Figures 2.4(a) and (b) show the logic flow of the two passes of our assembler. Although described for the simple assembler we are discussing, this is also the underlying logic for more complex two-pass assemblers that we consider later. We assume for simplicity that the source lines are written in a fixed format with fields LABEL, OPCODE, and OPERAND. If one of these fields contains a character string that represents a number, we denote its numeric value with the prefix # (for example, #[OPERAND]).

At this stage, it is very important for you to understand thoroughly the algorithms in Fig. 2.4. You are strongly urged to follow through the logic in these algorithms, applying them by hand to the program in Fig. 2.1 to produce the object program of Fig. 2.3.

Much of the detail of the assembler logic has, of course, been left out to emphasize the overall structure and main concepts. You should think about these details for yourself, and you should also attempt to identify those functions of the assembler that should be implemented as separate procedures or modules. (For example, the operations "search symbol table" and "read input line" might be good candidates for such implementation.) This kind of thoughtful analysis should be done before you make any attempt to actually implement an assembler or any other large piece of software.

Chapter 8 contains an introduction to software engineering tools and techniques, and illustrates the use of such techniques in designing and implementing a simple assembler. You may want to read this material now to gain further insight into how an assembler might be constructed.

2.2 MACHINE-DEPENDENT ASSEMBLER FEATURES

In this section, we consider the design and implementation of an assembler for the more complex XE version of SIC. In doing so, we examine the effect of the extended hardware on the structure and functions of the assembler. Many real machines have certain architectural features that are similar to those we consider here. Thus our discussion applies in large part to these machines as well as to SIC/XE.

Figure 2.5 shows the example program from Fig. 2.1 as it might be rewritten to take advantage of the SIC/XE instruction set. In our assembler language, indirect addressing is indicated by adding the prefix @ to the operand (see line 70). Immediate operands are denoted with the prefix # (lines 25, 55, 133). Instructions that refer to memory are normally assembled using either the program-counter relative or the base relative mode. The assembler directive BASE (line 13) is used in conjunction with base relative addressing. (See Section 2.2.1 for a discussion and examples.) If the displacements required

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}
```

Figure 2.4(a) Algorithm for Pass 1 of assembler.