Functions

By: Dr. Wael Hosny

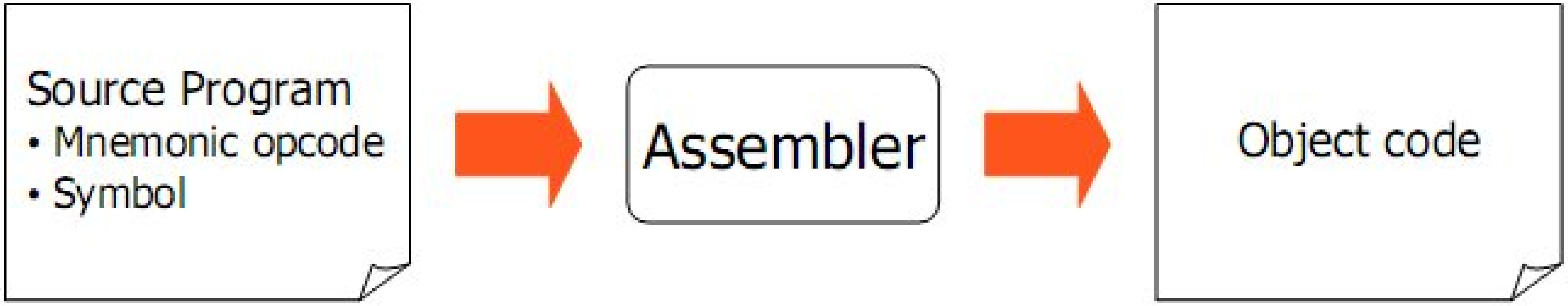


Outline

- Basic assembler functions
  - A simple SIC assembler
  - Assembler algorithm and data structure

Basic assembler functions

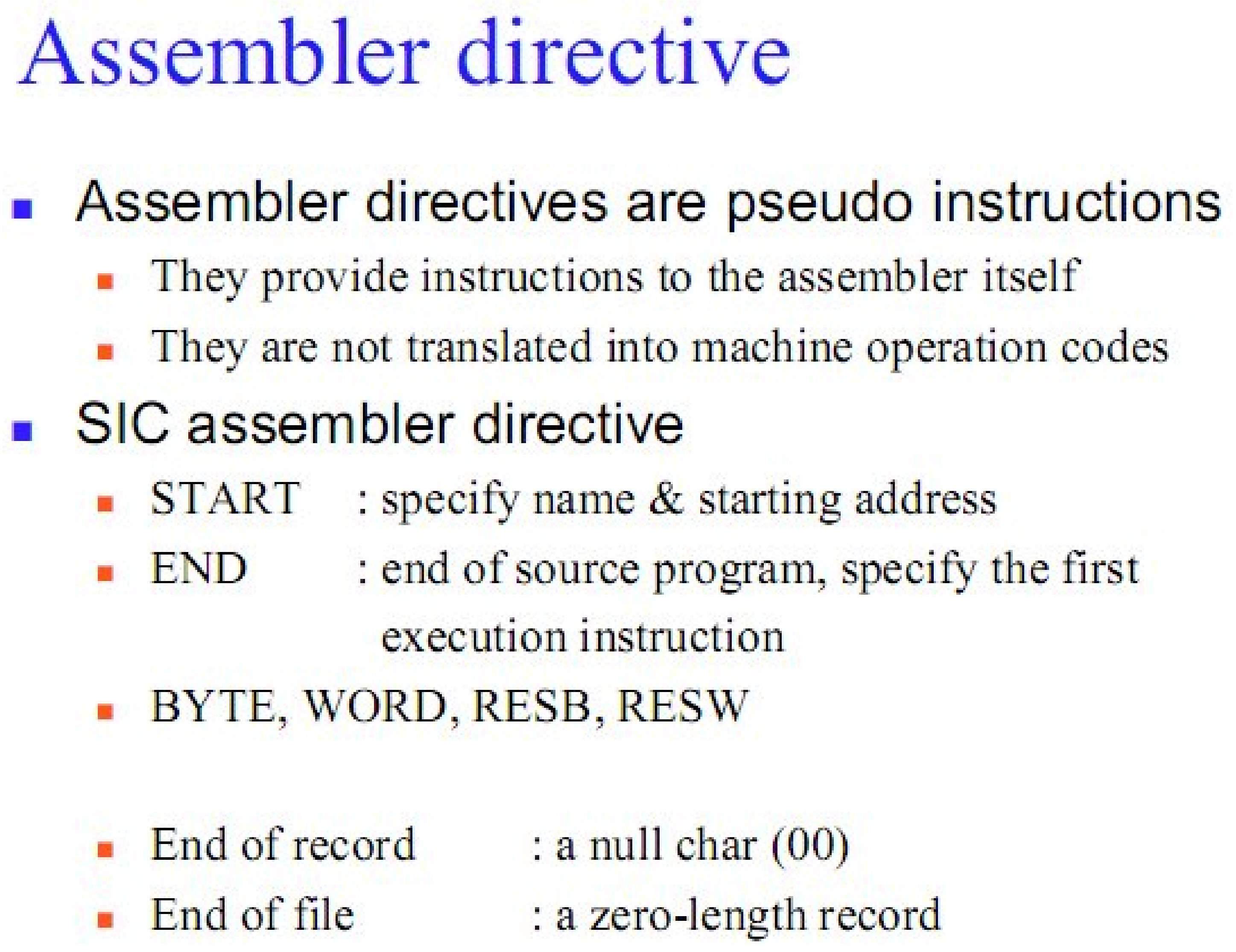
- Translating mnemonic operation codes to their machine language equivalents
- Assigning machine addresses to symbolic labels



Source Program
- Mnemonic opcode
- Symbol

Assembler

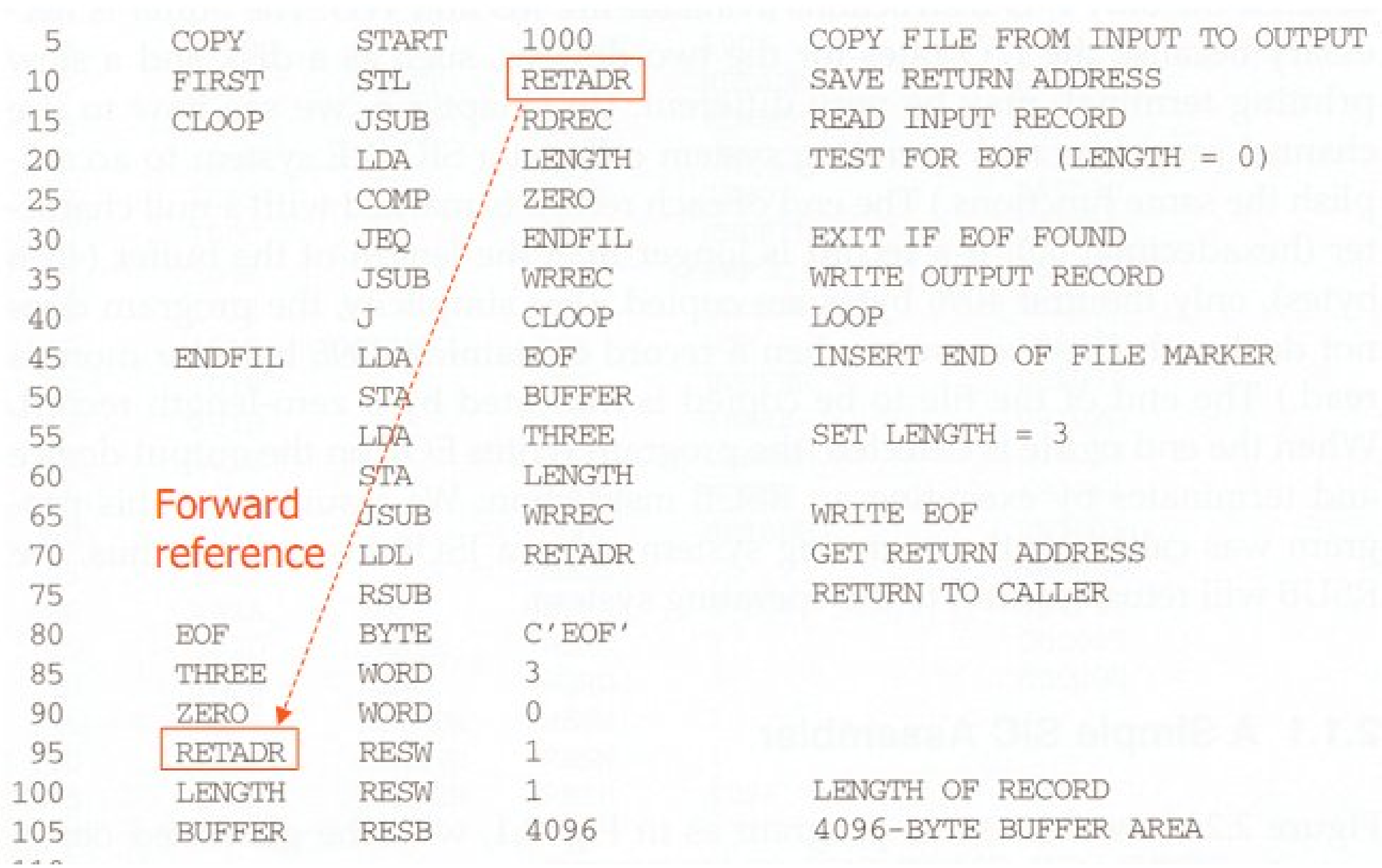
Object code



Assembler directive

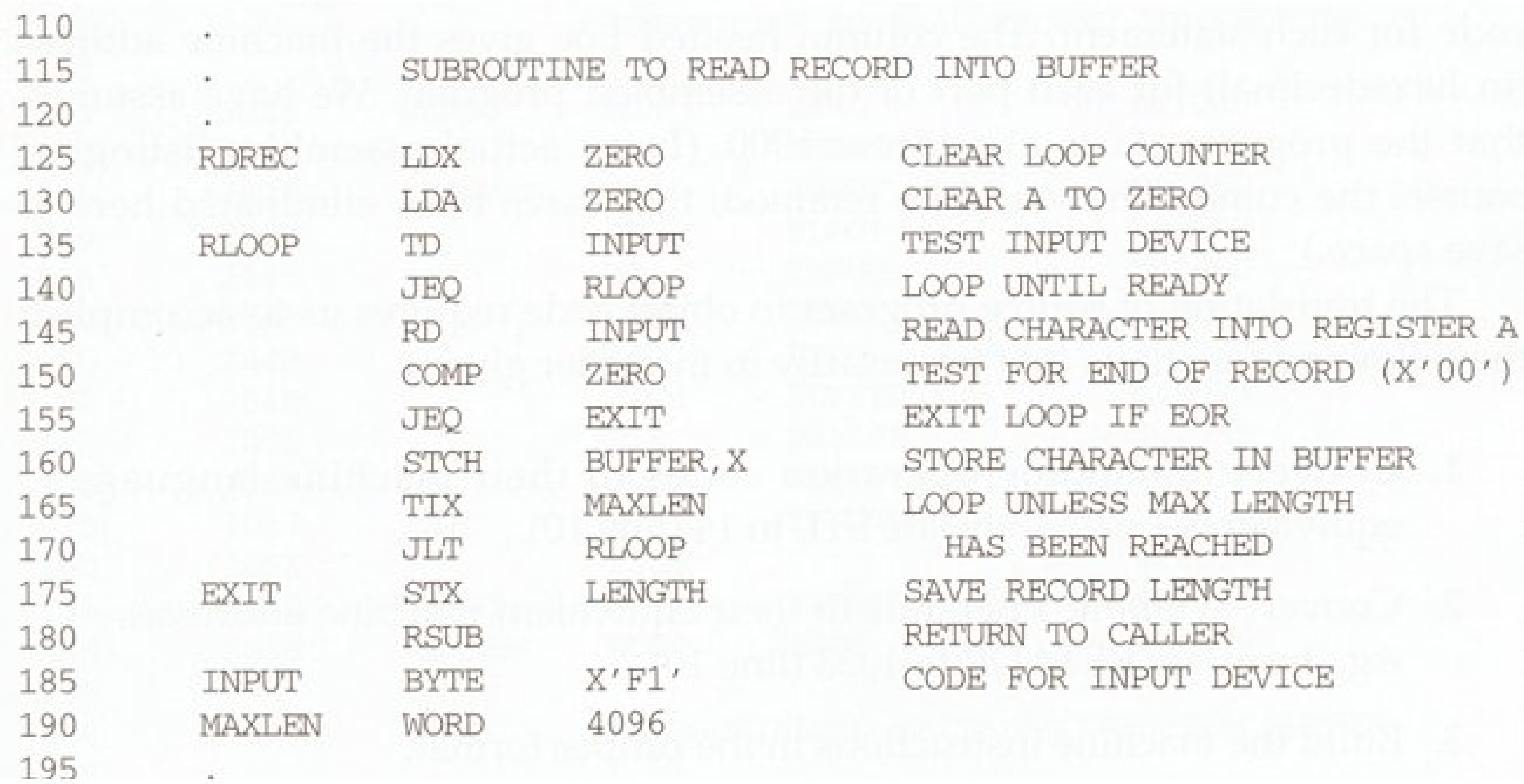
- Assembler directives are pseudo instructions
  - They provide instructions to the assembler itself
  - They are not translated into machine operation codes
- SIC assembler directive
  - START : specify name & starting address
  - END : end of source program, specify the first execution instruction
  - BYTE, WORD, RESB, RESW
  - End of record : a null char (00)
  - End of file : a zero-length record

Example program (Figure 2.1 pp. 45)



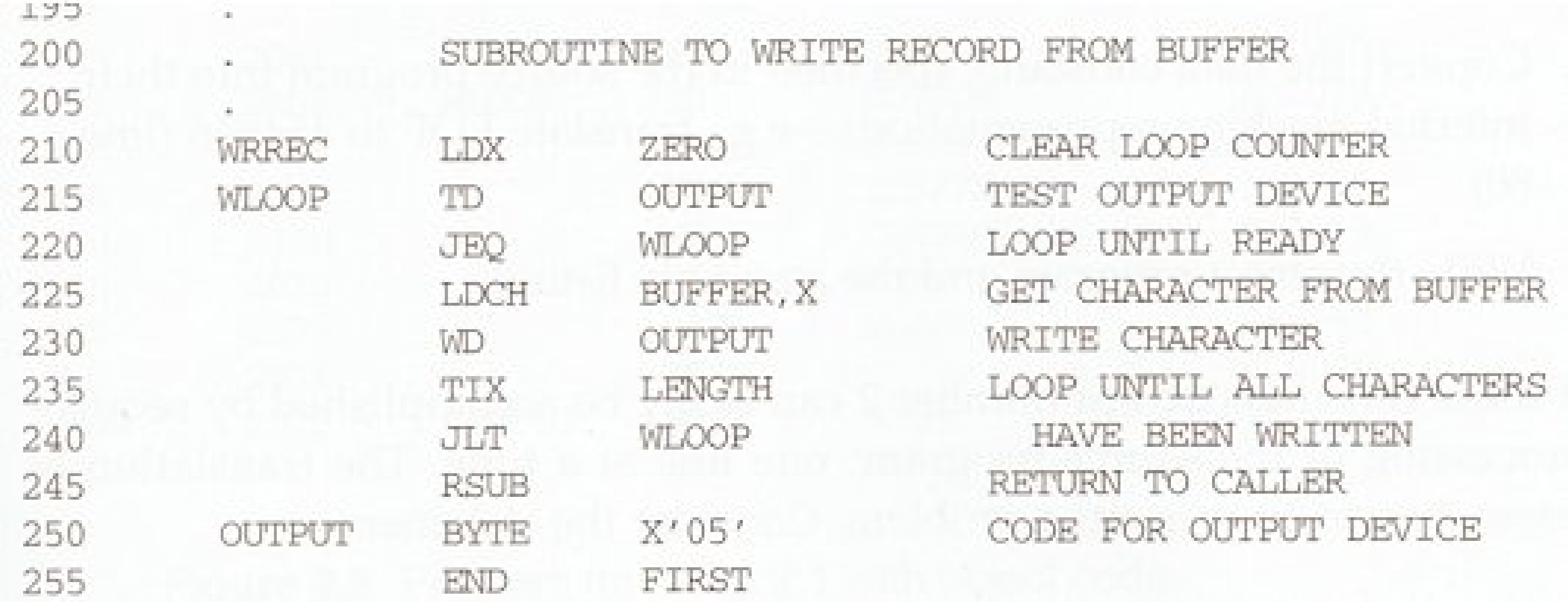
```
  5   COPY     START   1000       COPY FILE FROM INPUT TO OUTPUT
 10   FIRST    STL     RETADR     SAVE RETURN ADDRESS
 15   CLOOP    JSUB    RDREC      READ INPUT RECORD
 20            LDA     LENGTH     TEST FOR EOF (LENGTH = 0)
 25            COMP    ZERO
 30            JEQ     ENDFIL     EXIT 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
```

Forward reference

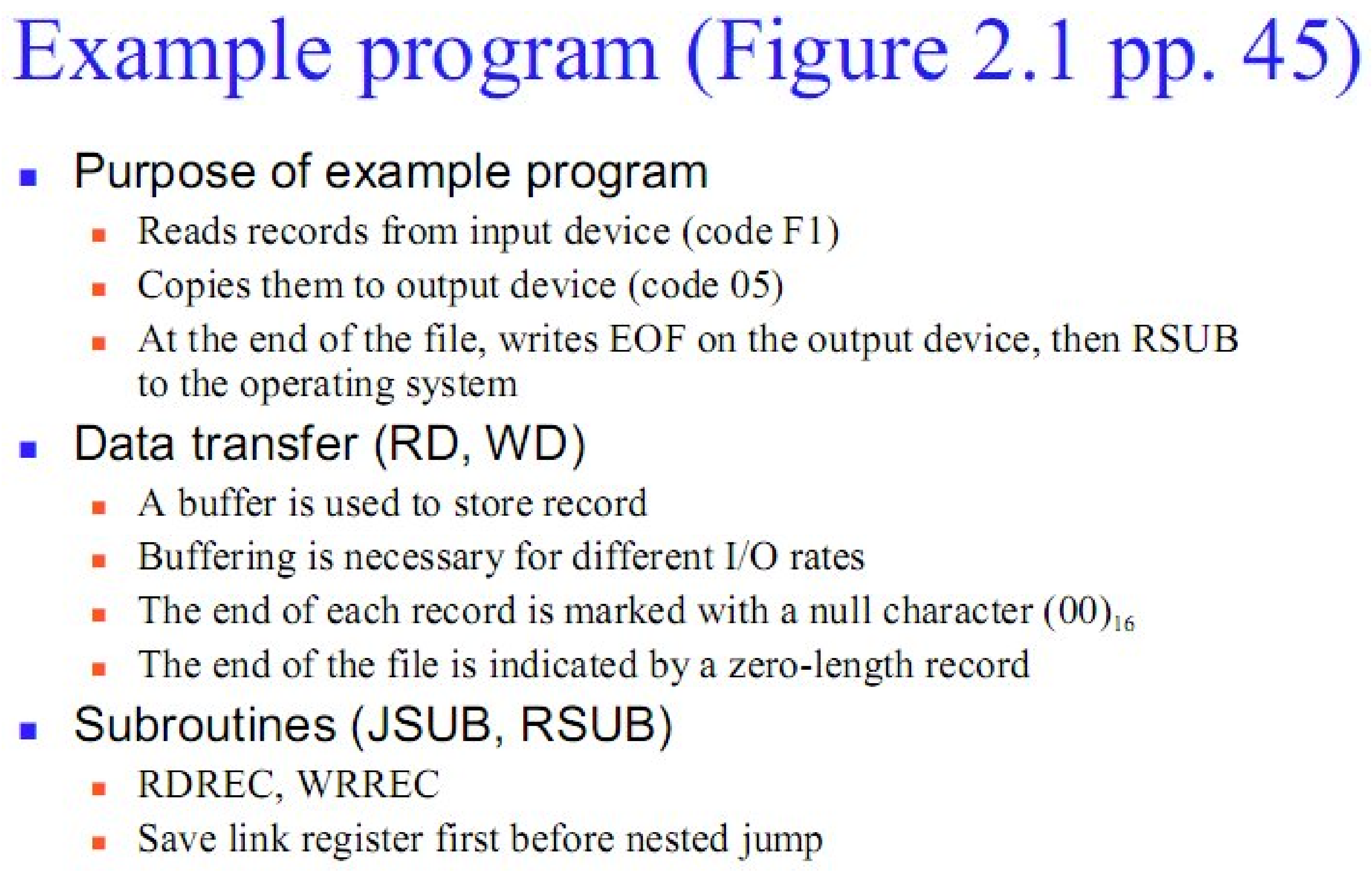


```
        .
        .          SUBROUTINE TO READ RECORD INTO BUFFER
        .
        RDREC      LDX        ZERO                 CLEAR LOOP COUNTER
                   LDA        ZERO                 CLEAR A TO ZERO
        RLOOP      TD         INPUT                TEST INPUT DEVICE
                   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 program (Figure 2.1 pp. 45)

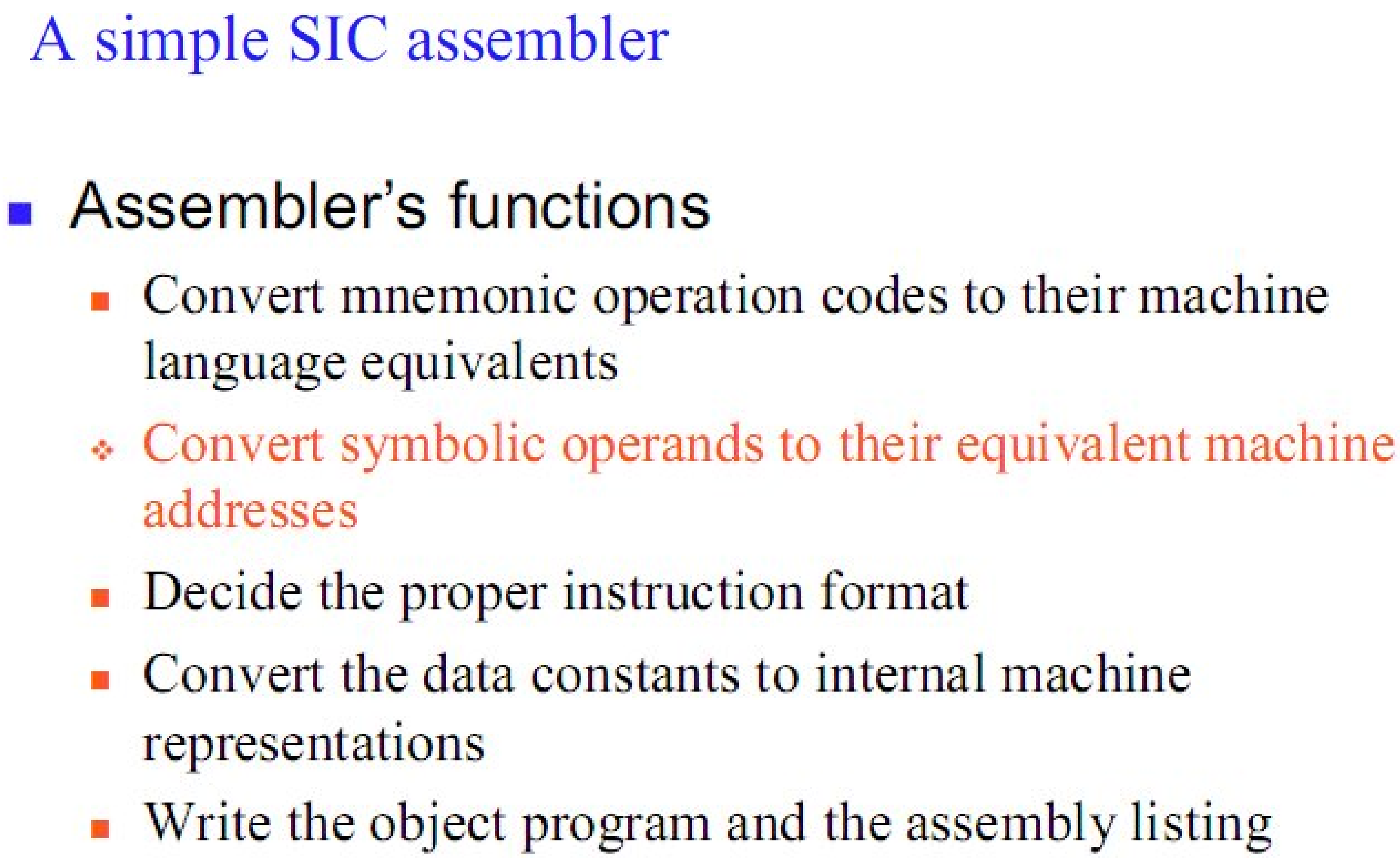


```
.
.          SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC      LDX      ZERO           CLEAR LOOP COUNTER
WLOOP      TD       OUTPUT         TEST OUTPUT DEVICE
           JEQ      WLOOP          LOOP UNTIL READY
           LDCH     BUFFER,X       GET CHARACTER FROM BUFFER
           WD       OUTPUT         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
```



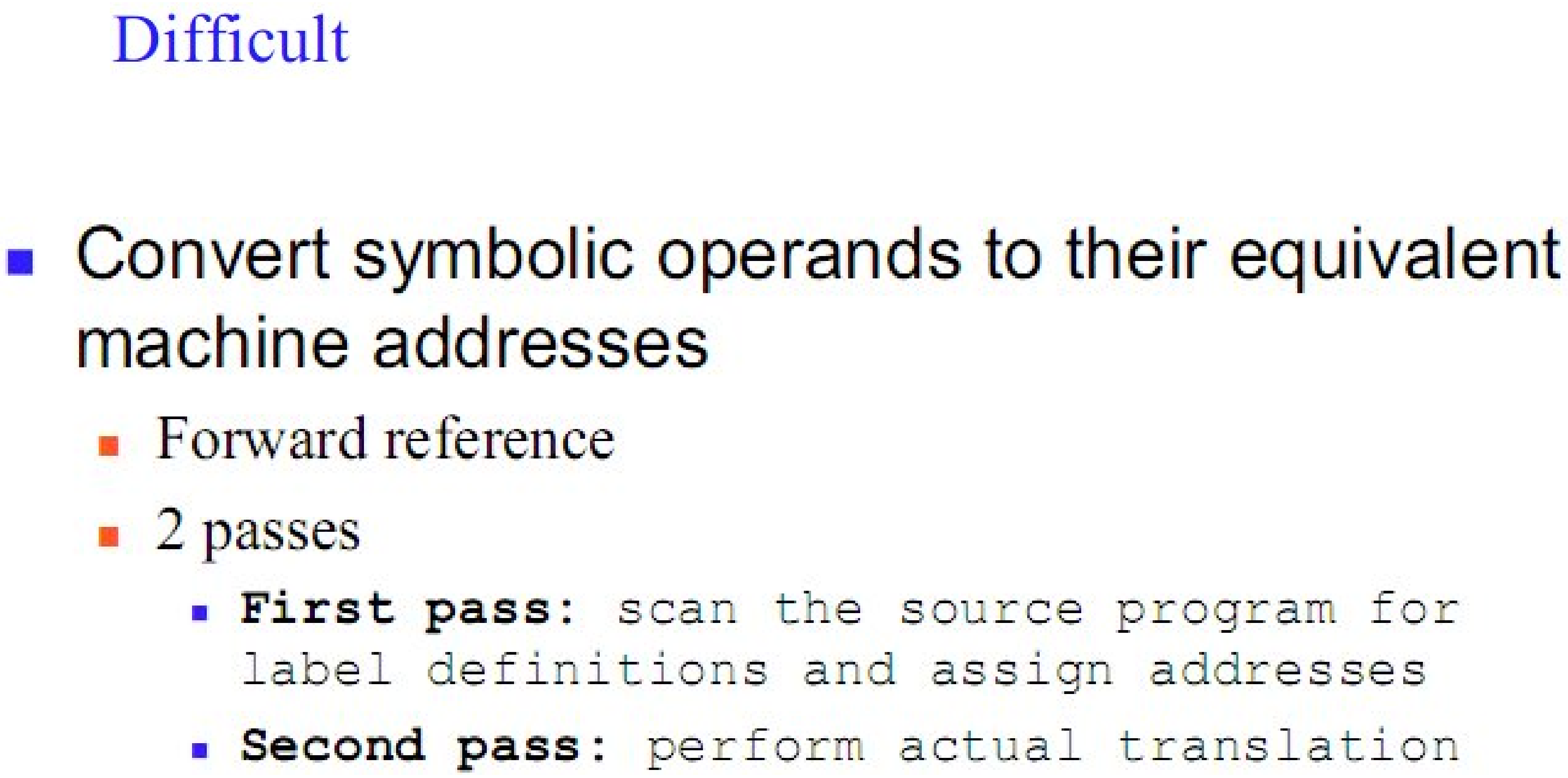
Example program (Figure 2.1 pp. 45)

- Purpose of example program
  - Reads records from input device (code F1)
  - Copies them to output device (code 05)
  - At the end of the file, writes EOF on the output device, then RSUB to the operating system
- 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 $(00)_{16}$
  - The end of the file is indicated by a zero-length record
- Subroutines (JSUB, RSUB)
  - RDREC, WRREC
  - Save link register first before nested jump



A simple SIC assembler

- Assembler's functions
  - Convert mnemonic operation codes to their machine language equivalents
  - Convert symbolic operands to their equivalent machine addresses
  - Decide the proper instruction format
  - Convert the data constants to internal machine representations
  - Write the object program and the assembly listing

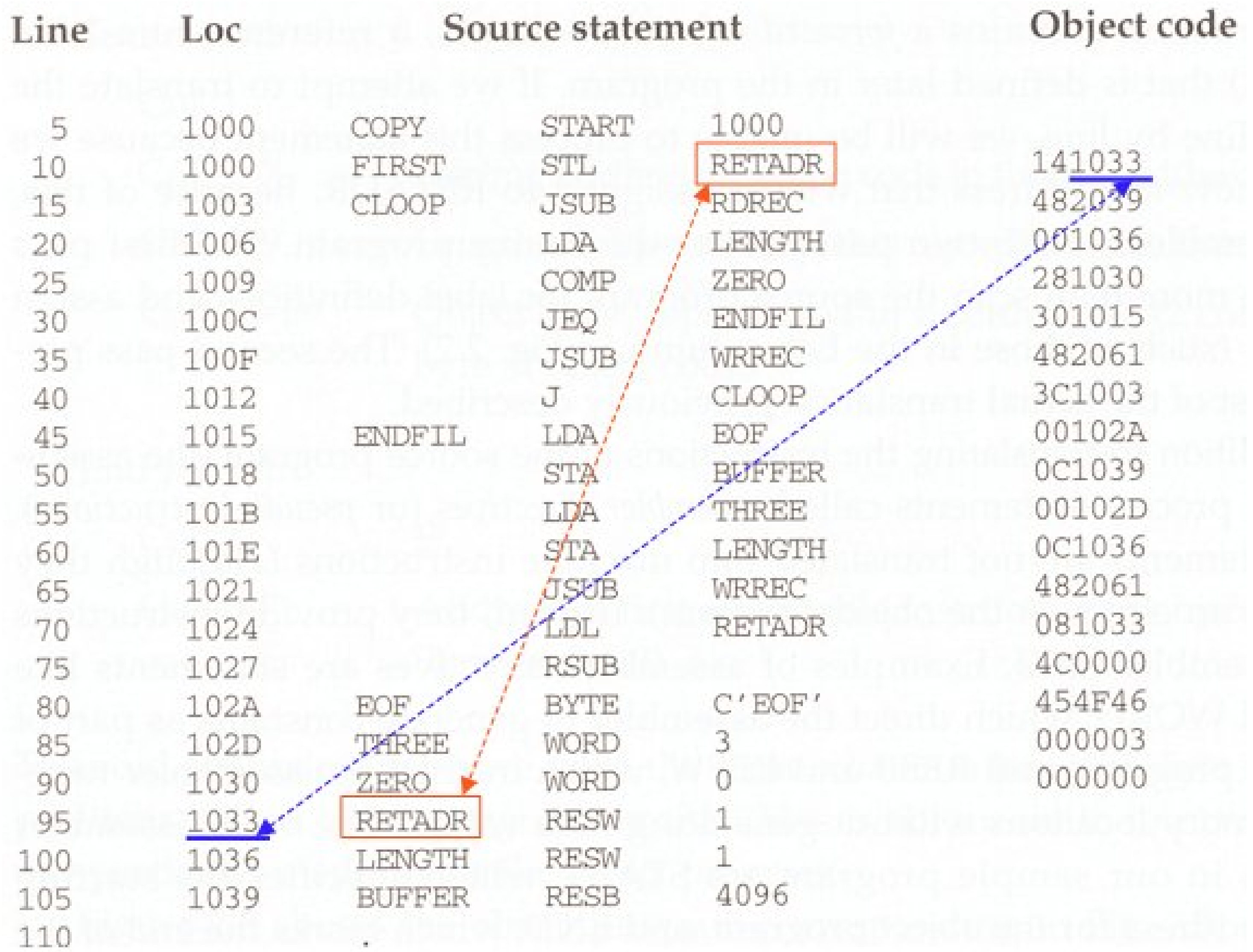


Difficult

- Convert symbolic operands to their equivalent machine addresses
  - Forward reference
  - 2 passes
    - **First pass:** scan the source program for label definitions and assign addresses
    - **Second pass:** perform actual translation

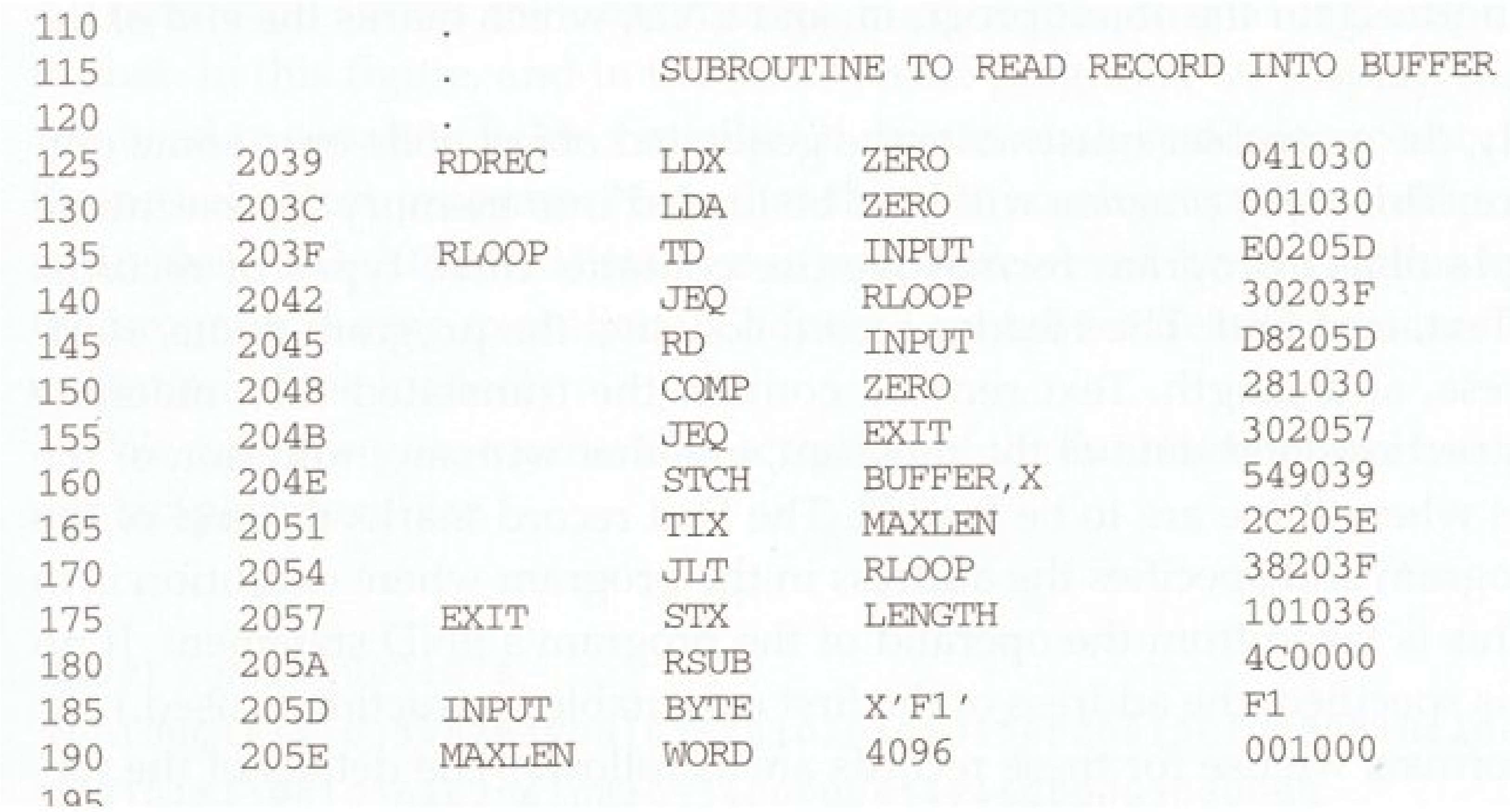
Example program with object code (Figure 2.2 pp. 47)

Opcode for STL is 14



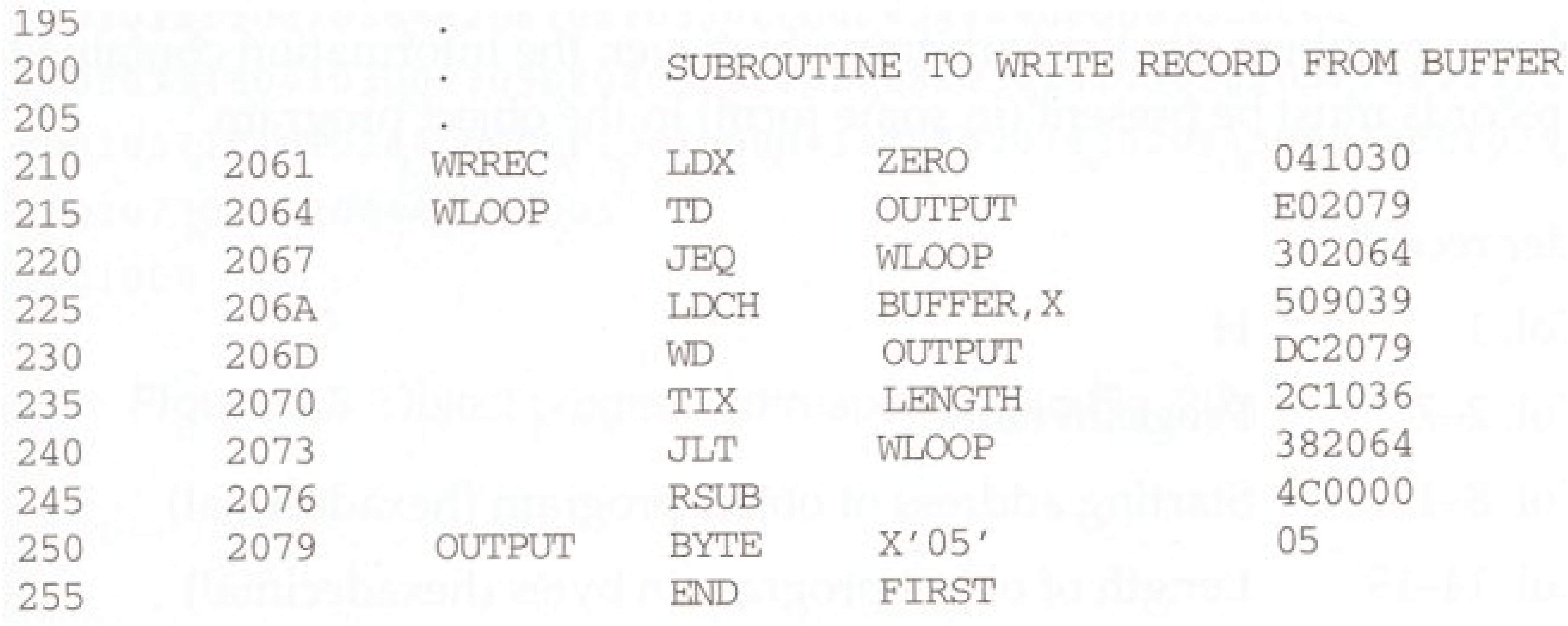
```
Line   Loc    Source statement                Object code

  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           .
```

Example program with object code (Figure 2.2 pp. 47)

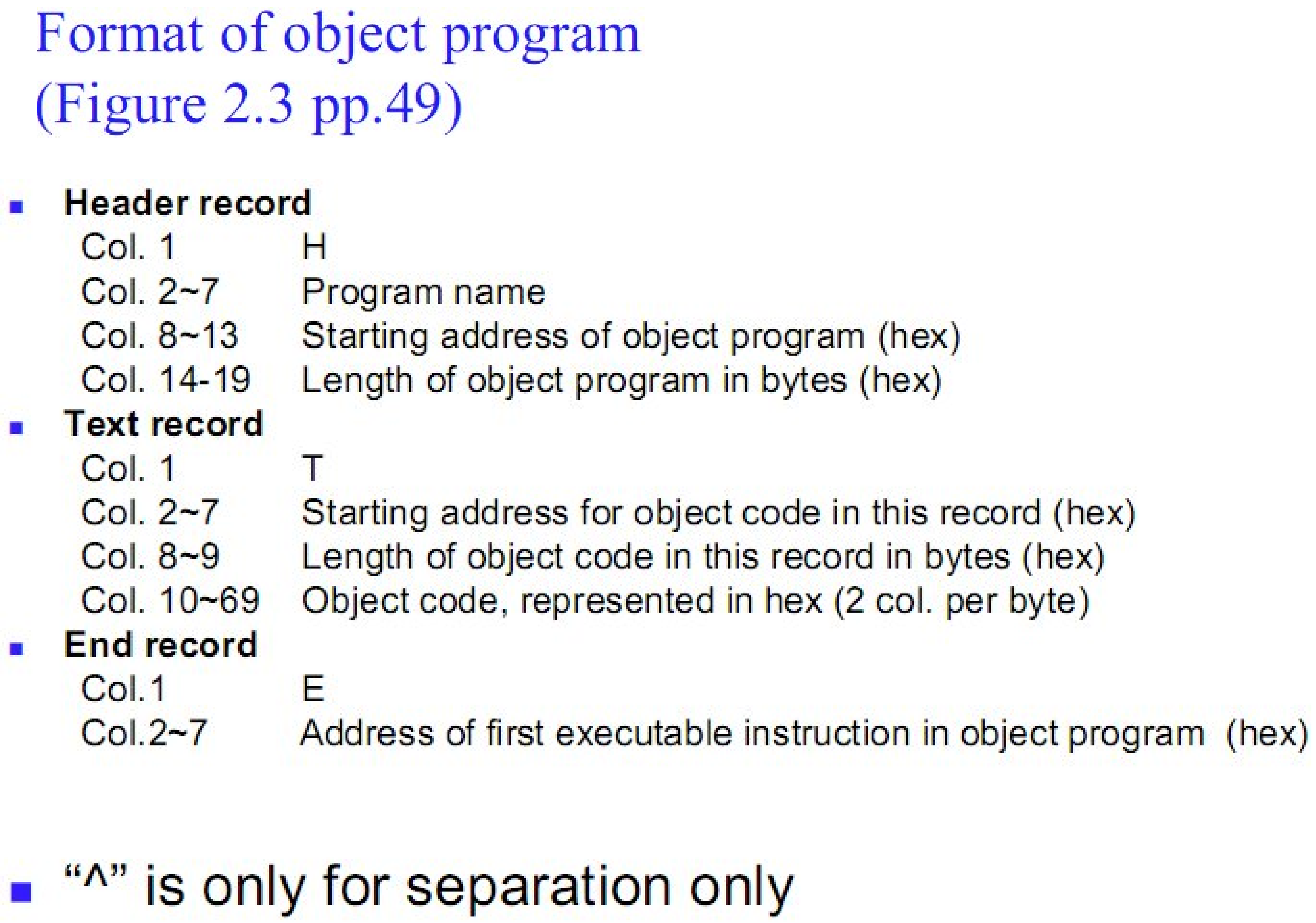


```
                 .
                 .         SUBROUTINE TO READ RECORD INTO BUFFER
                 .
        2039     RDREC     LDX       ZERO                 041030
        203C               LDA       ZERO                 001030
        203F     RLOOP     TD        INPUT                E0205D
        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
```

Example program with object code (Figure 2.2 pp. 47)



```
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
```



Format of object program (Figure 2.3 pp.49)

- **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 hex (2 col. per byte)

- **End record**

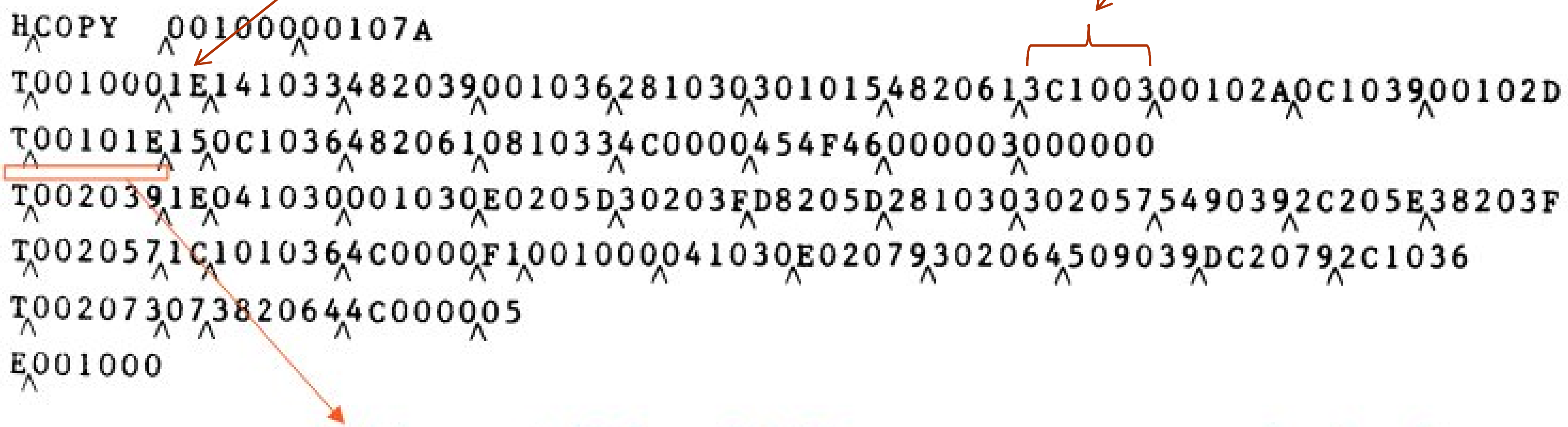
Col.1	E
Col.2~7	Address of first executable instruction in object program (hex)

- “^” is only for separation only

Format of object program (Figure 2.3 pp.49)

Length➔$(1E)_{16}=(30)_{10}$

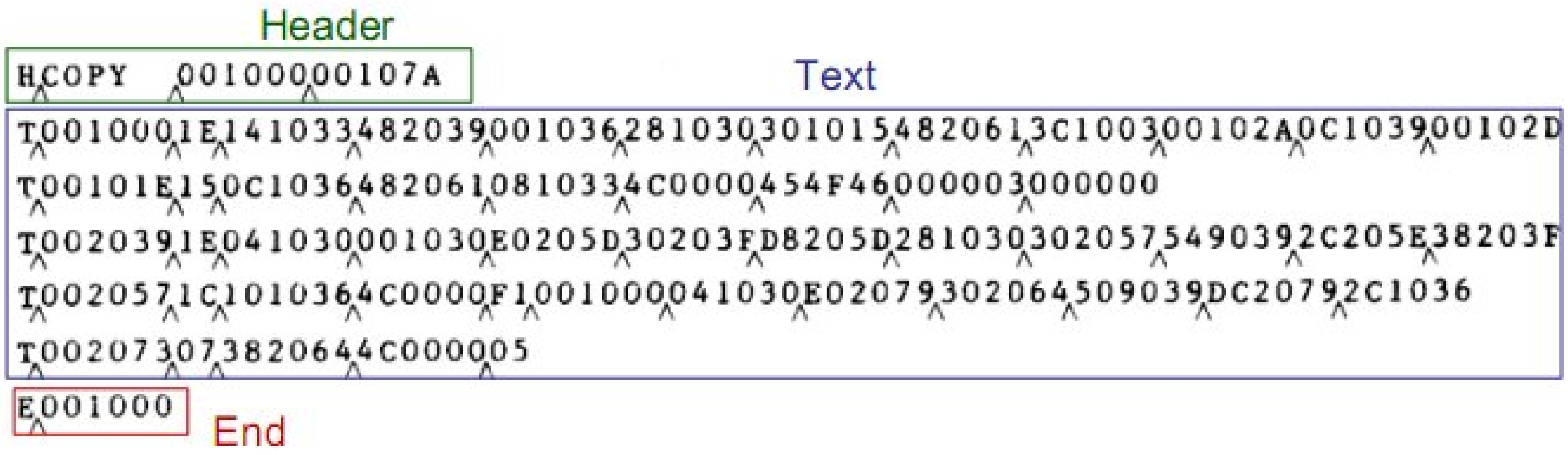
6*4=32bits =3 bytes



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

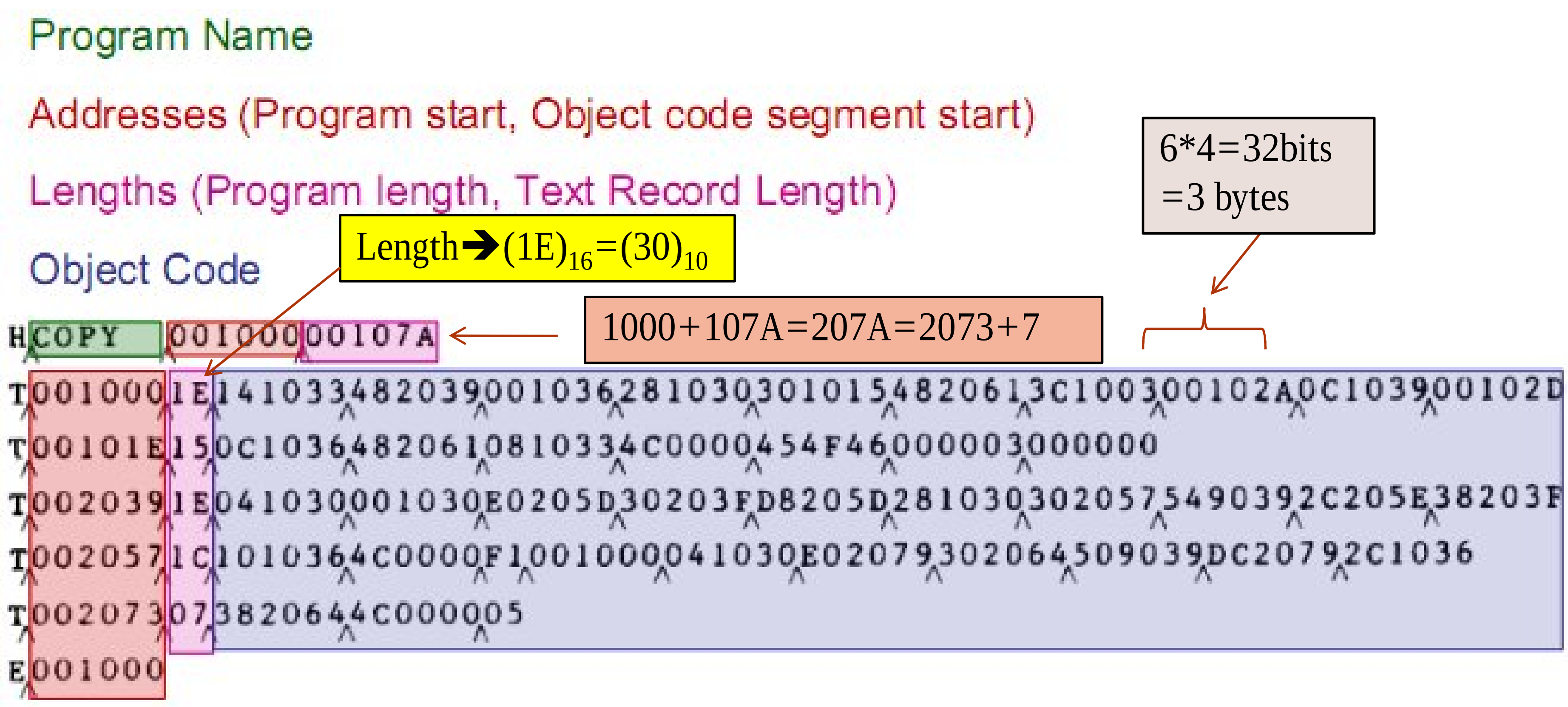
Address 1033 ~ 2038: reserve storage by loader

- RETADR: 3 bytes
- LENGTH: 3 bytes
- BUFFER: 4096 bytes = $(1000)_{16}$


Header
Text
End

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

Figure 2.3 Object program corresponding to Fig. 2.2.



Program Name

Addresses (Program start, Object code segment start)

Lengths (Program length, Text Record Length)

Object Code

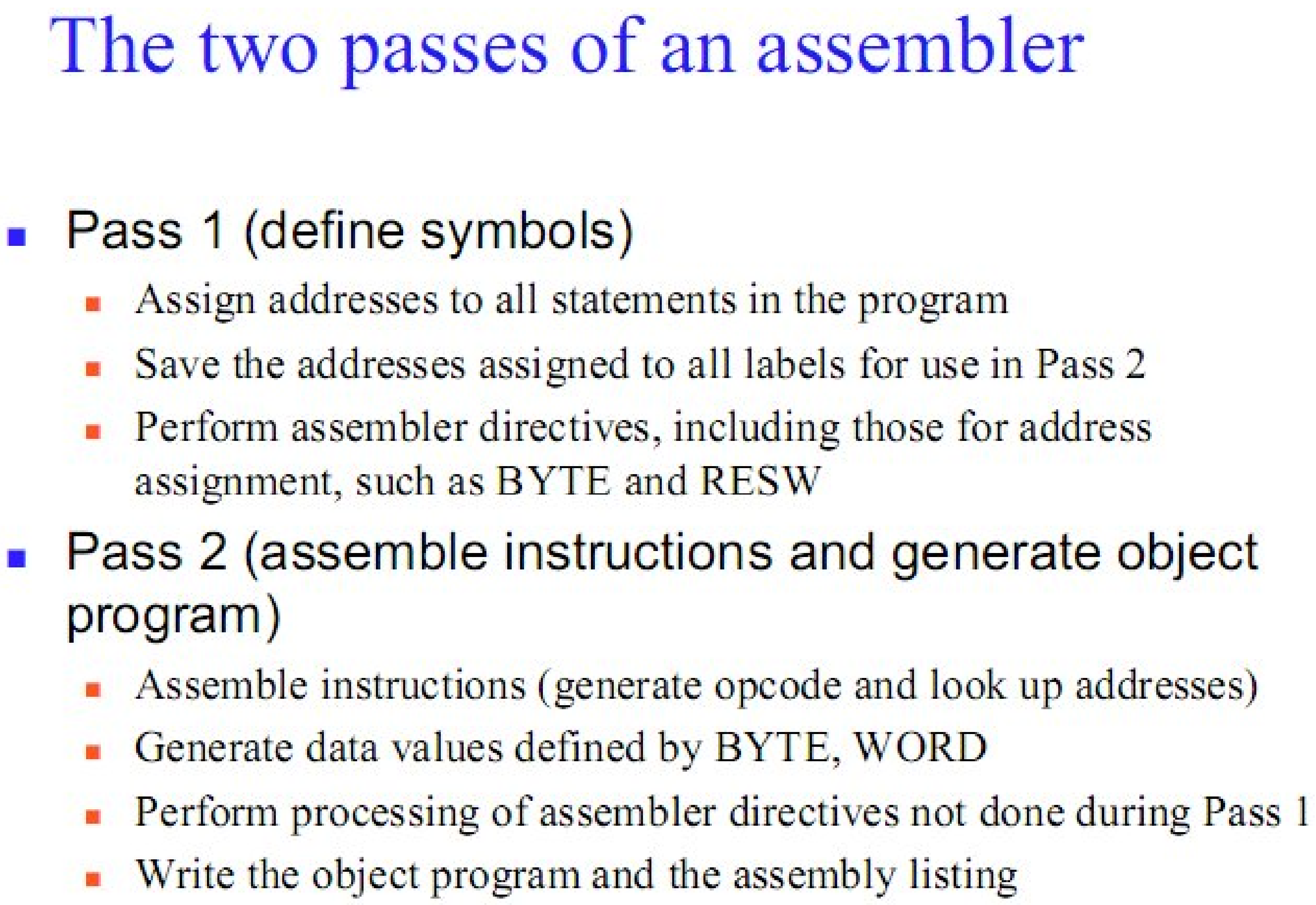
```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036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

Length→$(1E)_{16}=(30)_{10}$

1000+107A=207A=2073+7

6*4=32bits
=3 bytes

Figure 2.3 Object program corresponding to Fig. 2.2.

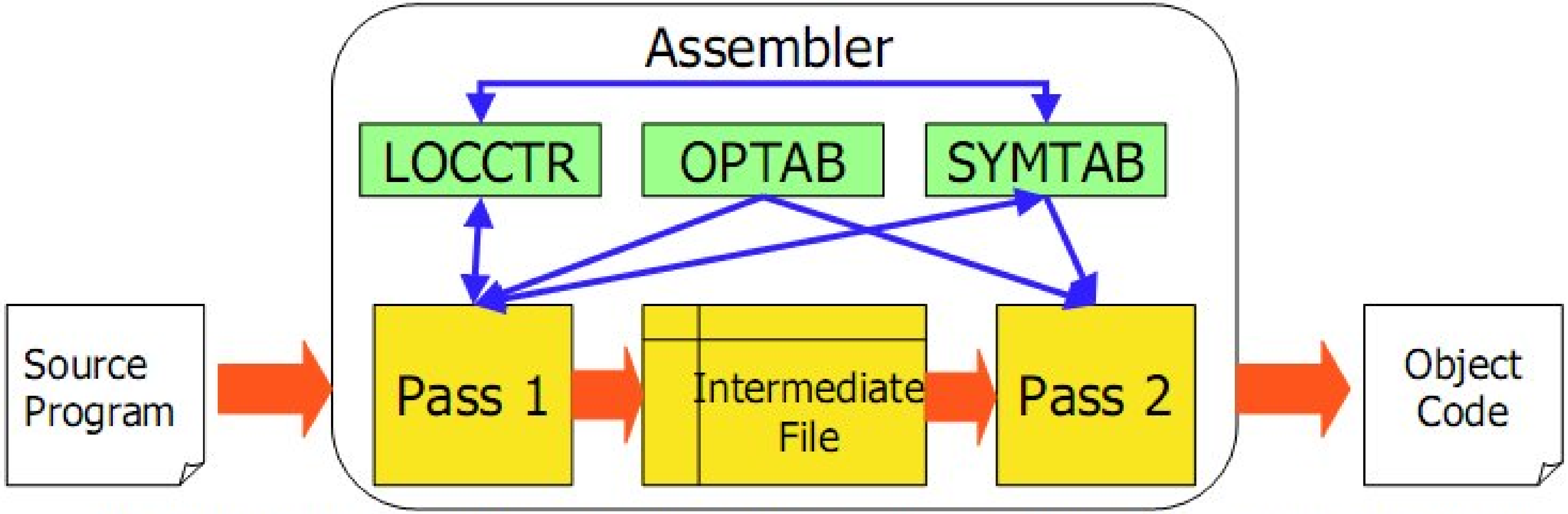


The two passes of an assembler

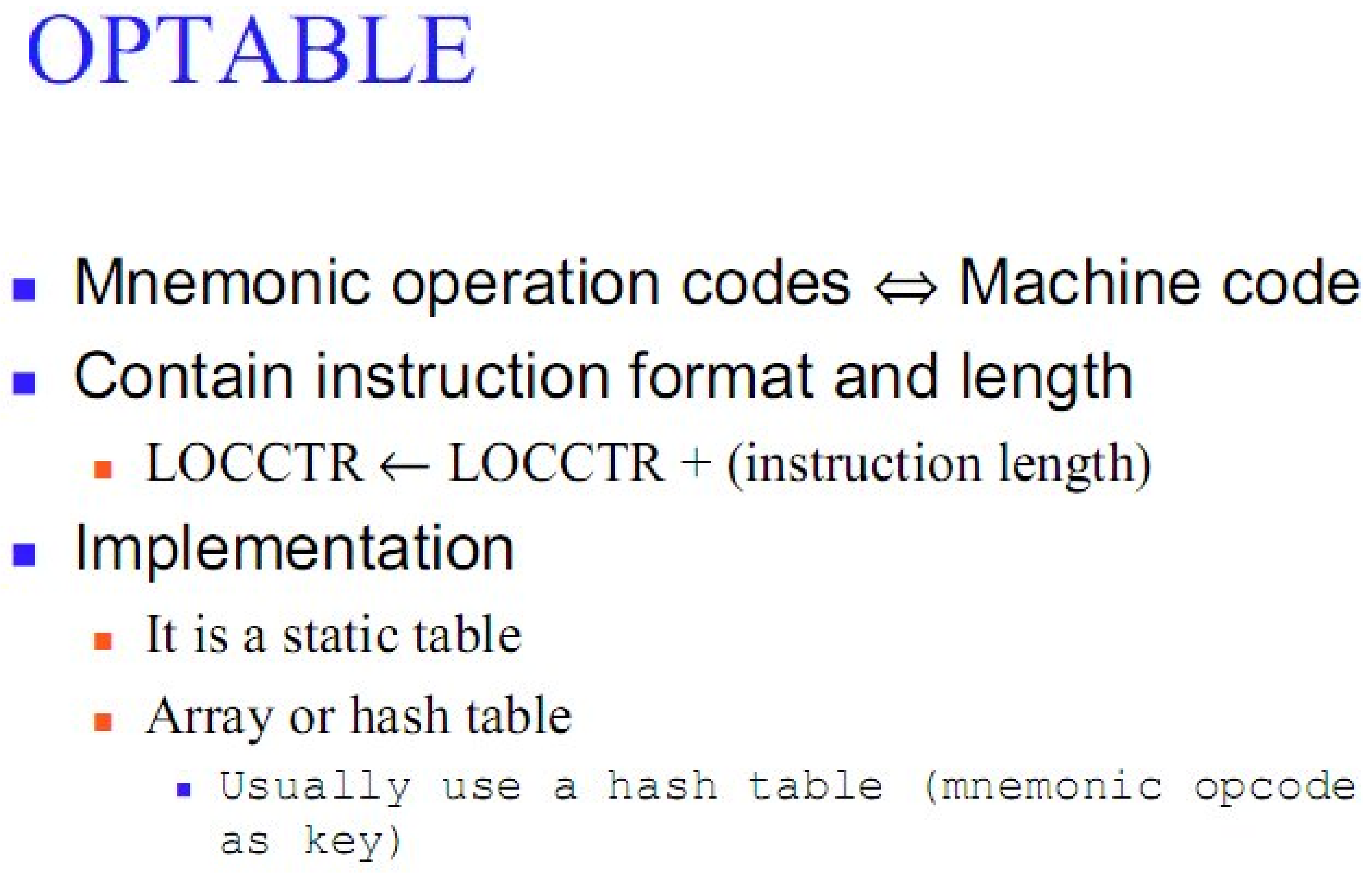
- Pass 1 (define symbols)
  - Assign addresses to all statements in the program
  - Save the addresses assigned to all labels for use in Pass 2
  - Perform assembler directives, including those for address assignment, such as BYTE and RESW
- Pass 2 (assemble instructions and generate object program)
  - Assemble instructions (generate opcode and look up addresses)
  - Generate data values defined by BYTE, WORD
  - Perform processing of assembler directives not done during Pass 1
  - Write the object program and the assembly listing

Assembler algorithm and data structures

- OPTAB: operation code table
- SYMTAB: symbol table
- LOCCTR: location counter

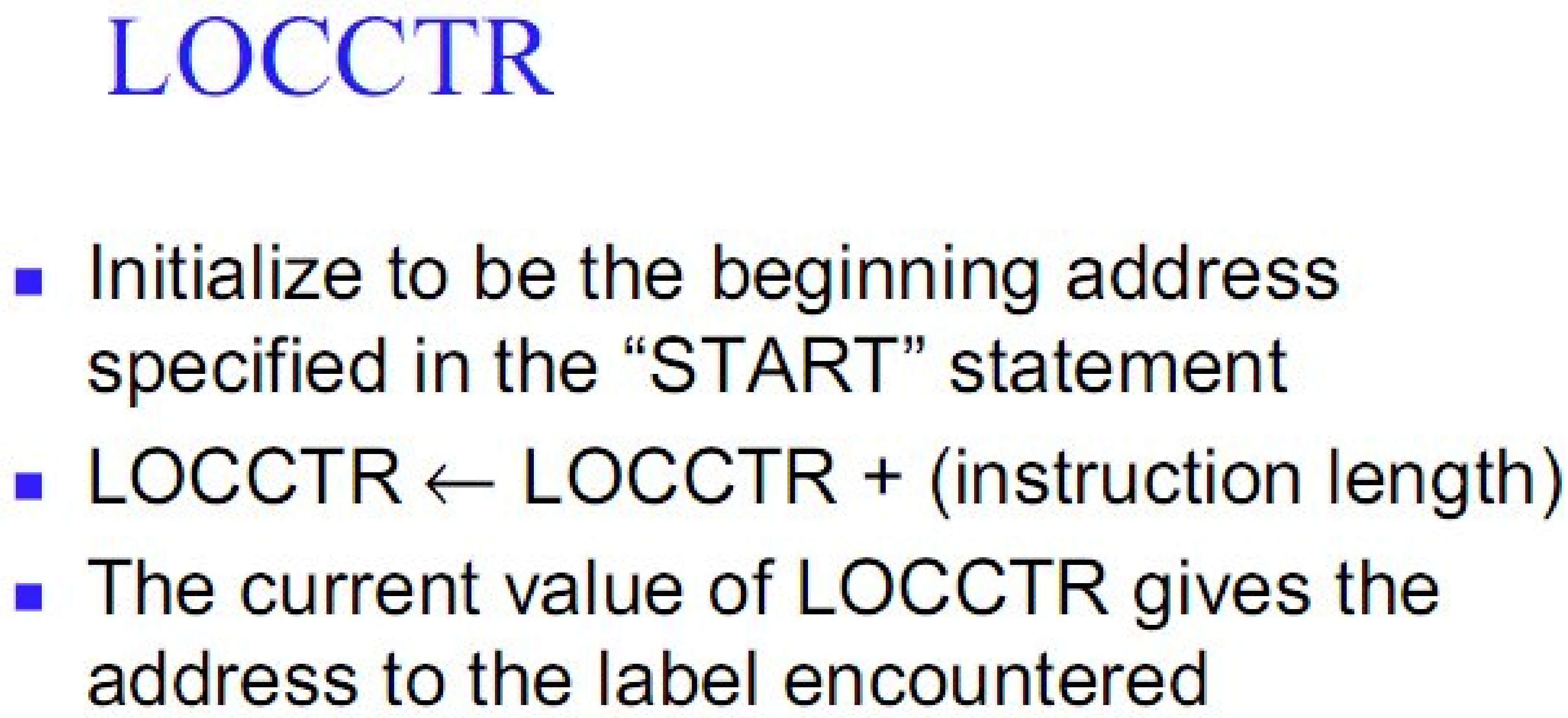


The intermediate file include each source statement, assigned address and error indicator



OPTABLE

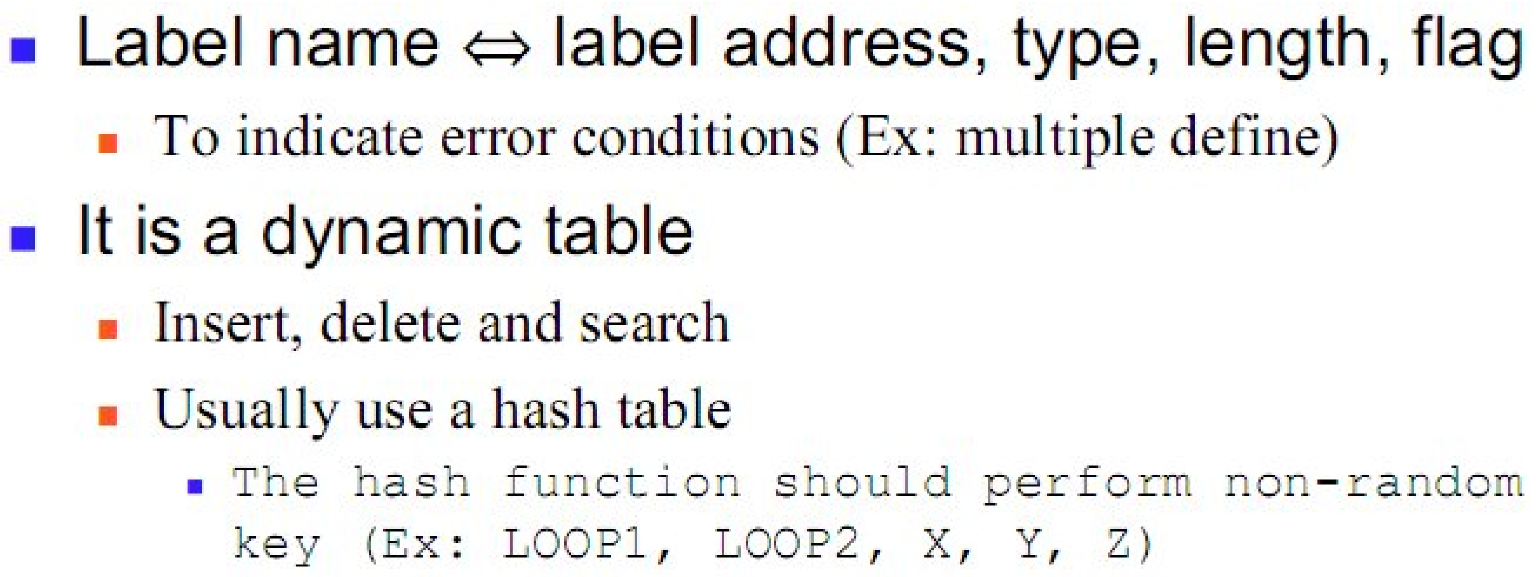
- Mnemonic operation codes ⇔ Machine code
- Contain instruction format and length
  - LOCCTR ← LOCCTR + (instruction length)
- Implementation
  - It is a static table
  - Array or hash table
    - Usually use a hash table (mnemonic opcode as key)



LOCCTR

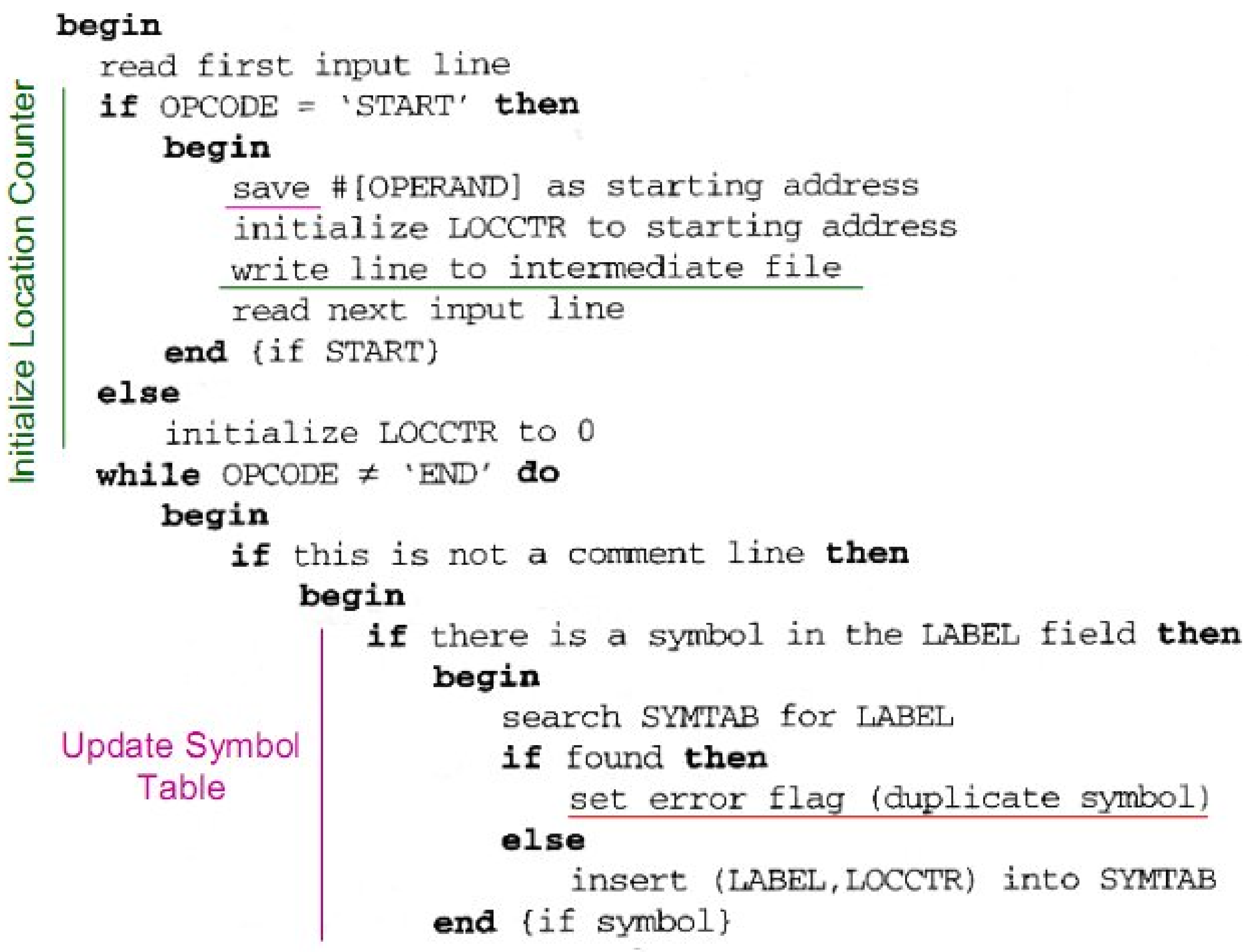
- Initialize to be the beginning address specified in the “START” statement
- LOCCTR ← LOCCTR + (instruction length)
- The current value of LOCCTR gives the address to the label encountered

SYMTAB



- Label name ⇔ label address, type, length, flag
  - To indicate error conditions (Ex: multiple define)
- It is a dynamic table
  - Insert, delete and search
  - Usually use a hash table
    - The hash function should perform non-random key (Ex: LOOP1, LOOP2, X, Y, Z)

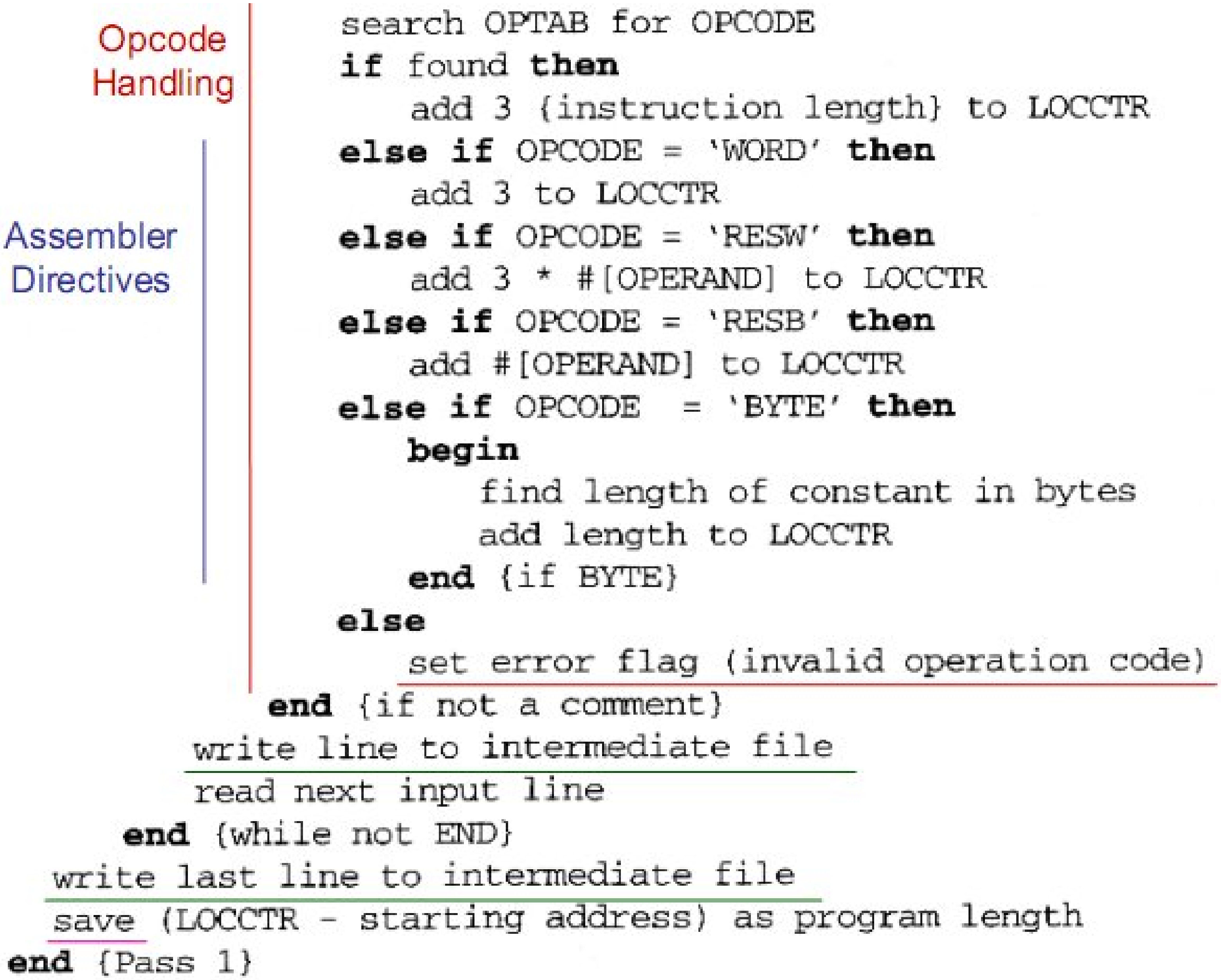
Two-Pass SIC Assembler: Pass 1 (Figure 2.4a)



Initialize Location Counter

Update Symbol Table

```
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}
```

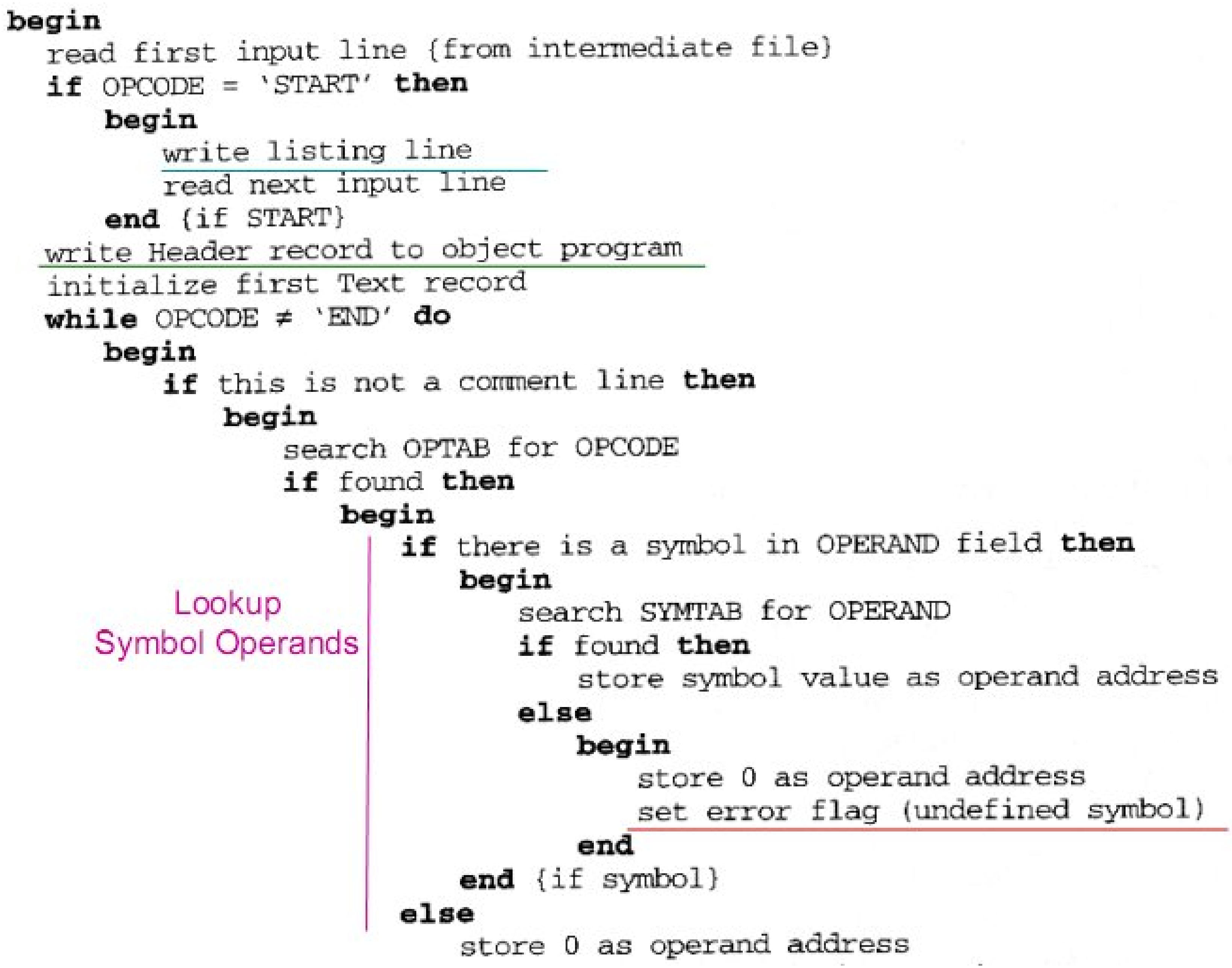


Opcode Handling

Assembler Directives

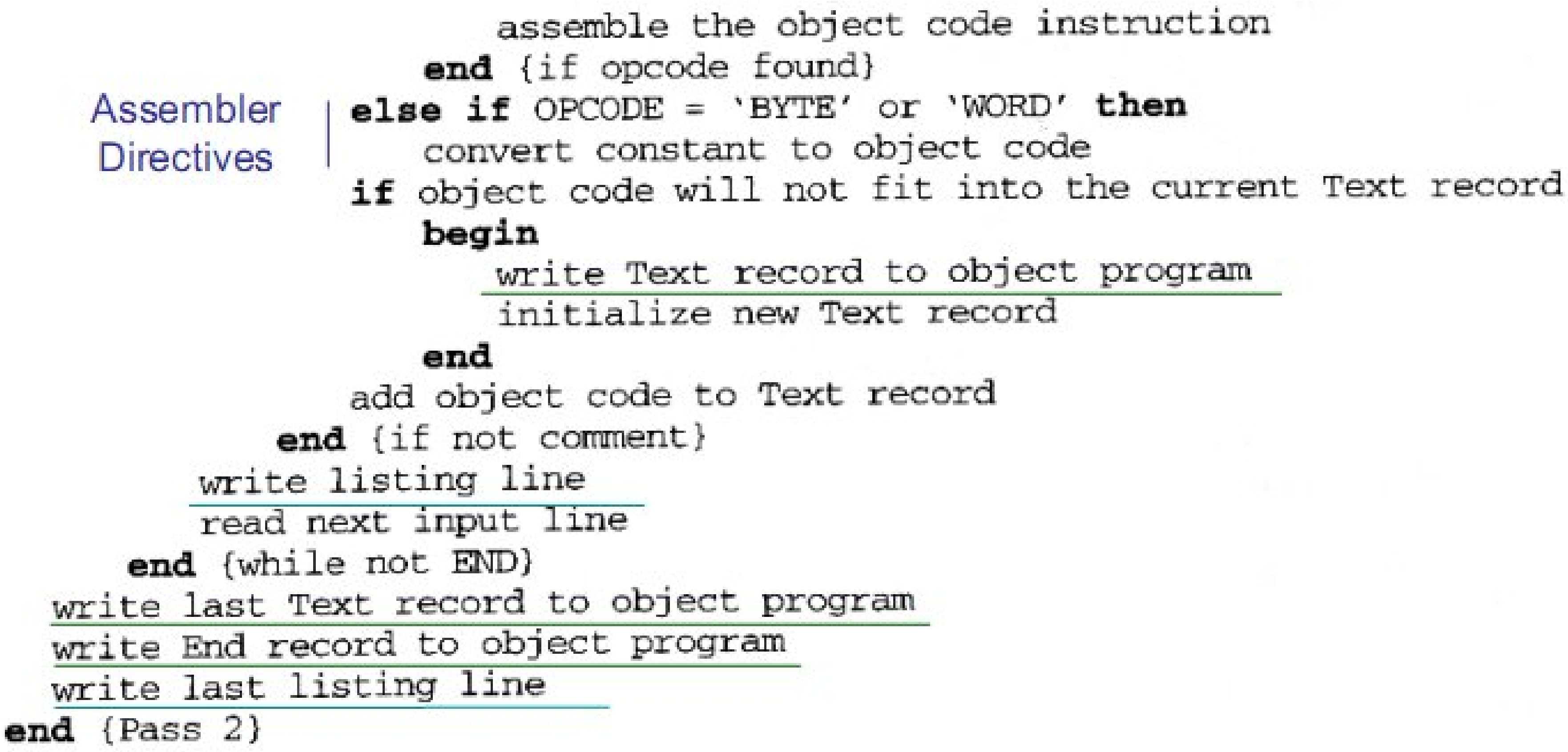
```
                search OPTAB for OPCODE
                if found then
                    add 3 {instruction length} to LOCCTR
                else if OPCODE = 'WORD' then
                    add 3 to LOCCTR
                else if OPCODE = 'RESW' then
                    add 3 * #[OPERAND] to LOCCTR
                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}
```

A Simple Two-Pass SIC Assembler: Pass 2 (Figure 2.4b)



```
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
```

Lookup
Symbol Operands



Assembler Directives

```
              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
            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}
```