A Simple Two-Pass Assembler

Main Functions

- Translate mnemonic operation codes to their machine language equivalents
- Assign machine addresses to symbolic labels used by the programmers

- Machine dependency

Design of an assembler depend heavily on the source language it translates and the machine language it produces.

E.g., the instruction format and addressing modes

Basic Assembler Functions

Purpose of Example 2.1 (COPY)

- It is a copy function that reads some records from a specified input device and then copies them to a specified output device
  - Reads a record from the input device (code F1)
  - Copies the record to the output device (code 05)
  - Repeats the above steps until encountering EOF.
  - Then writes EOF to the output device
  - Then call RSUB to return to the caller

Example Program (Fig. 2.1)

- Main routine calls subroutine RDREC to read a record into a buffer and subroutine WRREC to write the record from the buffer to the output device.
- Each subroutine transfer the record one character at a time.(the only I/O instructions are RD and WD)
  - a buffer is used to store record
  - buffering is necessary for different I/O rates of devices.

RDREC and WRREC

- Data transfer
  - A record is a stream of bytes with a null character (00_{16}) at the end.
  - If a record is longer than 4096 bytes(length of the buffer), only the first 4096 bytes are copied.
  - EOF is indicated by a zero-length record. (i.e., a byte stream with only a null character.)
  - Because the speed of the input and output devices may be different, a buffer is used to temporarily store the record.
  - When EOF is detected, the program writes it to the output device.
  - Then call RSUB to return to the caller

SIC Assembly Program (Fig. 2.1)

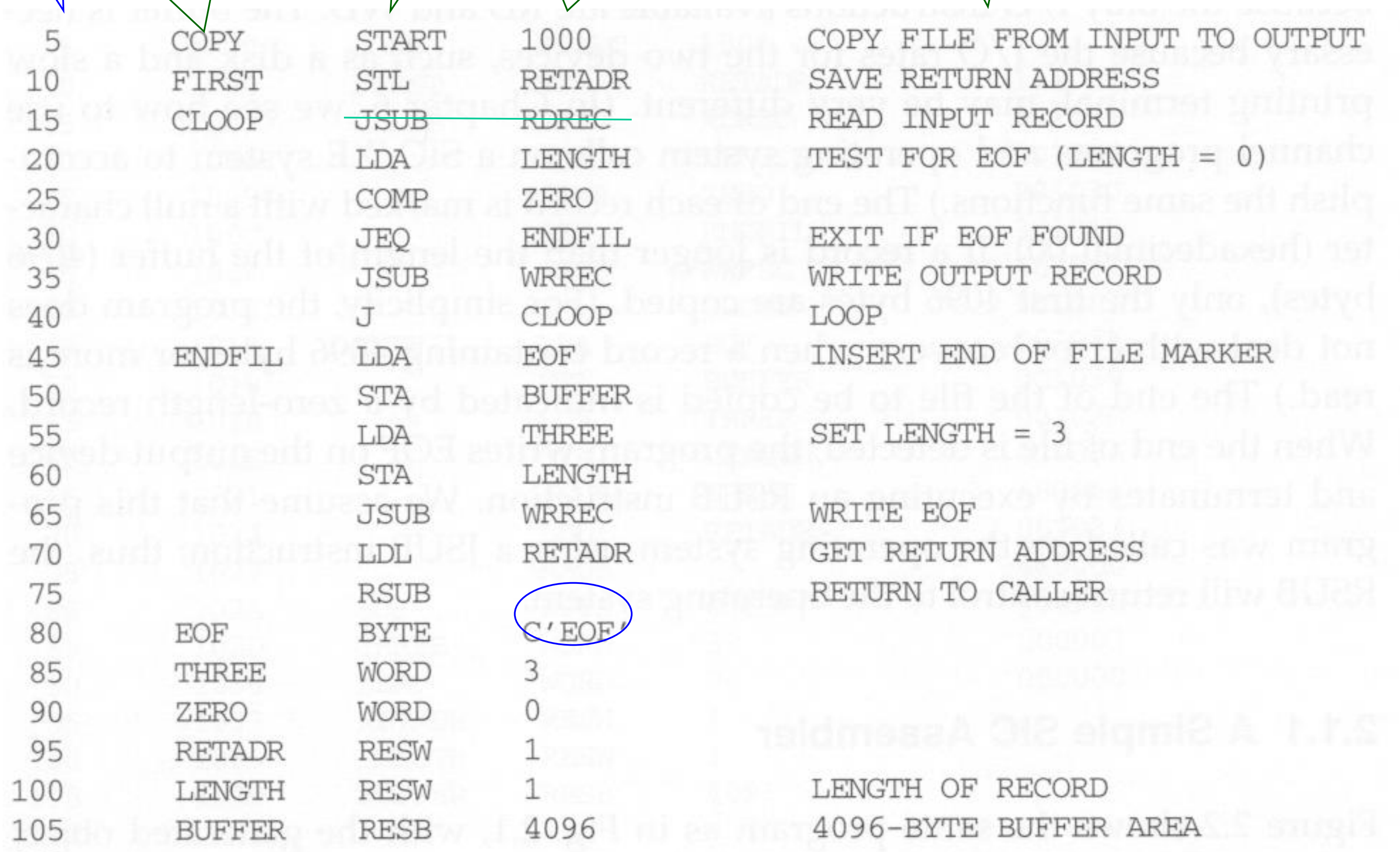
Line numbers (for reference)

Address labels

Mnemonic opcode

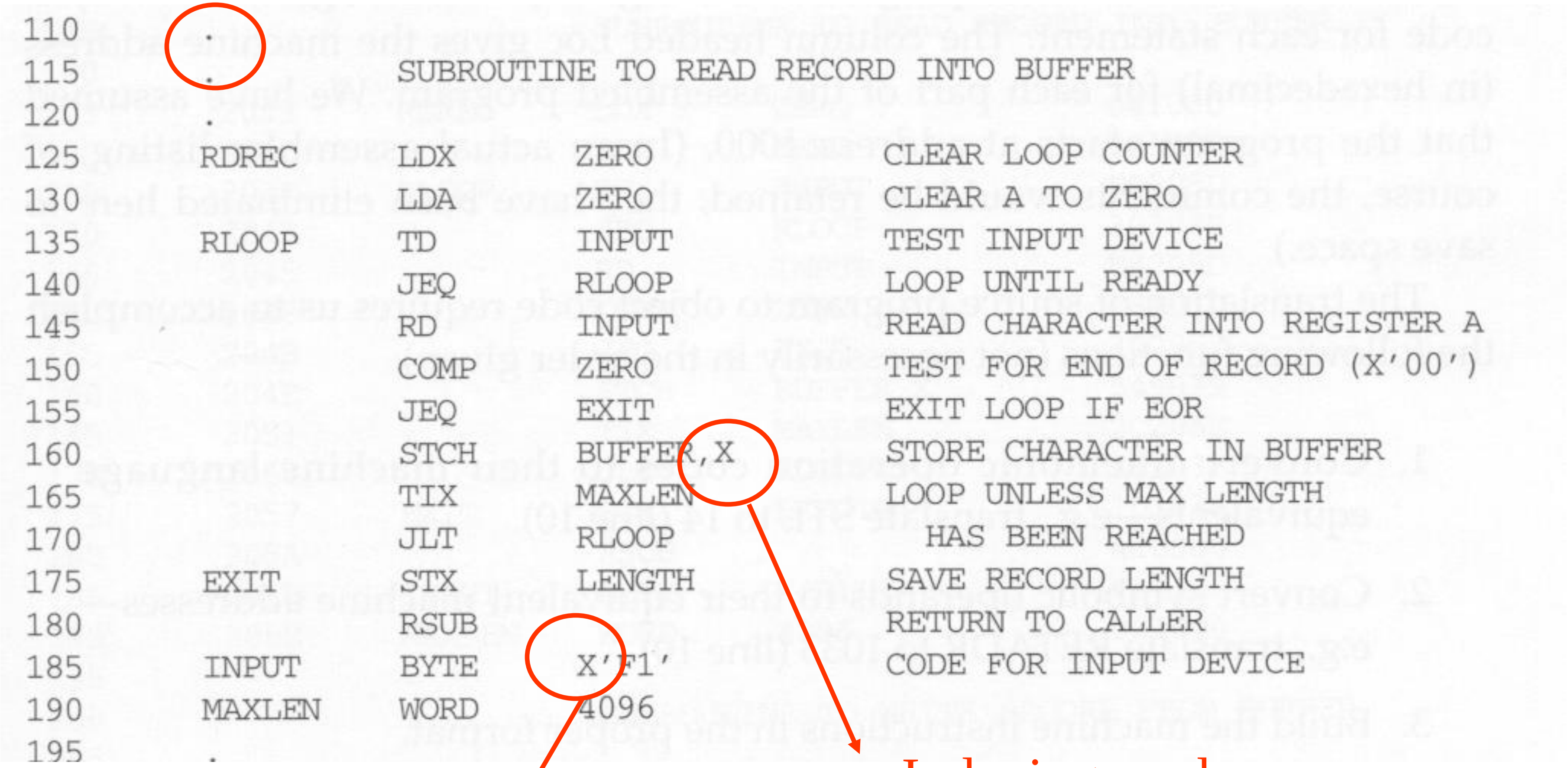
operands

comments



```
  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
```

Comment line

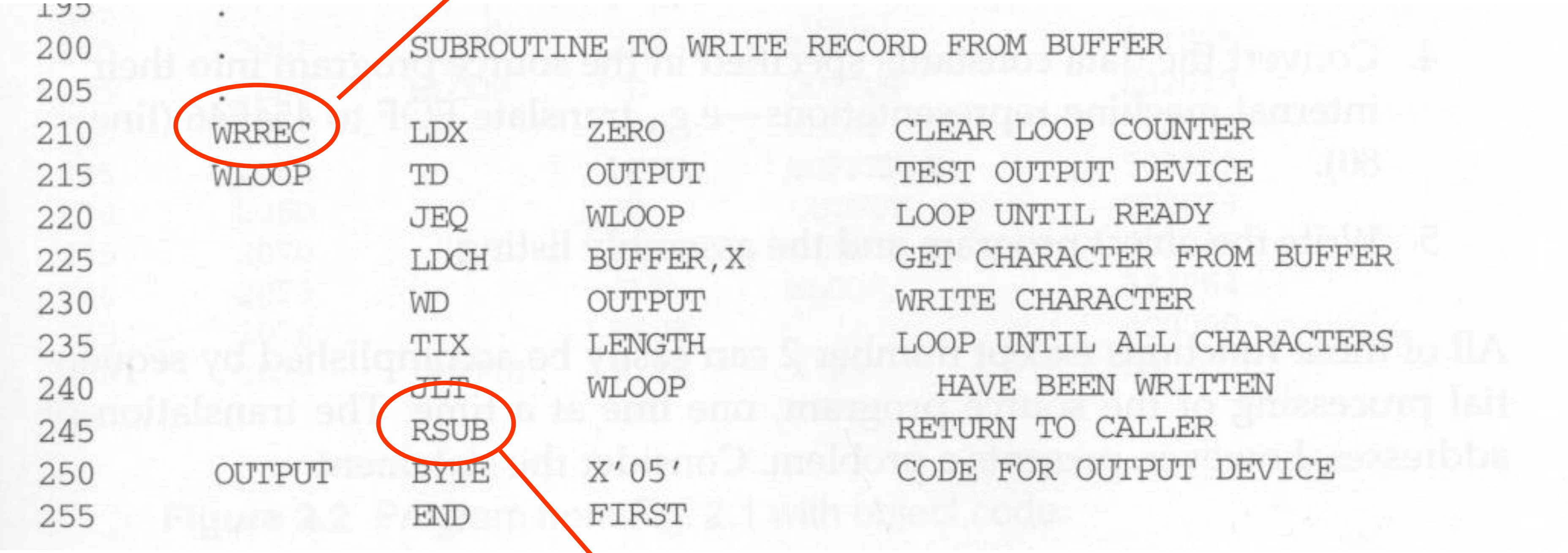


```
110        .
115        .         SUBROUTINE TO READ RECORD INTO BUFFER
120        .
125  RDREC      LDX      ZERO              CLEAR LOOP COUNTER
130             LDA      ZERO              CLEAR A TO ZERO
135  RLOOP      TD       INPUT             TEST INPUT DEVICE
140             JEQ      RLOOP             LOOP UNTIL READY
145             RD       INPUT             READ CHARACTER INTO REGISTER A
150             COMP     ZERO              TEST FOR END OF RECORD (X'00')
155             JEQ      EXIT              EXIT LOOP IF EOR
160             STCH     BUFFER,X          STORE CHARACTER IN BUFFER
165             TIX      MAXLEN            LOOP UNLESS MAX LENGTH
170             JLT      RLOOP               HAS BEEN REACHED
175  EXIT       STX      LENGTH            SAVE RECORD LENGTH
180             RSUB                       RETURN TO CALLER
185  INPUT      BYTE     X'F1'             CODE FOR INPUT DEVICE
190  MAXLEN     WORD     4096
195        .
```

Indexing mode

Hexadecimal number

Subroutine entry point



```
          .
          .         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
```

Subroutine return point

Assembler Directives

- Assembler directives are pseudo instructions
  - They will not be translated into machine instructions.
  - They only provide instruction/direction/information to the assembler.
- Basic assembler directives :
  - START :
    - Specify name and starting address for the program
  - END :
    - Indicate the end of the source program, and (optionally) the first executable instruction in the program.

Assembler Directives (cont'd)

- BYTE :
  - Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant.
- WORD :
  - Generate one-word integer constant
- RESB :
  - Reserve the indicated number of bytes for a data area
- RESW :
  - Reserve the indicated number of words for a data area

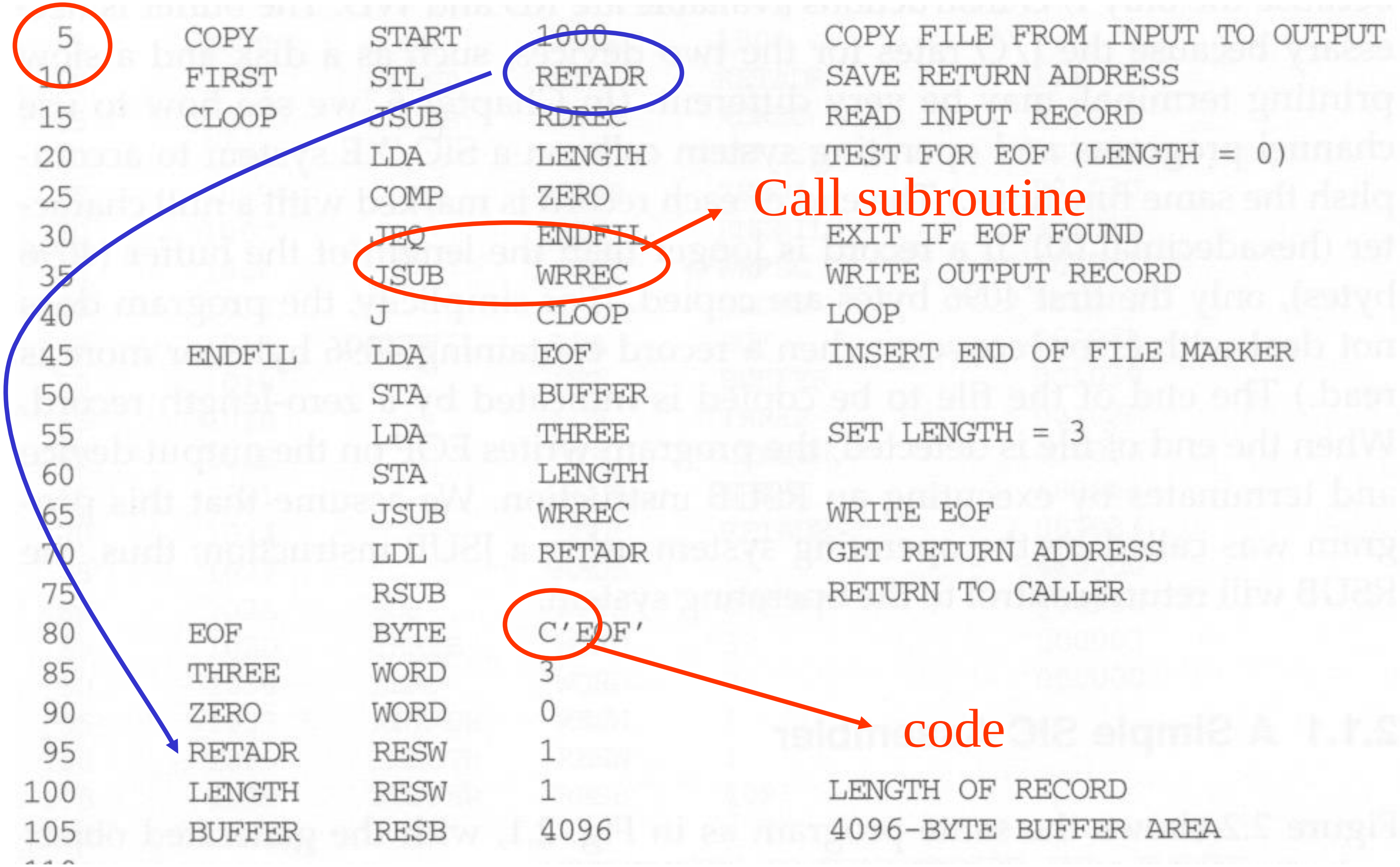
An Assembler’s Job

- Convert mnemonic operation codes to their machine language codes - eg: translate STL to 14 (line 10)
- Convert symbolic (e.g., jump labels, variable names) operands to their machine addresses - eg: translate RETADR to 1033 (line 10)
- Use proper addressing modes and formats to build efficient machine instructions
- Translate data constants into their internal machine representations - eg: translate EOF to 454F46 (line 80)
- Output the object program and provide other information (e.g., for linker and loader)

Example 2.1

Line numbers are not part of the program. They are for reference only.

Forward reference

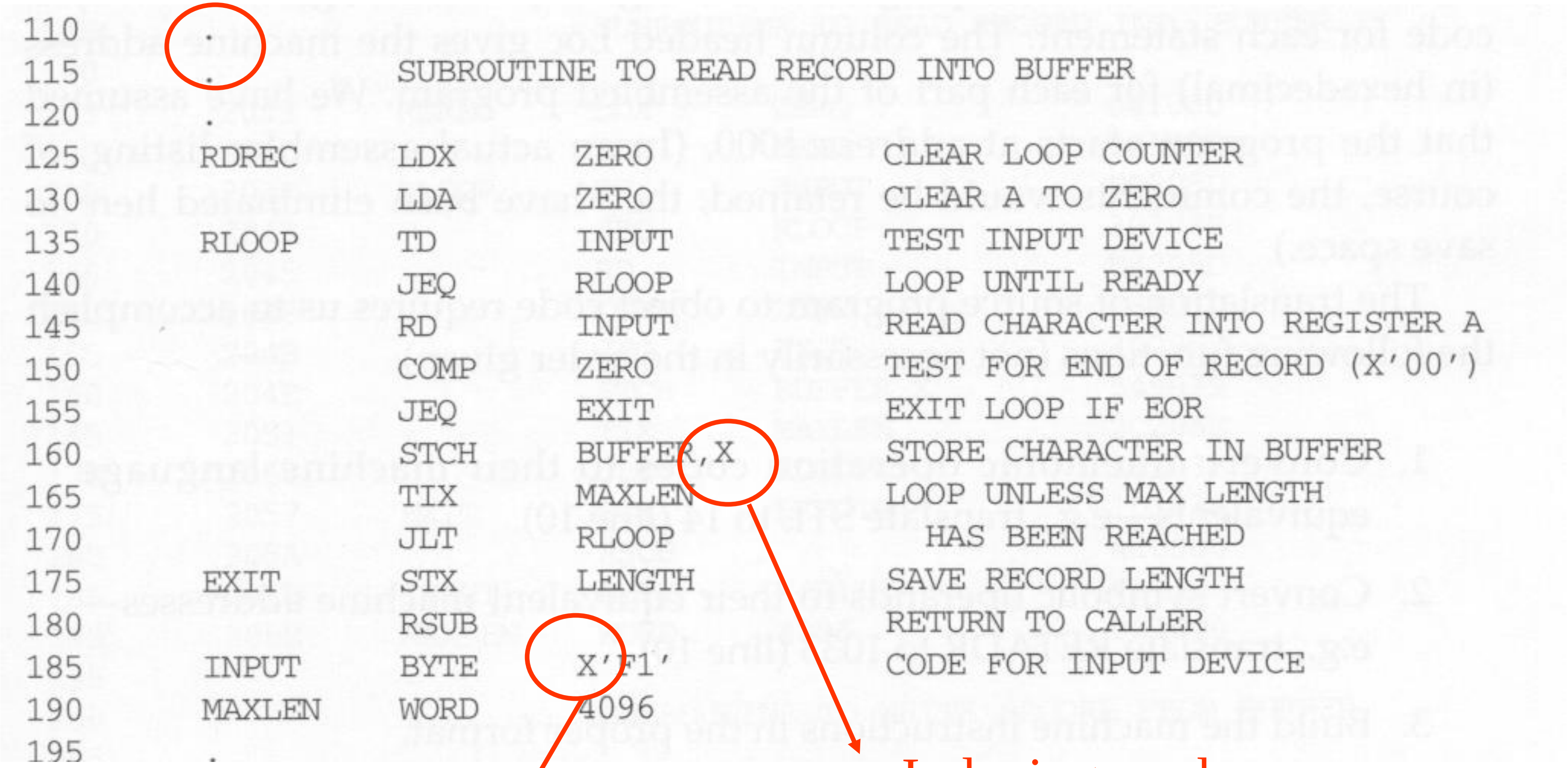


```
  5   COPY     START   1000       COPY FILE FROM INPUT TO OUTPUT
 10   FIRST    STL     RETADR     SAVE RETURN ADDRESS
 15   CLOOP    JSUB    RDREC      READ INPUT RECORD
 20            LDA     LENGTH     TEST FOR EOF (LENGTH = 0)
 25            COMP    ZERO
 30            JEQ     ENDFIL     EXIT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

Call subroutine

code

Comment line

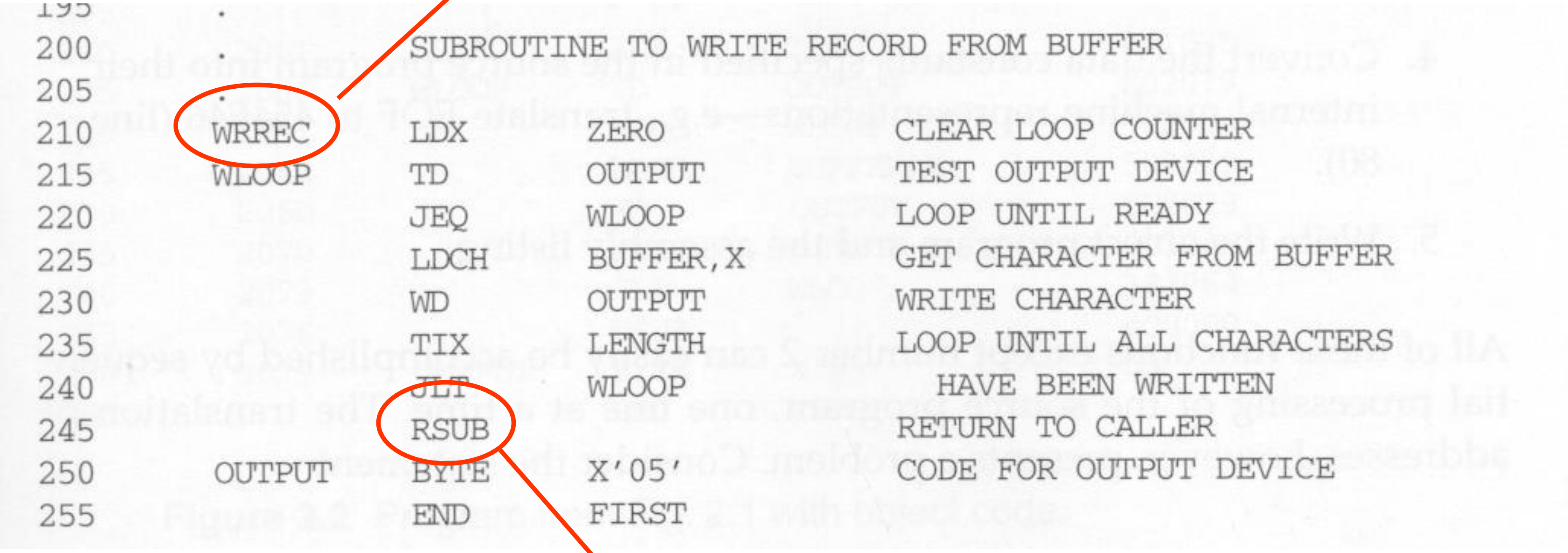


```
        .
        .         SUBROUTINE TO READ RECORD INTO BUFFER
        .
RDREC   LDX      ZERO            CLEAR LOOP COUNTER
        LDA      ZERO            CLEAR A TO ZERO
RLOOP   TD       INPUT           TEST INPUT DEVICE
        JEQ      RLOOP           LOOP UNTIL READY
        RD       INPUT           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
        .
```

Indexing mode

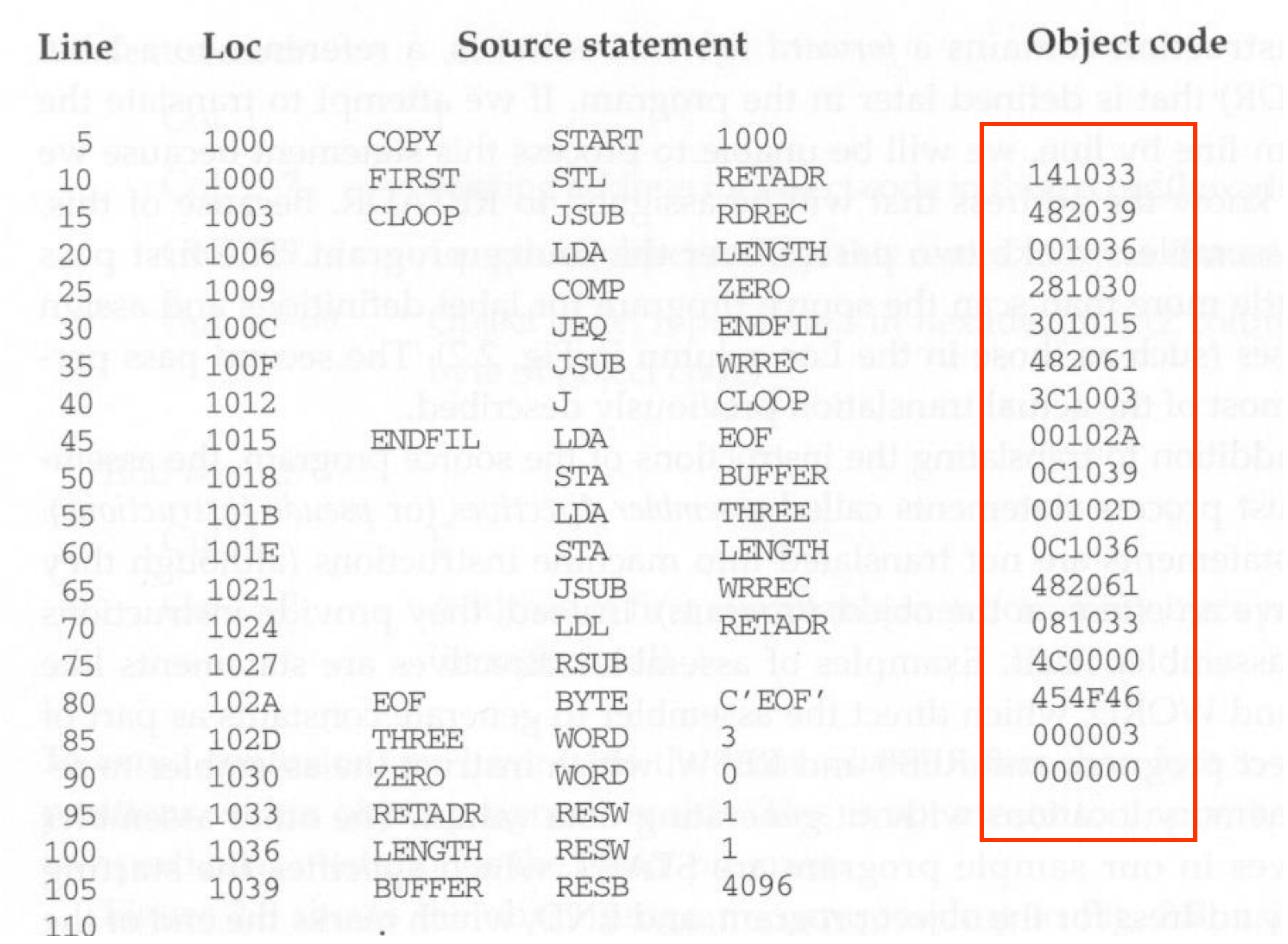
Hexadecimal number

Subroutine entry point



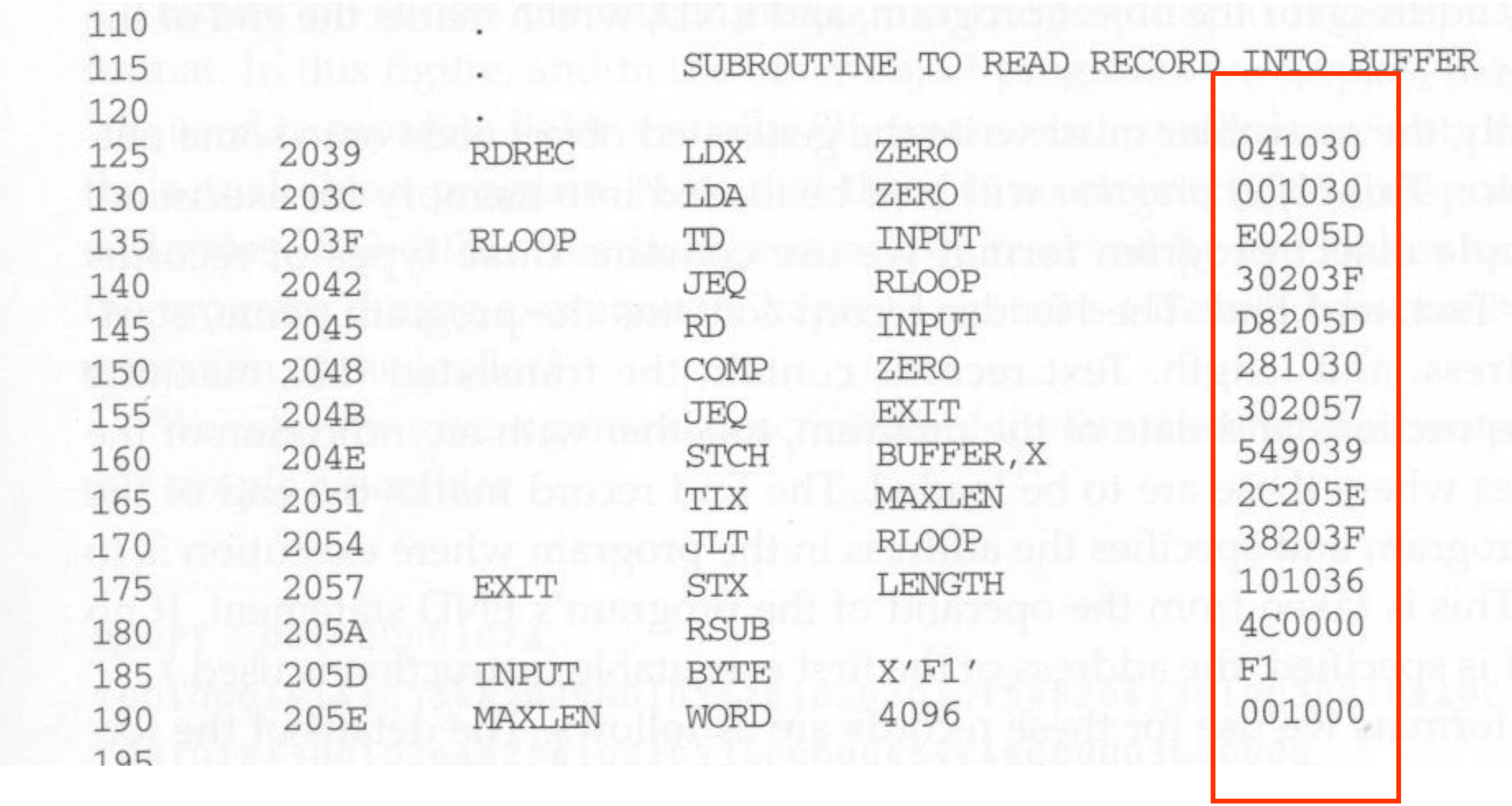
```
.
.          SUBROUTINE TO WRITE RECORD FROM BUFFER
.
WRREC      LDX        ZERO             CLEAR LOOP COUNTER
WLOOP      TD         OUTPUT           TEST OUTPUT DEVICE
           JEQ        WLOOP            LOOP UNTIL READY
           LDCH       BUFFER,X         GET CHARACTER FROM BUFFER
     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

Subroutine return point

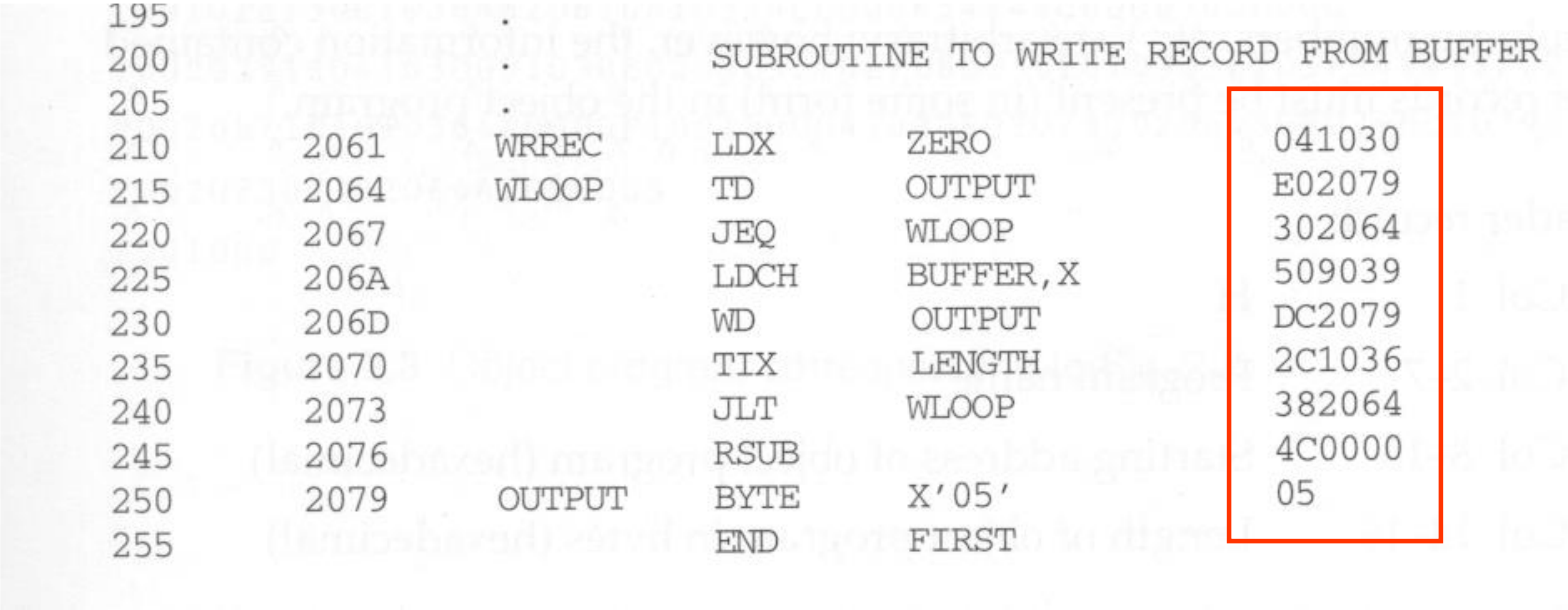


```
Line   Loc     Source statement                 Object code

  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            .
```



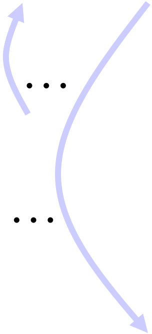
```
110                  .
115                  .               SUBROUTINE TO READ RECORD INTO BUFFER
120                  .
125     2039    RDREC       LDX     ZERO                          041030
130     203C                LDA     ZERO                          001030
135     203F    RLOOP       TD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



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

Difficulties: Forward Reference

- **Forward reference**: reference to a label (RETADR) that is defined later in the program.



Loc	Label	Operator	Operand	Object code
1000	FIRST	STL	RETADR	141033
1003	CLOOP	JSUB	RDREC	482039
…	…	…	…	…
1012		J	CLOOP	3C1003
…	…	…	…	…
1033	RETADR	RESW	1	

Forward Reference

- A reference to a label (RETADR) that is defined later in the program
- Solution
  - – Two passes
    - First pass: scan the source program for label definition and assign addresses.
    - Second pass: performs most of the actual instruction translation previously defined.

- The assembler must write the generated object code on to the output device.
- The simple object program format contains three types of records:
- Header record:- contains program name, staring address and length.
- Text Record:- contains translated instructions and data and the addresses where these are to be loaded.
- End record:- mark the end of the object program and specifies the address where execution is to begin.

Object Program Format

- Header

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

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 hexa (2 col. per byte)

- End

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

The Object Code for COPY

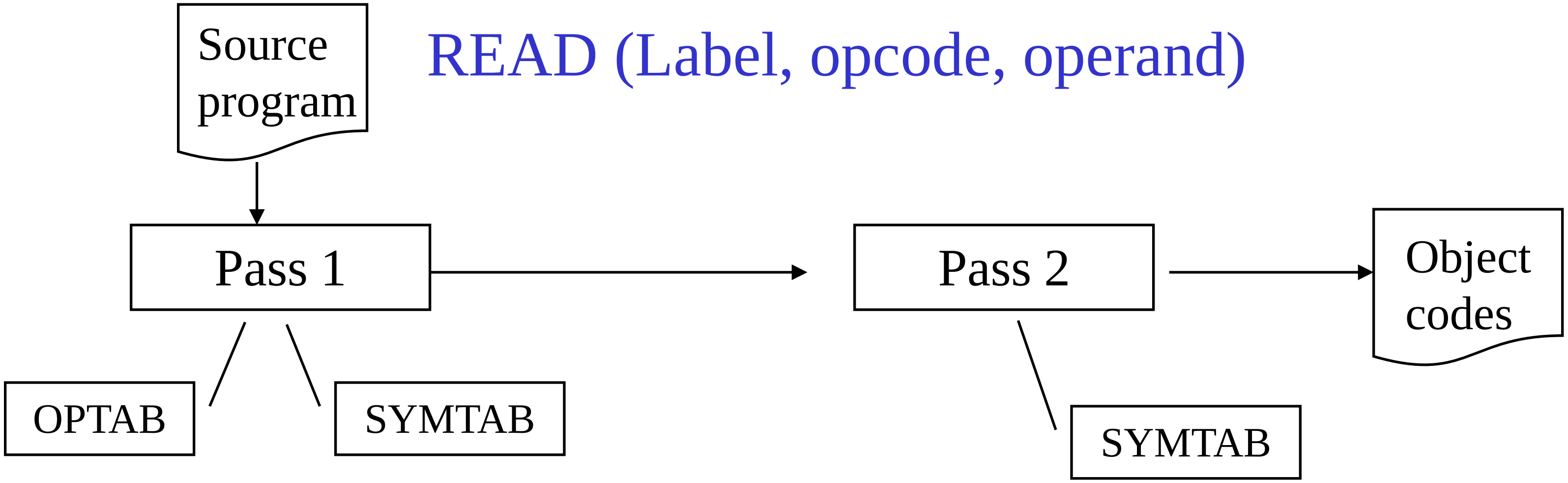
```
H COPY  001000 00107A
T 001000 1E 141033 482039 001036 281030 301015 482061 3C1003
    00102A 0C1039 00102D
T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 000000
T 002039 1E 041030 001030 E0205D 30203F D8205D 281030 302057
    549039 2C205E 38203F
T 002057 1C 101036 4C0000 F1 001000 041030 E02079 302064 509039
    DC2079 2C1036
T 002073 07 382064 4C0000 05
E 001000
```

There is no object code corresponding to addresses 1033-2038. This storage is simply reserved by the loader for use by the program during execution.

Two Pass Assembler

- Pass 1(define symbols):
  - Assign addresses to all statements in the program
  - Save the values (addresses) assigned to all labels (including label and variable names) for use in Pass 2 (deal with forward references)
  - Perform some processing of assembler directives (e.g., BYTE, RESW, these can affect address assignment)
- Pass 2(assemble instructions and generate object program):
  - Assemble instructions (translate opcode and look up addresses)
  - Generate data values defined by BYTE, WORD
  - Perform processing of assembler directives not done in Pass 1
  - Write the object program and the assembly listing

A Simple Two Pass Assembler Implementation



READ (Label, opcode, operand)

Mnemonic and opcode mappings are referenced from here

Label and address mappings enter here

Label and address mappings are referenced from here

Main Data Structures

- Operation Code Table (OPTAB) : used to lookup mnemonic opcodes and their machine language equivalent.
- Symbol Table (SYMTAB): used to store values(addresses) assigned to labels.

Location Counter (LOCCTR) :- a variable help in the assignment of addresses.

OPTAB (operation code table)

- Content
  - The mapping between mnemonic opcode and its equivalent machine code. Also include the instruction format, available addressing modes, and length info.
- Characteristic
  - Static table. The content will never change.
- Implementation
  - hash table with opcode as key.
  - it provide fast retrieval with minimum search.
- In pass 1, OPTAB is used to look up and validate mnemonics in the source program.
- In pass 2, OPTAB is used to translate mnemonics to machine instructions.

Location Counter (LOCCTR)

- This variable can help in the assignment of addresses.
- It is initialized to the beginning address specified in the START statement.
- After each source statement is processed, the length of the assembled instruction and data area to be generated is added to LOCCTR.
- Thus, when we reach a label in the source program, the current value of LOCCTR gives the address to be associated with that label.(ie. point to the next location where the code will be placed)

Symbol Table (SYMTAB)

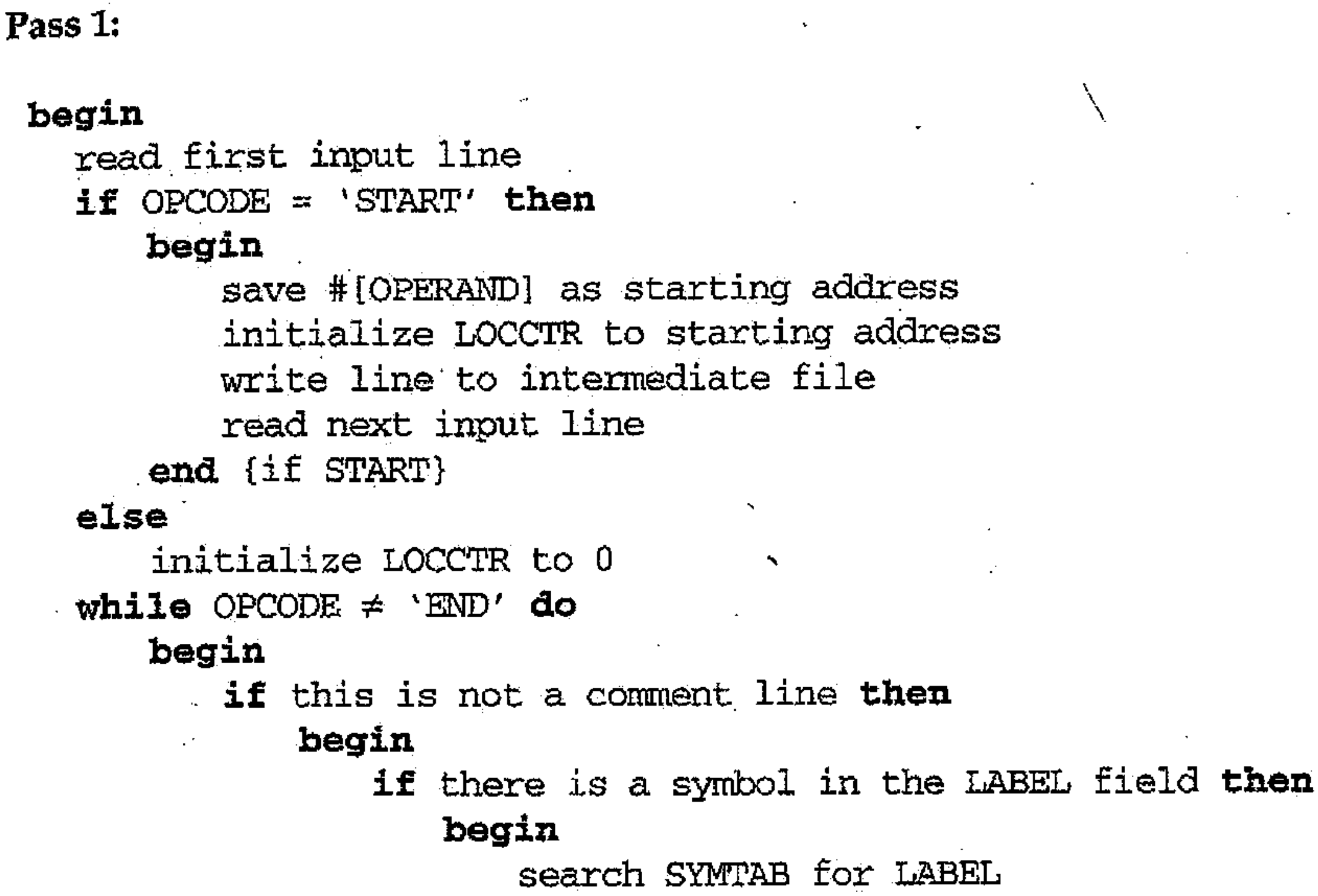
- Content
  - Include the label name and value (address) for each label in the source program.
  - Include type and length information of the data area or instruction labeled.
  - With flag to indicate errors (e.g., a symbol defined in two places)
- Characteristic
  - Dynamic table (i.e., symbols may be inserted, deleted, or searched in the table)
- Implementation
  - Hash table can be used to speed up search
  - Because variable names may be very similar (e.g., LOOP1, LOOP2), the selected hash function must perform well with such non-random keys.

Symbol Table (SYMTAB)

- During pass 1, labels are entered into the symbol table as they are encountered in the source program, along with their assigned addresses(from LOCCTR).
- During pass 2, symbols used as operands are looked up in SYNTAB to obtain the addresses to be inserted in the assembled instructions.

- Both passes of the assembler can read the original source program as input.
- Certain info. can/should be communicated between 2 passes.(LOCCTR value, error flag stmts, etc.)
- Pass 1 writes an intermediate file(source statement together with assigned addresses, error indicators etc.)
- This intermediate file is used as input to Pass 2.

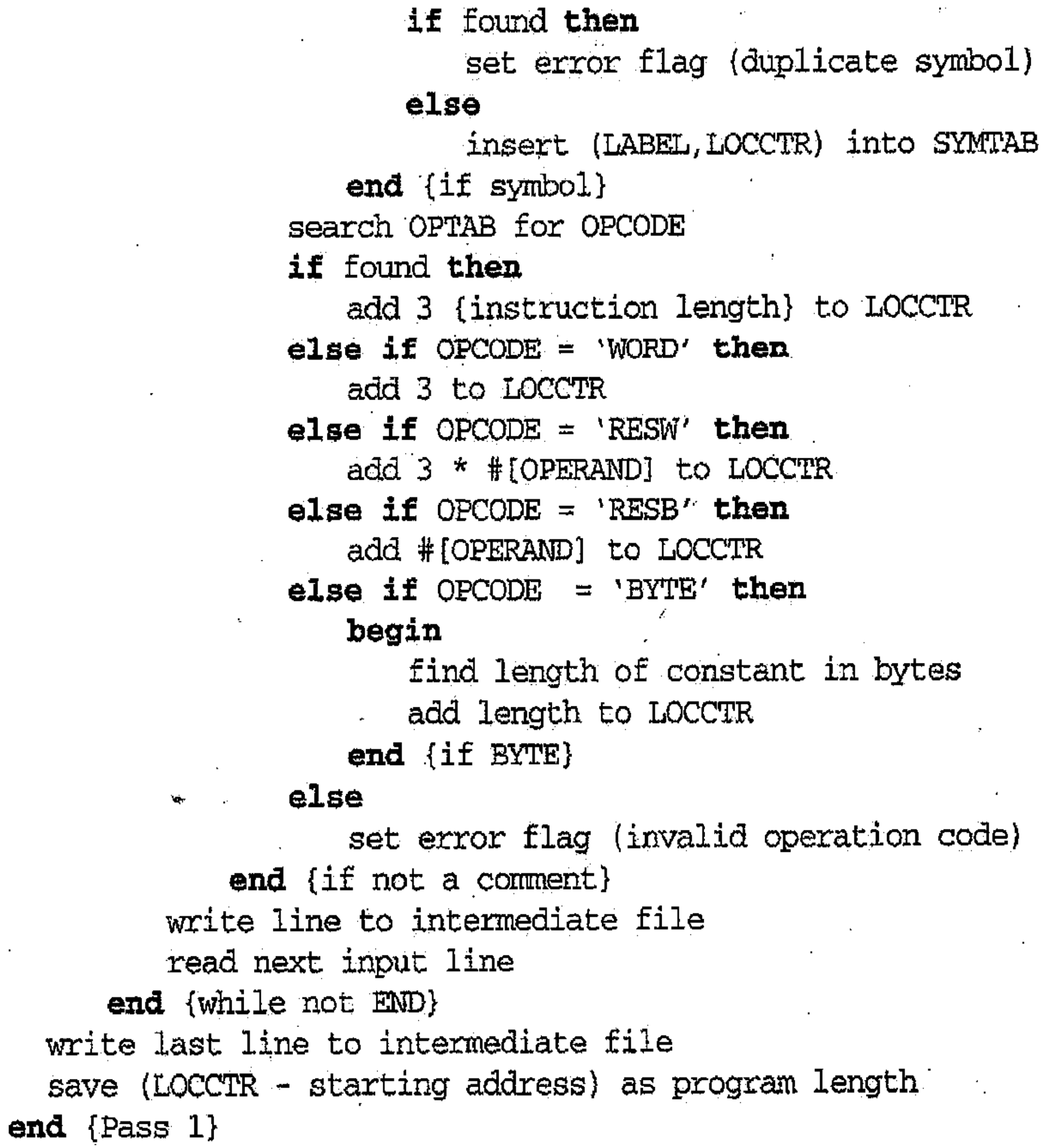
Algorithm for Pass 1 of assembler



Pass 1:

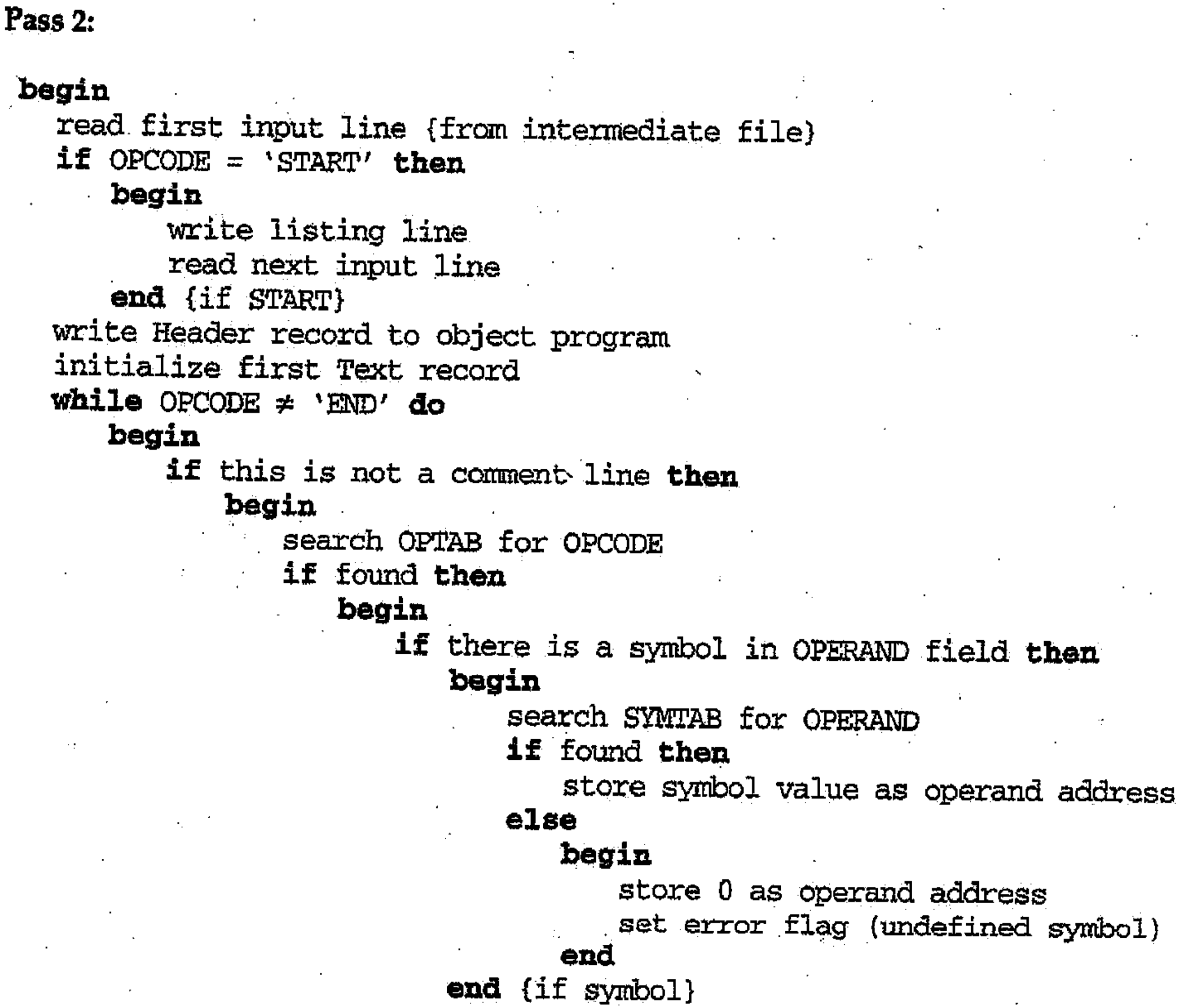
```
begin
   read first input line
   if OPCODE = 'START' then
      begin
         save #[OPERAND] as starting address
         initialize LOCCTR to starting address
         write line to intermediate file
         read next input line
      end {if START}
   else
      initialize LOCCTR to 0
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               if there is a symbol in the LABEL field then
                  begin
                     search SYMTAB for LABEL
```

Algorithm for Pass 1 of assembler(Contd.)



```
                     if found then
                        set error flag (duplicate symbol)
                     else
                        insert (LABEL,LOCCTR) into SYMTAB
                  end {if symbol}
               search OPTAB for OPCODE
               if found then
                  add 3 {instruction length} to LOCCTR
               else if OPCODE = 'WORD' then
                  add 3 to LOCCTR
               else if OPCODE = 'RESW' then
                  add 3 * #[OPERAND] to LOCCTR
               else if OPCODE = 'RESB' then
                  add #[OPERAND] to LOCCTR
               else if OPCODE  = 'BYTE' then
                  begin
                     find length of constant in bytes
                     add length to LOCCTR
                  end {if BYTE}
               else
                  set error flag (invalid operation code)
            end {if not a comment}
         write line to intermediate file
         read next input line
      end {while not END}
   write last line to intermediate file
   save (LOCCTR - starting address) as program length
end {Pass 1}
```

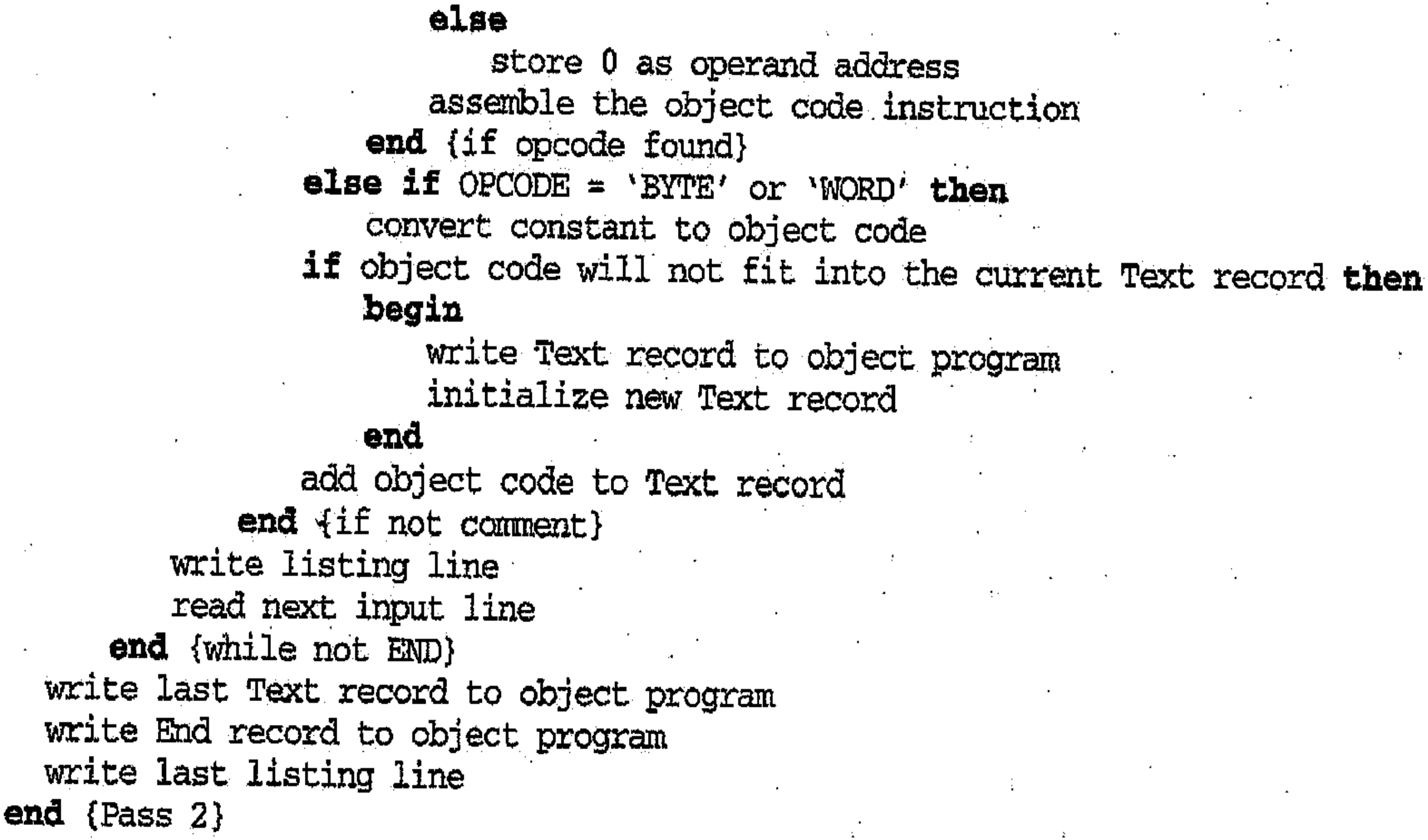
Algorithm for Pass 2 of assembler



Pass 2:

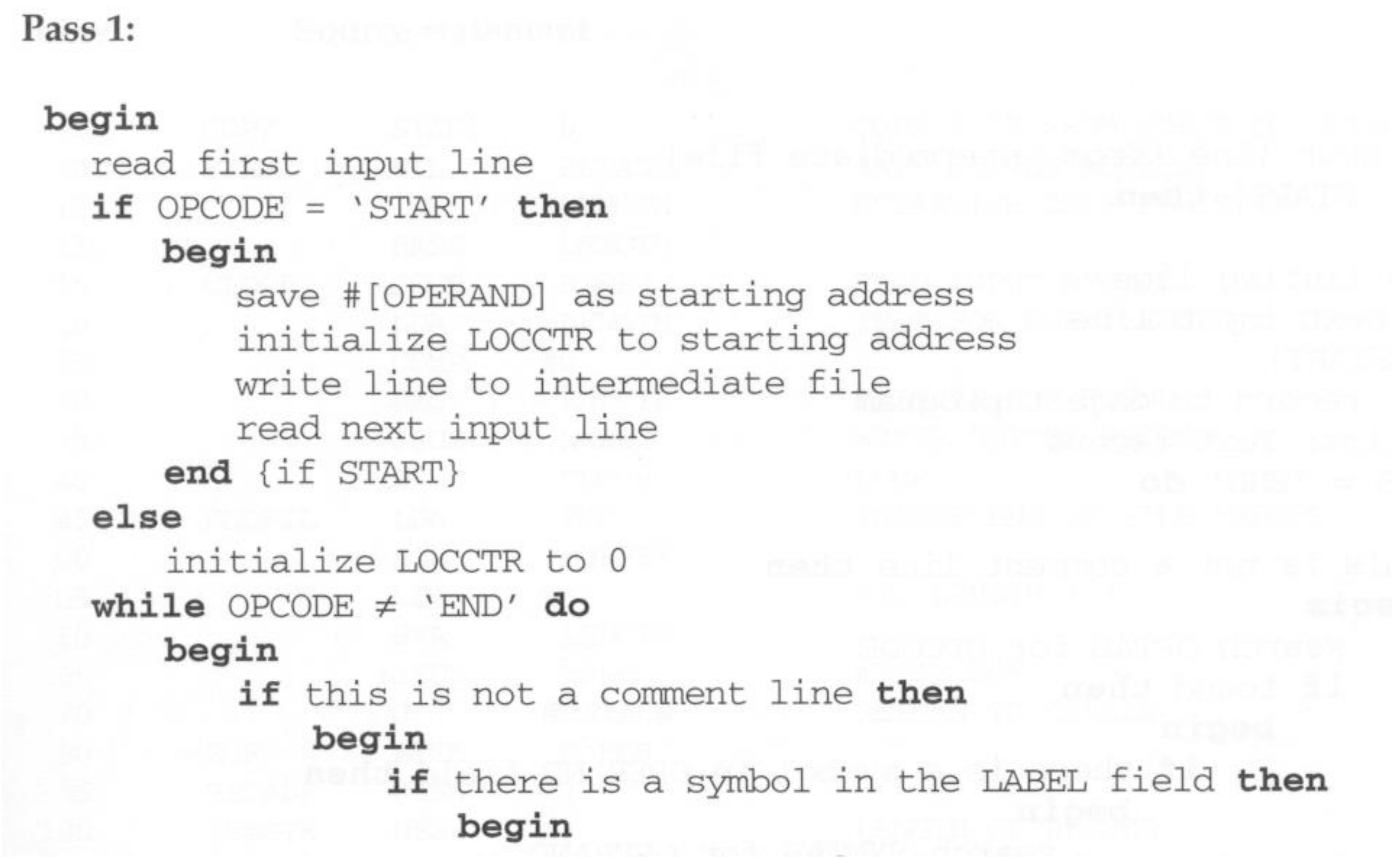
```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end {if START}
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               search OPTAB for OPCODE
               if found then
                  begin
                     if there is a symbol in OPERAND field then
                        begin
                           search SYMTAB for OPERAND
                           if found then
                              store symbol value as operand address
                           else
                              begin
                                 store 0 as operand address
                                 set error flag (undefined symbol)
                              end
                        end {if symbol}
```

Algorithm for Pass 2 of assembler(Contd.)



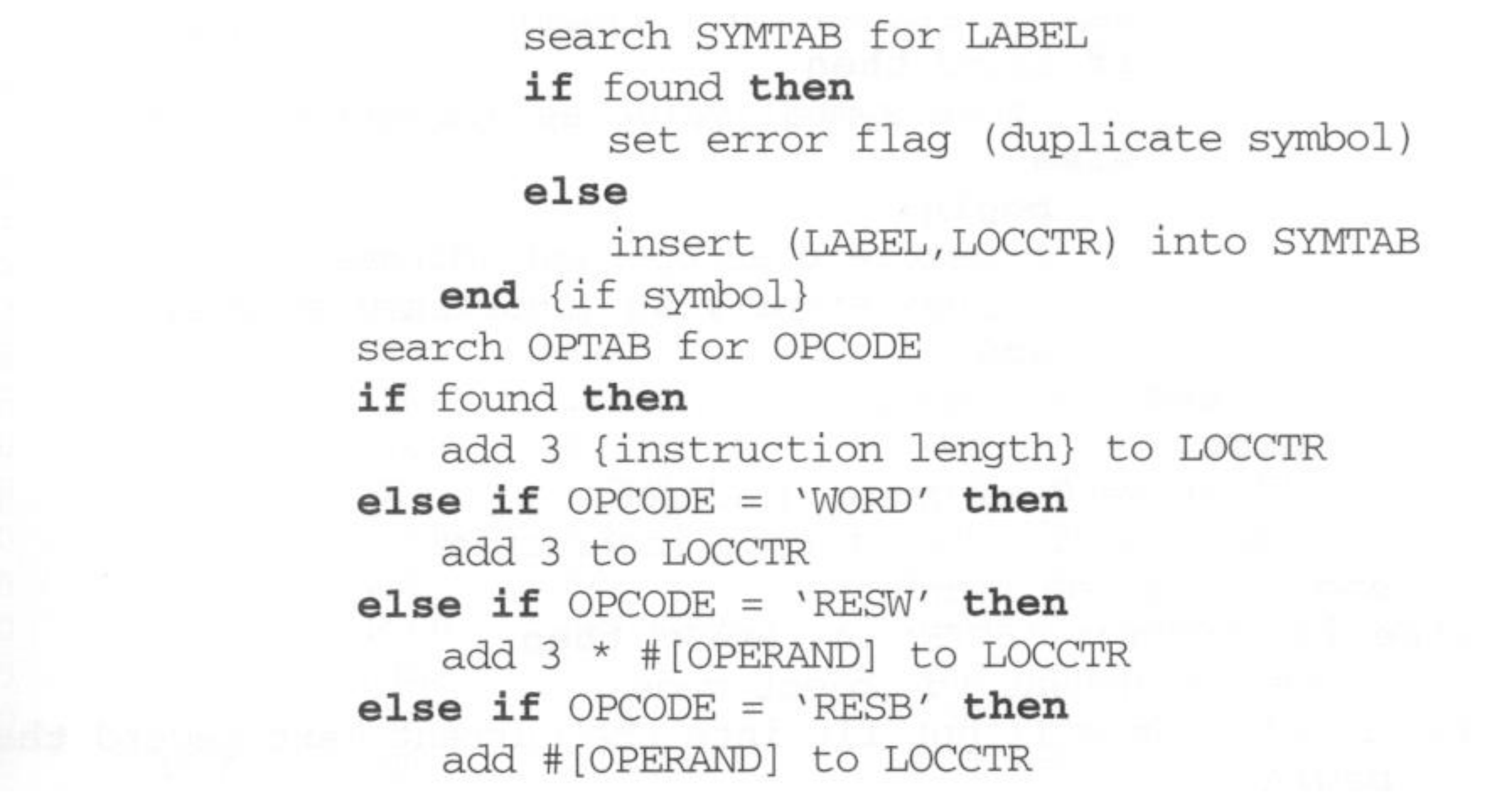
```
                     else
                        store 0 as operand address
                     assemble the object code instruction
                  end {if opcode found}
               else if OPCODE = 'BYTE' or 'WORD' then
                  convert constant to object code
               if 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}
```

The Pseudo Code for Pass 1

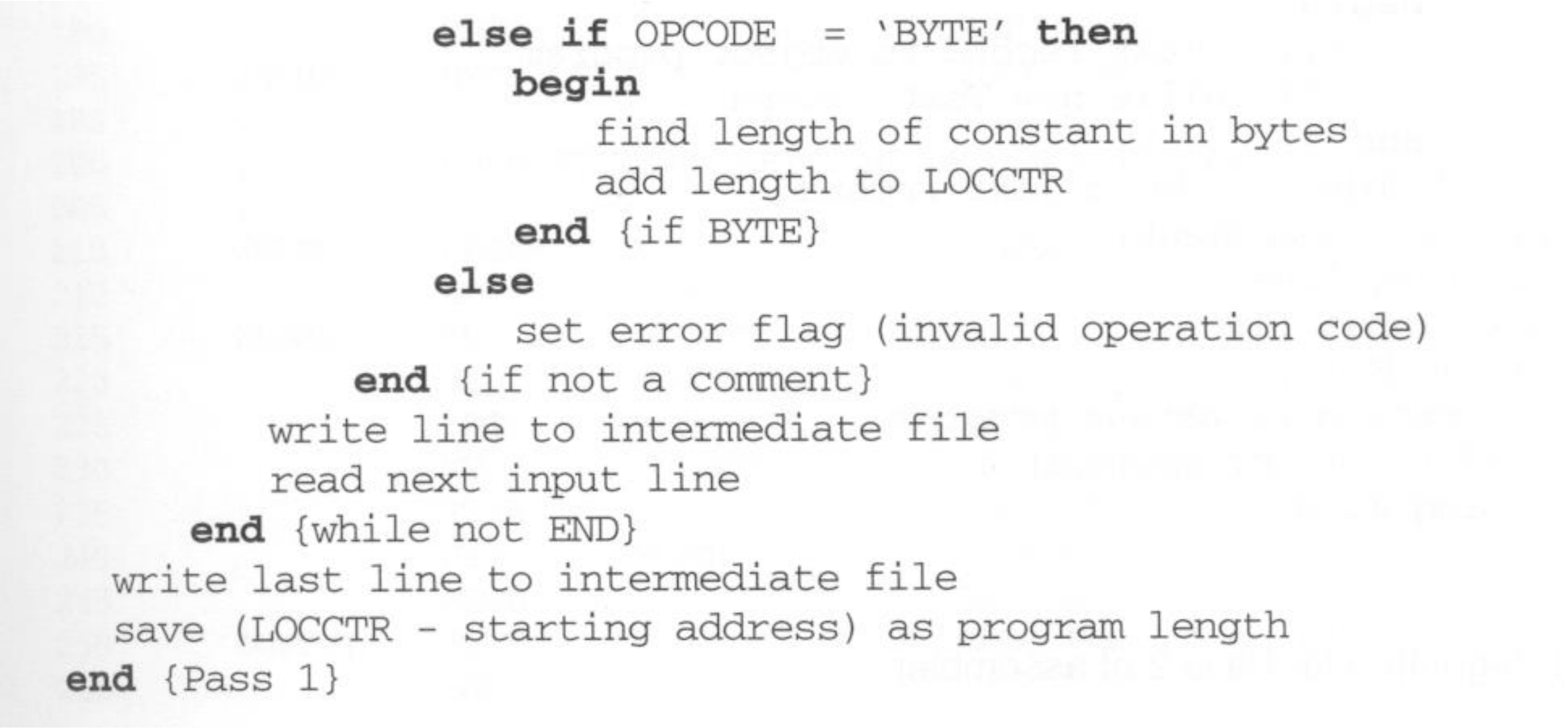


Pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
     begin
        save #[OPERAND] as starting address
        initialize LOCCTR to starting address
        write line to intermediate file
        read next input line
     end {if START}
  else
     initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              if there is a symbol in the LABEL field then
                 begin
```

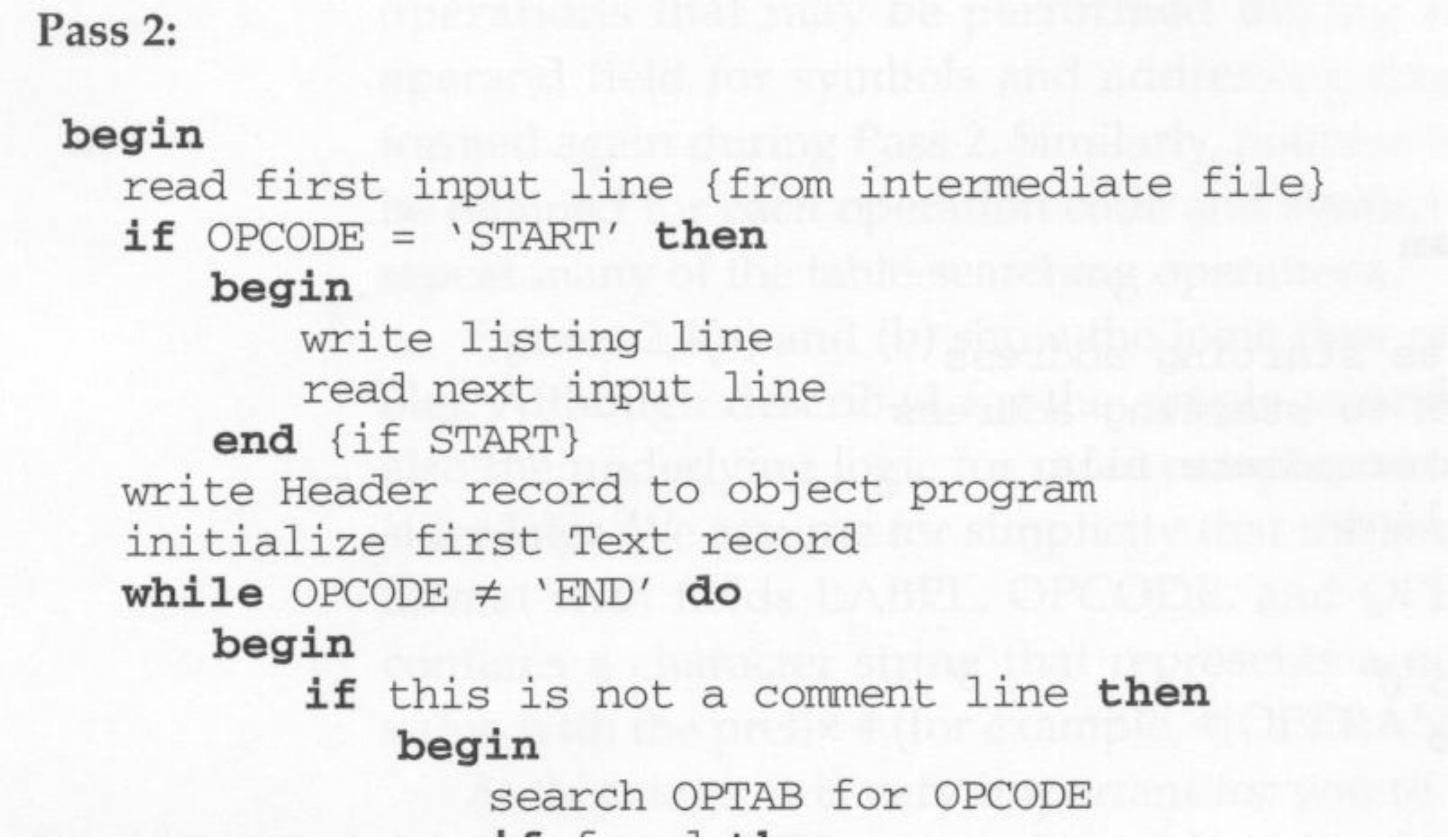


```
        search SYMTAB for LABEL
        if found then
            set error flag (duplicate symbol)
        else
            insert (LABEL,LOCCTR) into SYMTAB
    end {if symbol}
search OPTAB for OPCODE
if found then
    add 3 {instruction length} to LOCCTR
else if OPCODE = 'WORD' then
    add 3 to LOCCTR
else if OPCODE = 'RESW' then
    add 3 * #[OPERAND] to LOCCTR
else if OPCODE = 'RESB' then
    add #[OPERAND] to LOCCTR
```



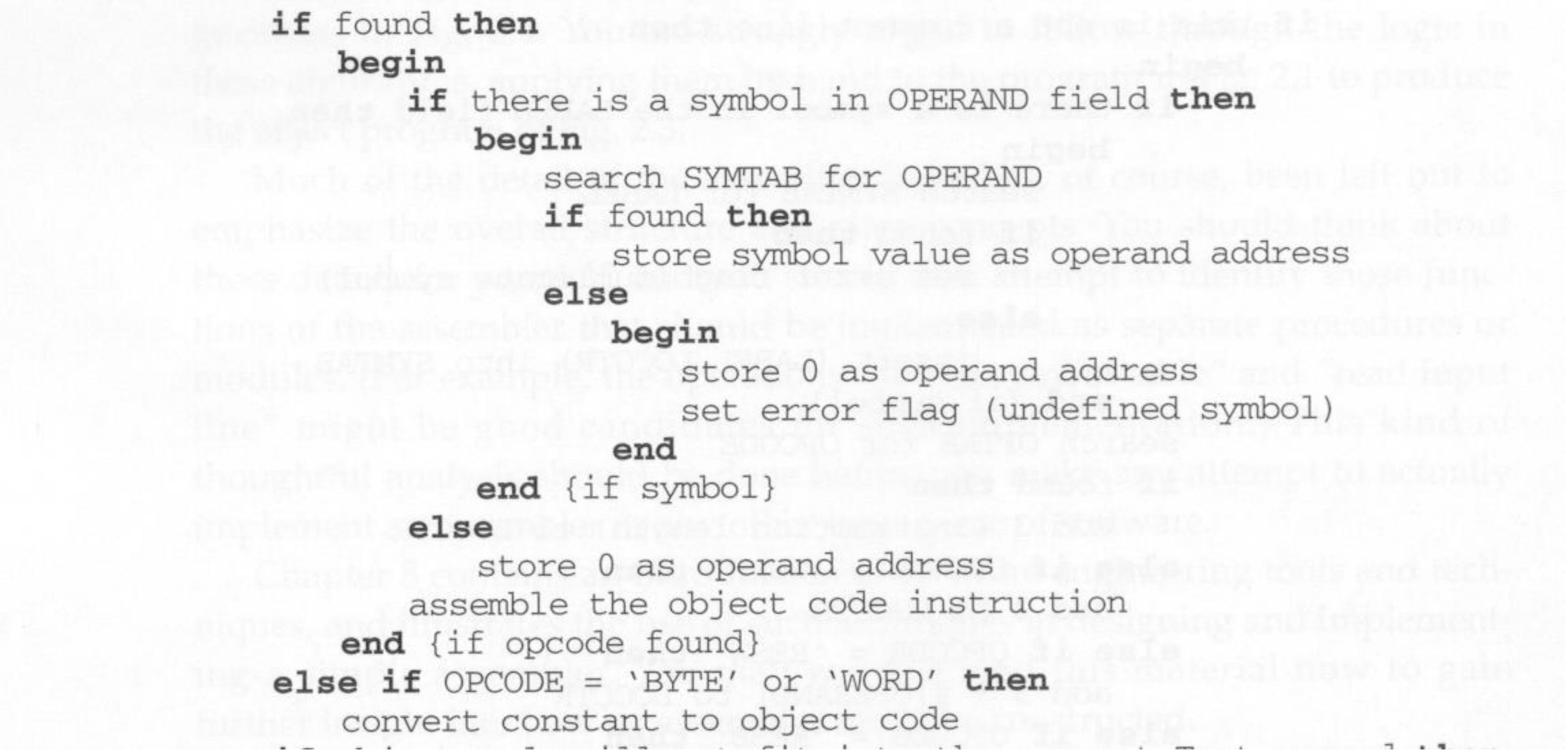
```
          else if OPCODE  = 'BYTE' then
            begin
              find length of constant in bytes
              add length to LOCCTR
            end {if BYTE}
          else
            set error flag (invalid operation code)
        end {if not a comment}
      write line to intermediate file
      read next input line
    end {while not END}
  write last line to intermediate file
  save (LOCCTR - starting address) as program length
end {Pass 1}
```

The Pseudo Code for Pass 2



Pass 2:

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
       begin
           write listing line
           read next input line
       end {if START}
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
       begin
           if this is not a comment line then
               begin
                   search OPTAB for OPCODE
```



```
if found then
   begin
      if there is a symbol in OPERAND field then
         begin
            search SYMTAB for OPERAND
            if found then
               store symbol value as operand address
            else
               begin
                  store 0 as operand address
                  set error flag (undefined symbol)
               end
         end {if symbol}
      else
         store 0 as operand address
      assemble the object code instruction
   end {if opcode found}
else if OPCODE = 'BYTE' or 'WORD' then
   convert constant to object code
```

```
                **if** object code will not fit into the current Text record **then**
                   **begin**
                       write Text record to object program
                       initialize new Text record
                   **end**
                add object code to Text record
            **end** {if not comment}
         write listing line
         read next input line
      **end** {while not END}
  write last Text record to object program
  write End record to object program
  write last listing line
**end** {Pass 2}
```