MODULE 2 - TWO PASS 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
                     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}
```

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}
```

MODULE 4

The algorithm for an absolute loader

```
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
```

Algorithm for the Bootstrap Loader

The algorithm for the bootstrap loader is as follows:

Begin

X = 0x80 (the address of the next memory location to be loaded.)

Loop

A ← GETC (and convert it from the ASCII character code to the value of the hexadecimal digit) save the value in the high-order 4 bits of S

A ← GETC

combine the value to form one byte A ← (A+S) store the value (in A) to the address in register X

X ← X + 1. .

End

It uses a subroutine GETC, which is

GETC A ← read one character

if A = 0x04 then jump to 0x80

if A < 48 then GETC

A ← A-48 (0x30)

if A < 10 then return

2 PASS LINKING LOADER

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 3.11(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 3.11(b) Algorithm for Pass 2 of a linking loader.

MODULE 5 – ONE PASS MACRO PROCESSOR

Algorithm

```
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}
```

```
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)
```