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.

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)
```

Figure 2.4(b) Algorithm for Pass 2 of assembler.

```
begin
  block number = 0 LOCCTR[i] = 0 for all i
  read the first input line
  if OPCODE = 'START' then
  begin
    write line to intermediate file
    read next input line
  end {if START}
  while OPCODE ≠ 'END' do
  if OPCODE = 'USE'
  begin
    if there is no OPEREND name then
      set block name as default
    else block name as OPERAND name
    if there is no entry for block name then
      insert (block name, block number ++) in block tabl
    i = block number for block name
    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[i]) into SYMTAB
        end {if symbol}
      Search OPTAB for OPCODE
      if found then
        add 3 instruction length to LOCCTR[i]
      else if OPCODE = 'WORD' then
        add 3 to LOCCTR[i]
      else if OPCODE = 'RESW' then
        add 3 * #[OPERAND] to LOCCTR[i]
      else if OPCODE = 'RESB' then
        add #[OPERAND] to LOCCTR[i]
      else if OPCODE = 'BYTE' then
      begin
        find length of constant in bytes
        add length to LOCCTR[i]
      end {if byte}
else
```

Figure 2.12(b) Pass 1 of program blocks.

```
    set error flag
    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 Length[i] as LOCCTR[i] for all i
Address[o] = starting address
Address[i] = address(i - 1) + Length(i - 1)
             [for i = 1 to max(block number)]
insert(address[i], Length[i]) in block table for all i
end {Pass 1}
```

Figure 2.12(b) (*cont'd*)

```
If OPCODE = 'USE' then
  set block number for block name with OPERAND field
  search SYMTAB for OPERAND
  store symbol value + address [block number] as operand address
end {Pass 2}
```

Figure 2.12(c) Pass 2 of program blocks.

```
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialize LOCCTR as starting address
      read next input line
    end {if START}
  else
    initialize LOCCTR to 0
while OPCODE ≠ 'END' do
    begin
      if there 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
              begin
                  if symbol value as null
                  set symbol value as LOCCTR and search
                    the linked list with the corresponding
                    operand
                  PTR addresses and generate operand
                    addresses as corresponding symbol
                    values
                  set symbol value as LOCCTR in symbol
                    table and delete the linked list
              end
              else
                insert (LABEL, LOCCTR) into SYMTAB
          end
            search OPTAB for OPCODE
              if found then
                begin
                  search SYMTAB for OPERAND address
              if found then
                if symbol value not equal to null then
                  store symbol value as OPERAND address
                else
                  insert at the end of the linked list
                    with a node with address as LOCCTR
                else
                  insert (symbol name, null)
```

Figure 2.19(c) Algorithm for One pass assembler.

```
            add 3 to LOCCTR
          end
          else if OPCODE = 'WORD' then
            add 3 to LOCCTR & convert comment to
             object code
          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
              convert constant to object code
            end
        if object code will not fit into current
          text record then
          begin
            write text record to object program
            initialize new text record
          end
        add object code to Text record
      end
    write listing line
    read next input line
  end
write last Text record to object program
write End record to object program
write last listing line
end {Pass 1}
```

Figure 2.19(c) *(cont'd)*

```
begin
    read Header record
    verify program name and length
    read first Text record
    while record type ≠ 'E' do
        begin
            {if object code is in character form, convert into
                internal representation}
            move object code to specified location in memory
            read next object program record
        end
    jump to address specified in End record
end
```

Figure 3.2 Algorithm for an absolute loader.

```
begin
    get PROGADDR from operating system
    while not end of input do
        begin
            read next record
            while record type ≠ 'E' do
                begin
                    read next input record
                    while record type = 'T' then
                        begin
                            move object code from record to location
                            PROGADDR + specified address
                        end
                    while record type = 'M'
                        add PROGADDR at the location PROGADDR +
                            specified address
                end
        end
end
```

Figure 3.6 SIC/XE relocation loader algorithm.

Pass 1:

```
begin
get PROGADDR from operating system
set CSADDR to PROGADDR {for first control section}
while not end of input do
    begin
        read next input record {Header record for control section}
        set CSLTH to control section length
        search ESTAB for control section name
        if found then
            set error flag {duplicate external symbol}
        else
            enter control section name into ESTAB with value CSADDR
        while record type ≠ 'E' do
            begin
                read next input record
                if record type = 'D' then
                    for each symbol in the record do
                        begin
                            search ESTAB for symbol name
                            if found then
                                set error flag (duplicate external symbol)
                            else
                                enter symbol into ESTAB with value
                                    (CSADDR + indicated address)
                        end {for}
            end {while ≠ 'E'}
        add CSLTH to CSADDR {starting address for next control section}
    end {while not EOF}
end {Pass 1}
```

Figure 3.13(a) Algorithm for Pass 1 of a linking loader.

Pass 2:

```
begin
set CSADDR to PROGADDR
set EXECADDR to PROGADDR
while not end of input do
   begin
      read next input record {Header record}
      set CSLTH to control section length
      while record type ≠ 'E' do
         begin
            read next input record
            if record type = 'T' then
               begin
                  {if object code is in character form, convert
                     into internal representation}
                  move object code from record to location
                     (CSADDR + specified address)
               end {if 'T'}
            else if record type = 'M' then
               begin
                  search ESTAB for modifying symbol name
                  if found then
                     add or subtract symbol value at location
                        (CSADDR + specified address)
                  else
                     set error flag (undefined external symbol)
               end {if 'M'}
         end {while ≠ 'E'}
      if an address is specified {in End record} then
         set EXECADDR to (CSADDR + specified address)
      add CSLTH to CSADDR
   end {while not EOF}
jump to location given by EXECADDR {to start execution of loaded program}
end {Pass 2}
```

Figure 3.13(b) Algorithm for Pass 2 of a linking loader.

```
begin {macro processor}
    EXPANDING := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
end {macro processor}


procedure PROCESSLINE
    begin
        search NAMTAB for OPCODE
        if found then
            EXPAND
        else if OPCODE = 'MACRO' then
            DEFINE
        else write source line to expanded file
    end {PROCESSLINE}


procedure DEFINE
    begin
        enter macro name into NAMTAB
        enter macro prototype into DEFTAB
        LEVEL := 1
        while LEVEL > 0 do
            begin
                GETLINE
                if this is not a comment line then
                    begin
                        substitute positional notation for parameters
                        enter line into DEFTAB
                        if OPCODE = 'MACRO' then
                            LEVEL := LEVEL + 1
                        else if OPCODE = 'MEND' then
                            LEVEL := LEVEL - 1
                    end {if not comment}
            end {while}
        store in NAMTAB pointers to beginning and end of definition
    end {DEFINE}
```

nested macros

Figure 4.5 Algorithm for a one-pass macro processor.

```
procedure EXPAND
   begin
      EXPANDING  := TRUE
      get first line of macro definition (prototype) from DEFTAB
      set up arguments from macro invocation in ARGTAB
      write macro invocation to expanded file as a comment
      while not end of macro definition do
         begin
            GETLINE
            PROCESSLINE
         end (while)
      EXPANDING := FALSE
   end (EXPAND)


procedure GETLINE
   begin
      if EXPANDING then
         begin
            get next line of macro definition from DEFTAB
            substitute arguments from ARGTAB for positional notation
         end (if)
      else
         read next line from input file
   end (GETLINE)
```

Figure 4.5 *(cont'd)*