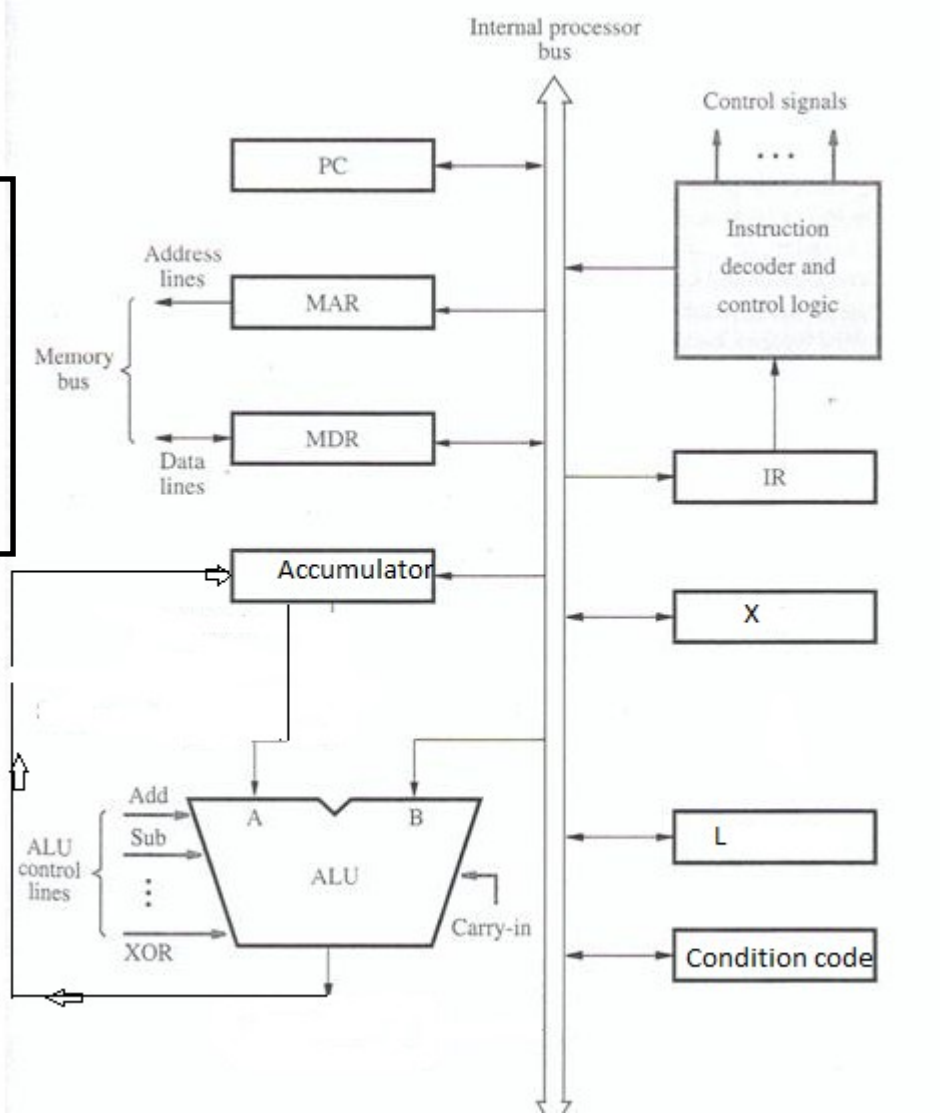
# Assembler design

# Hypothetical machine

## Processor architecture




Main memory

Size=64KB

Memory Address =2 Bytes

Simplified Instructional Computer (SIC)

# Word length

Accumulator,
Registers X, L

| Byte | Byte | Byte |
|---|---|---|

3 Bytes

Address:
2 bytes ⟷

Main Memory

| | | | |
|---|---|---|---|
| 4000 | Byte | Byte | Byte |
| 4003 | | | |
| 4006 | | | |
| 4009 | | | |
| 4012 | | | |

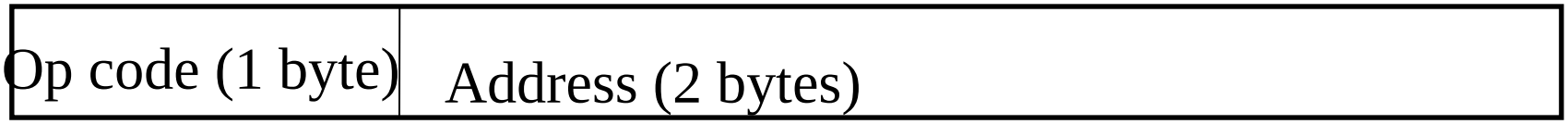
# Instruction set of SIC

| Mnemonic | Format | Opcode | Effect |
|---|---|---|---|
| ADD m | 3/4 | 18 | A ← (A) + (m..m+2) |
| AND m | 3/4 | 40 | A ← (A) & (m..m+2) |
| COMP m | 3/4 | 28 | (A) : (m..m+2) |
| DIV m | 3/4 | 24 | A ← (A) / (m..m+2) |
| J m | 3/4 | 3C | PC ← m |
| JEQ m | 3/4 | 30 | PC ← m if CC set to = |
| JGT m | 3/4 | 34 | PC ← m if CC set to > |
| JLT m | 3/4 | 38 | PC ← m if CC set to < |
| JSUB m | 3/4 | 48 | L ← (PC); PC ← m |
| LDA m | 3/4 | 00 | A ← (m..m+2) |
| LDCH m | 3/4 | 50 | A [rightmost byte] ← (m) |
| LDL m | 3/4 | 08 | L ← (m..m+2) |
| LDX m | 3/4 | 04 | X ← (m..m+2) |

| Mnemonic | Format | Opcode | Effect | Notes |
|---|---|---|---|---|
| OR m | 3/4 | 44 | A ← (A) \| (m..m+2) | |
| RD m | 3/4 | D8 | A [rightmost byte] ← data from device specified by (m) | |
| RSUB | 3/4 | 4C | PC ← (L) | |
| STA m | 3/4 | 0C | m..m+2 ← (A) | |
| STCH m | 3/4 | 54 | m ← (A) [rightmost byte] | |
| STL m | 3/4 | 14 | m..m+2 ← (L) | |
| STSW m | 3/4 | E8 | m..m+2 ← (SW) | P |
| STX m | 3/4 | 10 | m..m+2 ← (X) | |
| SUB m | 3/4 | 1C | A ← (A) – (m..m+2) | |

| Mnemonic | Format | Opcode | Effect | Notes |
|---|---|---|---|---|
| | | | | |
| | | | | |
| TD m | 3/4 | E0 | Test device specified by (m) | |
| | | | | |
| TIX m | 3/4 | 2C | X ← (X) + 1; (X): (m..m+2) | |
| | | | | |
| WD m | 3/4 | DC | Device specified by (m) ← (A) [rightmost byte] | |

# Instruction format



# Sample code (data movement)

| LABEL | Instruction | Operand | |
|---|---|---|---|
| TEST | START | 1003 | |
| FIRST | LDA | FIVE | |
| | STA | ALPHA | |
| ALPHA | RESW 1 | /*symbolic variable*/ | |
| FIVE | WORD | 5 | /*symbolic constant, Literal */ |
| | END | FIRST | |

**Assembly language program**

| Psuedo Opcode OR Assembler Directives |
|---|
| START |
| RESW |
| WORD, END |

# Sample code (data movement)

| Loc | LABEL | Instruction | Operand | Object code |
|---|---|---|---|---|
| | TEST | START | 1003 | |
| 1003 | FIRST | LDA | FIVE | 001012 |
| 1006 | | STA | ALPHA | 0C1009 |
| 1009 | ALPHA | RESW | 1 | ****** |
| 1012 | FIVE | WORD | 5 | 000005 |
| 1015 | | END | FIRST | |

**Assembly language program**

# Sample code (Arithmetic operation)

```
    LDA ALPHA
    ADD INCR
    SUB ONE
    STA BETA
ONE    WORD    1
ALPHA       RESW    1
BETA   RESW    1
INCR   RESW    1
```

# Assembler

# Sample code (data movement)

| Loc | LABEL | Instruction | Operand | Object code |
|---|---|---|---|---|
| 1003 | TEST | START | 1003 | |
| 1003 | FIRST | LDA | FIVE | 001012 |
| 1006 | | STA | ALPHA | 0C1009 |
| 1009 | ALPHA | RESW | 1 | ****** |
| 1012 | FIVE | WORD | 5 | 000005 |
| 1013 | | END | FIRST | |

**Assembly language program**

# How to design an Assembler

## Data structures

### (1) OPTAB

| Instruction | Op code | Length (bytes) |
|---|---|---|
| ADD m | 18 | 3 |
| LDA m | 00 | 3 |
| | | |
| | | |

### (3) Symbol Table

| LABEL | Address (LOC value) |
|---|---|
| FIRST | 1003 |
| ALPHA | 1009 |
| FIVE | 1012 |

### (2) Location counter (LOC)

# Data Structu

| Instruction | Op code | Length (bytes) |
|---|---|---|
| ADD m | 18 | 3 |
| LDA m | 00 | 3 |
| | | |
| | | |

- Operation Code Table (OPTAB)
  - Used to look up mnemonic operation codes and translate them into machine language equivalents
  - Contains the mnemonic operation code and its machine language equivalent
  - In more complex assemblers, contains information like instruction format and length

# Data Struc

| LABEL | Address (LOC value) |
|---|---|
| FIRST | 1003 |
| ALPHA | 1009 |
| FIVE | 1012 |

- Symbol Table
  - Used to store values (addresses) assigned to labels
  - Includes the name and value for each label
  - Flags to indicate error conditions, e.g. duplicate definition of labels
  - May contain other info like type or length about the data area or instruction labeled

# Data Structures

- LOCCTR
  - Used to help in the assignment of addresses
  - Initialized to the beginning address specified in the START statement
  - After each source statement is processed, the length of the assembled instruction or data area to be generated is added
  - Gives the address of a label

# How to design an Assembler

Two pass algorithm

Pass 1 (Define symbols):

(a)Assign addresses to all statements (LOC)

(b) Save the addresses assigned to all labels in symbol table

(c) Perform some processing for assembler directives

# How to design an Assembler

Two pass algorithm

Pass 2 (Generate object code):

a. Translate opcode and operands
b. Generate data values for WORD
c. Write object program

# Sample code (data movement)

| Loc | LABEL | Instruction | Operand | Object code |
|---|---|---|---|---|
| 1003 | TEST | START | 1003 | |
| 1003 | FIRST | LDA | FIVE | 001012 |
| 1006 | | STA | ALPHA | 0C1009 |
| 1009 | ALPHA | RESW | 1 | ****** |
| 1012 | FIVE | WORD | 5 | 000005 |
| 1013 | | END | FIRST | |

**Assembly language program**

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.

| 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 | | . | | | |
| 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 | | . | | | |
| 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 | |

**Figure 2.2** Program from Fig. 2.1 with object code.

H | COPY  | 001000 | 00107A
T | 001000 | 1E | 141033 | 482039 | 001036 | …
T | 00101E | 15 | 0C1036 | 482061 | 081033 | …
…
T | 002073 | 07 | 382064 | 4C0000 | 05
E | 001000

Header record:

| | |
|---|---|
| Col. 1 | H |
| Col. 2-7 | Program name |
| Col. 8-13 | Starting address of object program (hexadecimal) |
| Col. 14-19 | Length of object program in bytes (hexadecimal) |

Text record:

| | |
|---|---|
| Col. 1 | T |
| Col. 2-7 | Starting address for object code in this record (hexadecimal) |
| Col. 8-9 | Length of object code in this record in bytes (hexadecimal) |
| Col. 10 – 69 | Object code, represented in hexadecimal. (69-10+1)/6=10 instructions |

End record:

| | |
|---|---|
| Col. 1 | E |
| Col. 2-7 | Address of first executable instruction in object program (hexadecimal) |

Text record:

| | |
|---|---|
| Col. 1 | T |
| Col. 2–7 | Starting address for object code in this record (hexadecimal) |
| Col. 8–9 | Length of object code in this record in bytes (hexadecimal) |
| Col. 10–69 | Object code, represented in hexadecimal (2 columns per byte of object code) |

End record:

| | |
|---|---|
| Col. 1 | E |
| Col. 2–7 | Address of first executable instruction in object program (hexadecimal) |

To avoid confusion, we have used the term *column* rather than *byte* to refer to positions within object program records. This is not meant to imply the use of any particular medium for the object program.

Figure 2.3 shows the object program corresponding to Fig. 2.2, using this format. In this figure, and in the other object programs we display, the symbol ^ is used to separate fields visually. Of course, such symbols are not present in the actual object program. Note that 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. (Chapter 3 contains a detailed discussion of the operation of the loader.)

We can now give a general description of the functions of the two passes of our simple assembler.

**Pass 1** (define symbols):

1. Assign addresses to all statements in the program.
2. Save the values (addresses) assigned to all labels for use in Pass 2.

```
HCOPY  ^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.

# PASS -1

```
begin
  read first input line
  if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialized LOCCTR to starting address
      write line to intermediate file
      read next input line
    end {if START}
  else
    initialized LOCCTR to 0
```

| Loc | | Source Statement | | |
|---|---|---|---|---|
| 1000 | COPY | START | 1000 | COPY FILE FROM INPUT TO OUTPUT |
| 1000 | FIRST | STL | RETADR | SAVE RETURN ADDRESS |
| 1003 | CLOOP | JSUB | RDREC | READ INPUT RECORD |
| 1006 | | LDA | LENGTH | TEST FOR EOF (LENGTH = 0) |
| 1009 | | COMP | ZERO | |
| 100C | | JEQ | ENDFIL | EXIT IF EOF FOUND |
| 1000F | | JSUB | WRREC | WRITE OUTPUT RECORD |
| 1012 | | J | CLOOP | LOOP |
| 1015 | ENDFIL | LDA | EOF | INSERT END OF FILE MARKER |
| 1018 | | STA | BUFFER | |
| 101B | | LDA | THREE | SET LENGTH = 3 |
| 101E | | STA | LENGTH | |
| 1021 | | JSUB | WRREC | WRITE EOF |
| 1024 | | LDL | RETADR | GET RETURN ADDRESS |
| 1027 | | RSUB | | RETURN TO CALLER |
| 102A | EOF | | BYTE | C’EOF’ |
| 102D | THREE | WORD | 3 | |
| 1030 | ZERO | | WORD | 0 |
| 1033 | RETADR | RESW | 1 | |

```
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 lengh} 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
```

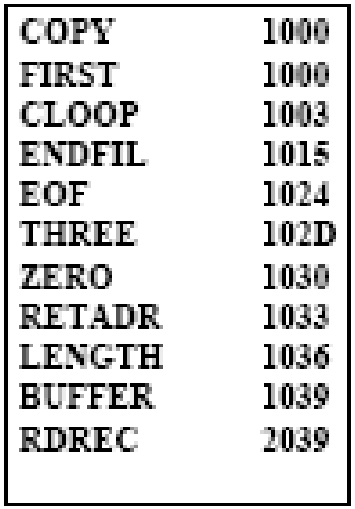
```
1000  COPY      START   1000        COPY FILE FROM INPUT TO
      OUTPUT
1000 FIRST      STL     RETADR      SAVE RETURN ADDRESS

1003 CLOOP      JSUB    RDREC       READ INPUT RECORD

1006            LDA     LENGTH      TEST FOR EOF (LENGTH = 0)

1009            COMP    ZERO

100C            JEQ     ENDFIL      EXIT IF EOF FOUND

1000F           JSUB    WRREC       WRITE OUTPUT RECORD

1012            J       CLOOP       LOOP

1015 ENDFIL     LDA     EOF         INSERT END OF FILE MARKER

1018            STA     BUFFER

101B            LDA     THREE       SET LENGTH = 3

101E            STA     LENGTH

1021            JSUB    WRREC       WRITE EOF

1024            LDL     RETADR      GET RETUR

1027            RSUB                RETURN TO

102A EOF                BYTE        C'EOF'

102D THREE      WORD    3

1030 ZERO               WORD        0

1033 RETADR     RESW    1

1036 LENGTH     RESW    1           LENGTH OF
```

```
COPY      1000
FIRST     1000
CLOOP     1003
ENDFIL    1015
EOF       1024
THREE     102D
ZERO      1030
RETADR    1033
LENGTH    1036
BUFFER    1039
RDREC     2039
```

# PASS -2

```
begin
  read first input file {from intermediate file}
  if OPCODE = 'START' then
    begin
      write listing line
      read next input line
    end {if START}
  write header record to object program
  initialized first Text record
```

| Loc | Source Statement | | | Object Code |
|---|---|---|---|---|
| 1000 | COPY | START | 1000 | COPY FILE FROM INPUT TO OUTPUT |
| 1000 | FIRST | STL | RETADR | SAVE RETURN ADDRESS |
| | | 141033 | | |
| 1003 | CLOOP | JSUB | RDREC | READ INPUT RECORD |
| | | 482039 | | |
| 1006 | | LDA | LENGTH | TEST FOR EOF (LENGTH = 0) |
| | | 001036 | | |
| 1009 | | COMP | ZERO | |
| | | 281030 | | |
| 100C | | JEQ | ENDFIL | EXIT IF EOF FOUND |
| | | 301015 | | |
| 1000F | | JSUB | WRREC | WRITE OUTPUT RECORD |
| | | 482061 | | |
| 1012 | | J | CLOOP | LOOP |
| | | 3C1003 | | |
| 1015 | ENDFIL | LDA | EOF | INSERT END OF FILE MARKER |
| | | 00102A | | |
| 1018 | | STA | BUFFER | |
| | | 0C1039 | | |
| 101B | | LDA | THREE | SET LENGTH = 3 |
| | | 00102D | | |
| 101E | | STA | LENGTH | |
| | | 0C1036 | | |
| 1021 | | JSUB | WRREC | WRITE EOF |
| | | 482061 | | |
| 1024 | | LDL | RETADR | GET RETUR |
| | | 081033 | | |
| 1027 | | RSUB | | RETURN TO |
| | | 4C000 | | |
| 102A | EOF | | BYTE | C’EOF’ |
| | | 454F46 | | |
| 102D | THREE | WORD | 3 | |
| | | 000003 | | |
| 1030 | ZERO | | WORD | 0 |
| | | 000000 | | |
| 1033 | RETADR | RESW | 1 | |
| 1036 | LENGTH | RESW | 1 | LENGTH OF |



```
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 not fit into the current Text record then
          begin
            write Text record to object program
            initialized 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}
```

```
1000   COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
1000 FIRST       STL     RETADR      SAVE RETURN ADDRESS
       141033
1003 CLOOP       JSUB    RDREC       READ INPUT RECORD
       482039
1006             LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
       001036
1009             COMP    ZERO
       281030
100C             JEQ     ENDFIL      EXIT IF EOF FOUND
       301015
1000F            JSUB    WRREC       WRITE OUTPUT RECORD
       482061
1012             J       CLOOP       LOOP
       3C1003
1015 ENDFIL      LDA     EOF         INSERT END OF FILE MARKER
       00102A
1018             STA     BUFFER
       0C1039
101B             LDA     THREE       SET LENGTH = 3
       00102D
101E             STA     LENGTH
       0C1036
1021             JSUB    WRREC       WRITE EOF
       482061
1024             LDL     RETADR      GET RETUR
       081033
1027             RSUB                RETURN TO
       4C000
102A EOF                 BYTE        C'EOF'
       454F46
102D THREE       WORD    3
       000003
1030 ZERO                WORD        0
       000000
1033 RETADR      RESW    1
1036 LENGTH      RESW    1           LENGTH OF
1039 BUFFER      RESB    4096        4096-BYTE
.
```

| | |
|---|---|
| COPY | 1000 |
| FIRST | 1000 |
| CLOOP | 1003 |
| ENDFIL | 1015 |
| EOF | 1024 |
| THREE | 102D |
| ZERO | 1030 |
| RETADR | 1033 |
| LENGTH | 1036 |
| BUFFER | 1039 |
| RDREC | 2039 |

write last Text record to object program
write End record to object program
write last listing line
end

```
1000   COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
1000 FIRST       STL     RETADR      SAVE RETURN ADDRESS
       141033
1003 CLOOP       JSUB    RDREC       READ INPUT RECORD
       482039
1006             LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
       001036
1009             COMP    ZERO
       281030
100C             JEQ     ENDFIL      EXIT IF EOF FOUND
       301015
1000F            JSUB    WRREC       WRITE OUTPUT RECORD
       482061
1012             J       CLOOP       LOOP
       3C1003
1015 ENDFIL      LDA     EOF         INSERT END OF FILE MARKER
       00102A
1018             STA     BUFFER
       0C1039
101B             LDA     THREE       SET LENGTH = 3
       00102D
101E             STA     LENGTH
       0C1036
1021             JSUB    WRREC       WRITE EOF
       482061
1024             LDL     RETADR      GET RETURN    ADDRESS
       081033
1027             RSUB                RETURN TO CALLER
       4C000
102A EOF                 BYTE        C'EOF'
       454F46
102D THREE       WORD    3
       000003
1030 ZERO                WORD        0
       000000
1033 RETADR      RESW    1
1036 LENGTH      RESW    1           LENGTH OF RECORD
1039 BUFFER      RESB    4096        4096-BYTE  BUFFER AREA
.
```

H | COPY  | 001000 | 00107A
T | 001000 | 1E | 141033 | 482039 | 001036 | …
T | 00101E | 15 | 0C1036 | 482061 | 081033 | …
…
T | 002073 | 07 | 382064 | 4C0000 | 05
E | 001000

Header record:

| | |
|---|---|
| Col. 1 | H |
| Col. 2-7 | Program name |
| Col. 8-13 | Starting address of object program (hexadecimal) |
| Col. 14-19 | Length of object program in bytes (hexadecimal) |

Text record:

| | |
|---|---|
| Col. 1 | T |
| Col. 2-7 | Starting address for object code in this record (hexadecimal) |
| Col. 8-9 | Length of object code in this record in bytes (hexadecimal) |
| Col. 10 – 69 | Object code, represented in hexadecimal. (69-10+1)/6=10 instructions |

End record:

| | |
|---|---|
| Col. 1 | E |
| Col. 2-7 | Address of first executable instruction in object program (hexadecimal) |