System programming

# Assemblers

REFERENCES:
SYSTEM SOFTWAREBY **LELAND L. BECK**
SYSTEM SOFTWARE BY **SANTANU CHATTOPADHAY**
SYSTEM PROGRAMMING BY **SRIMANTA PAL**

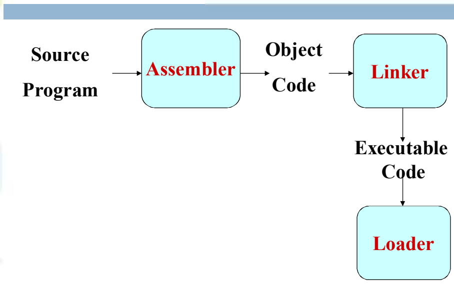
System programming

## OUTLINE

- Basic Assembler Functions
- Machine-dependent Assembler Features
- Machine-independent Assembler Features
- Assembler Design Options
- Appropriate data structures

System programming

## ROLE OF ASSEMBLER




- Assembler is the tool to convert an assembly language program into machine language one, understandable by the processor that executes it

System programming

## INTRODUCTION TO ASSEMBLERS

- Fundamental functions
  - translating mnemonic operation codes to their machine language equivalents
  - assigning machine addresses to symbolic labels
- Machine dependency
  - different machine instruction formats and codes

System programming

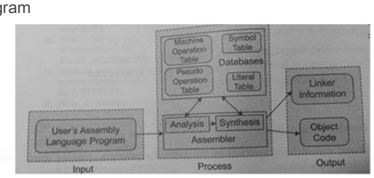
## ASSEMBLERS

- To Design and implementation of an assembler needs the following:
  - Development of finite specification of the machine language
  - Development of finite specification of the Assembly language
  - Implementation of mapping the that performs
    - The reading of an assembly language program, statement by statement
    - The Mapping of each statement into machine language representation and
    - The generation of machine language module

System programming

## BASIC TASK OF AN ASSEMBLER

- Noting the external and user defined symbol
  - Passing these symbol to the loader and linker
  - Finding the programs containing the address( or values)
    - If not known
  - Generating information for the loader, loading this information into the memory, and placing these values of the symbols in the calling program




The working Principal of an assembly Process

System programming

## LIMITATIONS OF AN ASSEMBLER

- Problems in manual assembly
  - Converting Mnemonic codes
  - Converting decimal numbers
  - Assigning address to program or sub program
  - Counting number of data words
  - Assigning addresses in memory
- General problems in Assembler
  - Designed for particular computers not generic
  - Assembler must be loaded in computer memory…additional memory is required for that

System programming

## EXAMPLE 2.1

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

*Forward reference*

*Call subroutine*

*code*

```
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
```

System programming

*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*

System programming

*Subroutine entry point*

```
195  .
200  .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205  .
210  WRREC    LDX     ZERO      CLEAR LOOP COUNTER
215  WLOOP    TD      OUTPUT    TEST OUTPUT DEVICE
220           JEQ     WLOOP     LOOP UNTIL READY
225           LDCH    BUFFER,X  GET CHARACTER FROM BUFFER
230           WD      OUTPUT    WRITE CHARACTER
235           TIX     LENGTH    LOOP UNTIL ALL CHARACTERS
240           JLT     WLOOP       HAVE BEEN WRITTEN
245           RSUB              RETURN TO CALLER
250  OUTPUT   BYTE    X'05'     CODE FOR OUTPUT DEVICE
255           END     FIRST
```

*Subroutine return point*

System programming

## 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

System programming

## 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, 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
- Subroutine call and return
  - On line 10, “STL RETADDR” is called to save the return address that is already stored in register L.
  - Otherwise, after calling RD or WR, this COPY cannot return back to its caller.

System programming

## 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.

System programming

## 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

System programming

## THE ASSEMBLER'S JOB

- Convert mnemonic operation codes to their machine language codes
- Convert symbolic (e.g., jump labels, variable names) operands to their machine addresses
- Use proper addressing modes and formats to build efficient machine instructions
- Translate data constants into internal machine representations
- Output the object program and provide other information (e.g., for linker and loader)

System programming

## FIGURE 2.1 (PSEUDO CODE)

```
Program copy {
        save return address;
 cloop:   call subroutine RDREC to read one record;
          if length(record)=0 {
             call subroutine WRREC to write EOF;
        }  else {
                call subroutine WRREC to write one record;
             goto cloop;
          }
          load return address
          return to caller
  }
```

System programming

## AN EXAMPLE (FIGURE 2.1, CONT.)

EOR:
character x'00'

```
Subroutine RDREC {
          clear A, X register to 0;
 rloop:   read character from input device to A register
          if not EOR {
                store character into buffer[X];
                X++;
                if X < maximum length
                    goto rloop;
          }
          store X to length(record);
          return
}
```

System programming

## AN EXAMPLE (FIGURE 2.1, CONT.)

```
Subroutine WDREC {
          clear X register to 0;
 wloop:   get character from buffer[X]
          write character from X to output device
          X++;
          if X < length(record)
                goto wloop;
          return
  }
```

## System programming

| 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 | | . | | | |

## System programming

| | | | | | |
|---|---|---|---|---|---|
| 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 |

## System programming

| | | | | | |
|---|---|---|---|---|---|
| 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 | |

## System programming

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)* |

```
H^COPY  ^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.

## System programming

### EXAMPLE OF INSTRUCTION ASSEMBLE

STCH BUFFER,X 549039

| 8 | 1 | 15 |
|---|---|---|
| opcode | x | address |
| | | m |
| $(54)_{16}$ | $1 \ (001)_2$ | $(039)_{16}$ |

- Forward reference

## System programming

### DIFFICULTIES: FORWARD REFERENCE

- Forward reference: reference to a label that is defined later in the program.

| Loc | Label | Operator | Operand | |
|---|---|---|---|---|
| 1000 | FIRST | STL | RETADR | |
| 1003 | CLOOP | JSUB | RDREC | |
| … | … | … | … | … |
| 1012 | | J | CLOOP | |
| … | … | … | … | … |
| 1033 | RETADR | RESW | 1 | |

System programming

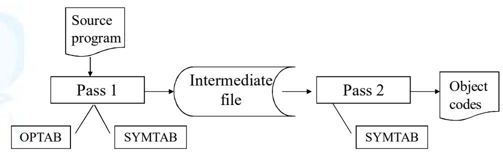
## TWO PASS ASSEMBLER

- Pass 1
  - Assign addresses to all statements in the program
  - Save the values assigned to all labels for use in Pass 2
  - Perform some processing of assembler directives
- Pass 2
  - Assemble instructions
  - 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

System programming

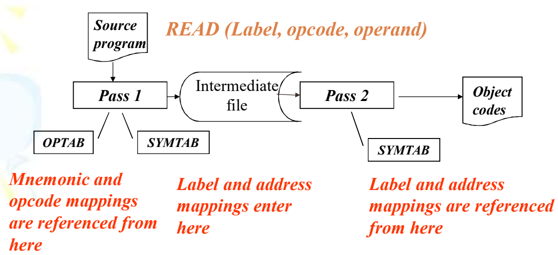
## TWO PASS ASSEMBLER

- Read from input line
  - LABEL, OPCODE, OPERAND




System programming

## TWO PASS ASSEMBLER




System programming

## DATA STRUCTURES

- Operation Code Table (OPTAB)
- Symbol Table (SYMTAB)
- Location Counter(LOCCTR)

System programming

## OPTAB (OPERATION CODE TABLE)

- Content
  - The mapping between mnemonic and machine code. Also include the instruction format, available addressing modes, and length information.
- Characteristic
  - Static table. The content will never change.
- Implementation
  - Array or hash table. Because the content will never change, we can optimize its search speed.
- 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.

System programming

## SYMTAB (SYMBOL TABLE)

- Content
  - Include the label name and value (address) for each label in the source program.
  - Include type and length information (e.g., int64)
  - 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.
  - Binary search tree
  - Linked lists indexed by the first alphabet of the mnemonic

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

System programming

## 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.

System programming

## HOMEWORK #3

```
SUM     START   4000
FIRST   LDX     ZERO
        LDA     ZERO
LOOP    ADD     TABLE,X
        TIX     COUNT
        JLT     LOOP
        STA     TOTAL
        RSUB
TABLE   RESW    2000
COUNT   RESW    1
ZERO    WORD    0
TOTAL   RESW    1
        END     FIRST
```

Assemblers 55

**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.

56 System Software

**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.

System programming

## 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
```

System programming

```
                     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
```

System programming

```
          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}
```

System programming

## 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
```

System programming

```
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
```

System programming

```
            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}
```

System programming

## 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
1036 LENGTH     RESW      1         LENGTH OF RECORD
1039 BUFFER     RESB      4096      4096-BYTE  BUFFER AREA
.
.               SUBROUTINE TO READ RECORD INTO BUFFER
.
2039 RDREC      LDX       ZERO      CLEAR LOOP COUNTER
...
.
.               SUBROUTINE TO WRITE RECORD FROM BUFFER
.
2061 WRREC      LDX       ZERO      CLEAR LOOP COUNTER
.
.
                END       FIRST
```

System programming

```
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 RETURN   ADDRESS
1027            RSUB                RETURN TO CALLER
102A EOF                  BYTE      C'EOF'
102D THREE      WORD      3
1030 ZERO                 WORD      0
1033 RETADR     RESW      1
1036 LENGTH     RESW      1         LENGTH OF RECORD
1039 BUFFER     RESB      4096      4096-BYTE  BUFFER AREA
.
.               SUBROUTINE TO READ RECORD INTO BUFFER
.
2039 RDREC      LDX       ZERO      CLEAR LOOP COU
...
.
.               SUBROUTINE TO WRITE RECORD FROM BUFFE
.
2061 WRREC      LDX       ZERO      CLEAR LOOP COU
.
.
                END       FIRST
```

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

System programming

ASS-2

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 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
.
.             SUBROUTINE TO READ RECORD INTO BUFFER
.
2039 RDREC    LDX     ZERO     CLEAR LOOP COU
…
.
.             SUBROUTINE TO WRITE RECORD FROM BUFFE
.
2061 WRREC    LDX     ZERO     CLEAR LOOP COU
.
.
              END     FIRST
```

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

```
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
```

System programming

```
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 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
.
.             SUBROUTINE TO READ RECORD INTO BUFFER
.
2039 RDREC    LDX     ZERO     CLEAR LOOP COU
…
.
.             SUBROUTINE TO WRITE RECORD FROM BUFFE
.
2061 WRREC    LDX     ZERO     CLEAR LOOP COU
.
.
              END     FIRST
```

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


System programming

```
  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
.
.             SUBROUTINE TO READ RECORD INTO BUFFER
.
2039 RDREC    LDX     ZERO     CLEAR LOOP COUNTER             041030
…
.
.             SUBROUTINE TO WRITE RECORD FROM BUFFER
.
2061 WRREC    LDX     ZERO     CLEAR LOOP COUNTER             041030
.
.
              END     FIRST
```

System programming

Chap 2

End of Sec 2-1

System programming

## ASSEMBLER DESIGN

- Algorithm: (simple assembler) only mnemonic machine instructions but not macro or subroutine calls
- Input: assembly program using only mnemonic code
- Output: object code of assembly language
  - Step 1: [Initialization] Initialize the location counter
  - Step 2: [Input] Read an assembly program with one mnemonic instruction at a time
  - Step 3: [Intermediate Program files] Save the program in condensed form (i.e., exclude comments parts, if any).

System programming

## ASSEMBLER DESIGN

- Step 4: [Syntax checking] verify the validity of the mnemonics of the instruction
- Step 5: [Type determination] Distinguish the type of the operands used( i.e. Symbolic names, absolute addresses, relative locations, literals, etc.) in the instruction
- *Step 6: [Insertion in the literal tables] Collect the literals in a table known as the literal table (LTab)*
- *Step 7: [Symbol table construction] Collect the symbolic names used to label the instruction in a table known as symbol table (STab) and assign the name with the current value of the location counter.*
- Step 8: [Increment LC] Increase the location counter value by the instruction length.
- Step 9: [End Checking] Detect the end of the assembly program
- Step 10: [Address assignment] Assign address for each literal in the literal table

System programming

## ASSEMBLER DESIGN

- Step 11: [Replacement opcode] Replace all mnemonic instruction names with there corresponding codes
- Step 12: [Replacement symbol table] Resolve all symbolic names with assigned values in the table
- Step 13: [Error if any] Provide error and diagnostic messages if there are invalid mnemonics or unresolved symbolic names
- Step 14: Provide a machine code program with additional information indicating those addresses which are either relative or absolute, and those values which are designed as literals
- Step 15: [Termination]
  - Steps 1-10 First pass
  - Steps 11-15 second pass

System programming

## ASSEMBLER FEATURES

- Machine Dependent Assembler Features
  - instruction formats
  - addressing modes
  - program relocation
- Machine Independent Assembler Features
  - literals
  - symbol-defining statements
  - Expressions handling in the program
  - program blocks
  - control sections and
  - program linking

System programming

## ASSEMBLER DESIGN CRITERIA

- How many times does a translation process look at the source code to resolve the forward references,
  - i.e., how many passes are required by a translation process?
    - e.g. two pass required two complete pass
- How can one ensure that the translation process will do everything
  - i.e. symbol table generation, code generation, and program listing on a minimum number of passes and look at the source code with minimum number of times?
- How does one handle forward references?
- How can the total processing time be minimized?

System programming

## MACHINE-DEPENDENT ASSEMBLER FEATURES

System programming

## INSTRUCTION FORMAT AND ADDRESSING MODE

- SIC/XE
  - PC-relative or Base-relative addressing: op m
  - Indirect addressing: op @m
  - Immediate addressing: op #c
  - Extended format: +op m
  - Index addressing: op m,x
  - register-to-register instructions
  - larger memory -> multi-programming (program allocation)

System programming

## TRANSLATION

- Register translation
  - register name (A, X, L, B, S, T, F, PC, SW) and their values (0,1, 2, 3, 4, 5, 6, 8, 9)
  - preloaded in SYMTAB
- Address translation
  - Most register-memory instructions use program counter relative or base relative addressing
  - Format 3: 12-bit address field
    - base-relative: 0~4095
    - pc-relative: -2048~2047
  - Format 4: 20-bit address field

System programming

```
  5  COPY     START   0          COPY FILE FROM INPUT TO OUTPUT
 10  FIRST    STL     RETADR     SAVE RETURN ADDRESS
 12           LDB     #LENGTH    ESTABLISH BASE REGISTER
 13           BASE    LENGTH
 15  CLOOP    +JSUB   RDREC      READ INPUT RECORD
 20           LDA     LENGTH     TEST FOR EOF (LENGTH = 0)
 25           COMP    #0
 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     #3         SET LENGTH = 3
 60           STA     LENGTH
 65           +JSUB   WRREC      WRITE EOF
 70           J       @RETADR    RETURN TO CALLER
 80  EOF      BYTE    C'EOF'
 95  RETADR   RESW    1
100  LENGTH   RESW    1          LENGTH OF RECORD
105  BUFFER   RESB    4096       4096-BYTE BUFFER AREA
110  .
```

System programming

```
115  .        SUBROUTINE TO READ RECORD INTO BUFFER
120  .
125  RDREC    CLEAR   X          CLEAR LOOP COUNTER
130           CLEAR   A          CLEAR A TO ZERO
132           CLEAR   S          CLEAR S TO ZERO
133           +LDT    #4096
135  RLOOP    TD      INPUT      TEST INPUT DEVICE
140           JEQ     RLOOP      LOOP UNTIL READY
145           RD      INPUT      READ CHARACTER INTO REGISTER A
150           COMPR   A,S        TEST FOR END OF RECORD (X'00')
155           JEQ     EXIT       EXIT LOOP IF EOR
160           STCH    BUFFER,X   STORE CHARACTER IN BUFFER
165           TIXR    T          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
195  .
```

System programming

```
     .
200  .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205  .
210  WRREC    CLEAR   X          CLEAR LOOP COUNTER
212           LDT     LENGTH
215  WLOOP    TD      OUTPUT     TEST OUTPUT DEVICE
220           JEQ     WLOOP      LOOP UNTIL READY
225           LDCH    BUFFER,X   GET CHARACTER FROM BUFFER
230           WD      OUTPUT     WRITE CHARACTER
235           TIXR    T          LOOP UNTIL ALL CHARACTERS
240           JLT     WLOOP        HAVE BEEN WRITTEN
245           RSUB               RETURN TO CALLER
250  OUTPUT   BYTE    X'05'      CODE FOR OUTPUT DEVICE
255           END     FIRST
```

| Line | Loc | Source statement | | | Object code |
|---|---|---|---|---|---|
| 5 | 0000 | COPY | START | 0 | |
| 10 | 0000 | FIRST | STL | RETADR | 17202D |
| 12 | 0003 | | LDB | #LENGTH | 69202D |
| 13 | | | BASE | LENGTH | |
| 15 | 0006 | CLOOP | +JSUB | RDREC | 4B101036 |
| 20 | 000A | | LDA | LENGTH | 032026 |
| 25 | 000D | | COMP | #0 | 290000 |
| 30 | 0010 | | JEQ | ENDFIL | 332007 |
| 35 | 0013 | | +JSUB | WRREC | 4B10105D |
| 40 | 0017 | | J | CLOOP | 3F2FEC |
| 45 | 001A | ENDFIL | LDA | EOF | 032010 |
| 50 | 001D | | STA | BUFFER | 0F2016 |
| 55 | 0020 | | LDA | #3 | 010003 |
| 60 | 0023 | | STA | LENGTH | 0F200D |
| 65 | 0026 | | +JSUB | WRREC | 4B10105D |
| 70 | 002A | | J | @RETADR | 3E2003 |
| 80 | 002D | EOF | BYTE | C'EOF' | 454F46 |
| 95 | 0030 | RETADR | RESW | 1 | |
| 100 | 0033 | LENGTH | RESW | 1 | |
| 105 | 0036 | BUFFER | RESB | 4096 | |

System programming

```
110         .
115         .        SUBROUTINE TO READ RECORD INTO BUFFER
120         .
125  1036   RDREC    CLEAR   X          B410
130  1038            CLEAR   A          B400
132  103A            CLEAR   S          B440
133  103C            +LDT    #4096      75101000
135  1040   RLOOP    TD      INPUT      E32019
140  1043            JEQ     RLOOP      332FFA
145  1046            RD      INPUT      DB2013
150  1049            COMPR   A,S        A004
155  104B            JEQ     EXIT       332008
160  104E            STCH    BUFFER,X   57C003
165  1051            TIXR    T          B850
170  1053            JLT     RLOOP      3B2FEA
175  1056   EXIT     STX     LENGTH     134000
180  1059            RSUB               4F0000
185  105C   INPUT    BYTE    X'F1'      F1
```

System programming

```
195         .
200         .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205         .
210  105D   WRREC    CLEAR   X          B410
212  105F            LDT     LENGTH     774000
215  1062   WLOOP    TD      OUTPUT     E32011
220  1065            JEQ     WLOOP      332FFA
225  1068            LDCH    BUFFER,X   53C003
230  106B            WD      OUTPUT     DF2008
235  106E            TIXR    T          B850
240  1070            JLT     WLOOP      3B2FEF
245  1073            RSUB               4F0000
250  1076   OUTPUT   BYTE    X'05'      05
255                  END     FIRST
```

System programming

## PC-RELATIVE ADDRESSING MODES

- PC-relative
  - 10 0000 FIRST STL RETADR 17202D

| op(6) | n | I | x | b | p | e | disp(12) |
|---|---|---|---|---|---|---|---|
| $(14)_{16}$ | 1 | 1 | 0 | 0 | 1 | 0 | $(02D)_{16}$ |

  - displacement= RETADR - PC = 30-3 = 2D
  - 40 0017 J CLOOP 3F2FEC

| op(6) | n | I | x | b | p | e | disp(12) |
|---|---|---|---|---|---|---|---|
| $(3C)_{16}$ | 1 | 1 | 0 | 0 | 1 | 0 | $(FEC)_{16}$ |

  - displacement= CLOOP-PC= 6 - 1A= -14= FEC

System programming

## BASE-RELATIVE ADDRESSING MODES

- Base-relative
  - base register is under the control of the programmer
  - 12 LDB #LENGTH
  - 13 BASE LENGTH
  - 160 104E STCH BUFFER, X 57C003

| op(6) | n | I | x | b | p | e | disp(12) |
|---|---|---|---|---|---|---|---|
| $(54)_{16}$ | 1 | 1 | 1 | 1 | 0 | 0 | $(003)_{16}$ |
| (54) | 1 | 1 | 1 | 0 | 1 | 0 | 0036-1051= $-101B_{16}$ |

  - displacement= BUFFER - B = 0036 - 0033 = 3
  - NOBASE is used to inform the assembler that the contents of the base register no longer be relied upon for addressing

System programming

## IMMEDIATE ADDRESS TRANSLATION

- Immediate addressing
  - 55 0020 LDA #3 010003

| op(6) | n | I | x | b | p | e | disp(12) |
|---|---|---|---|---|---|---|---|
| $(00)_{16}$ | 0 | 1 | 0 | 0 | 0 | 0 | $(003)_{16}$ |

  - 133 103C +LDT #4096 75101000

| op(6) | n | I | x | b | p | e | disp(20) |
|---|---|---|---|---|---|---|---|
| $(74)_{16}$ | 0 | 1 | 0 | 0 | 0 | 1 | $(01000)_{16}$ |

System programming

## IMMEDIATE ADDRESS TRANSLATION (CONT.)

- Immediate addressing
  - 12 0003 LDB #LENGTH 69202D

| op(6) | n | I | x | b | p | e | disp(12) | |
|---|---|---|---|---|---|---|---|---|
| $(68)_{16}$ | 0 | 1 | 0 | 0 | 1 | 0 | $(02D)_{16}$ | |
| $(68)_{16}$ | 0 | 1 | 0 | 0 | 0 | 0 | $(033)_{16}$ | 690033 |

  - the immediate operand is the symbol LENGTH
  - the address of this symbol LENGTH is loaded into register B
  - LENGTH=0033=PC+displacement=0006+02D
  - if immediate mode is specified, the target address becomes the operand

System programming

## INDIRECT ADDRESS TRANSLATION

- Indirect addressing
  - target addressing is computed as usual (PC-relative or BASE-relative)
  - only the n bit is set to 1
  - 70 002A J @RETADR 3E2003

| op(6) | n | I | x | b | p | e | disp(12) |
|---|---|---|---|---|---|---|---|
| $(3C)_{16}$ | 1 | 0 | 0 | 0 | 1 | 0 | $(003)_{16}$ |

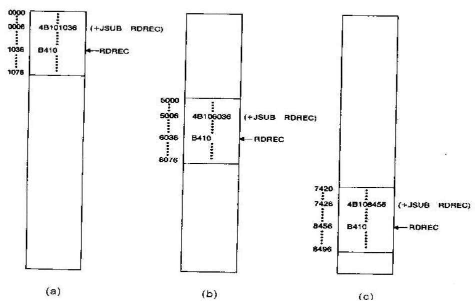
  - TA=RETADR=0030
  - TA=(PC)+disp=002D+0003

System programming

## PROGRAM RELOCATION

- Example Fig. 2.1
  - *Absolute program*, starting address 1000
    e.g. 55 101B LDA THREE 00102D
  - Relocate the program to 2000
    e.g. 55 101B LDA THREE 00202D
  - Each Absolute address should be modified
- Example Fig. 2.5:
  - Except for absolute address, the rest of the instructions need not be modified
    - not a memory address (immediate addressing)
    - PC-relative, Base-relative
  - The only parts of the program that require modification at load time are those that specify direct addresses

## EXAMPLE




## RELOCATABLE PROGRAM

- Modification record
  - Col 1 M
  - Col 2-7 Starting location of the address field to be modified, relative to the beginning of the program
  - Col 8-9 length of the address field to be modified, in half-bytes

## OBJECT CODE

```
H^COPY  ^000000^001077
T^000000^1D^17202D^69202D^4B101036^032026^290000^332007^4B10105D^3F2FEC^032010
T^00001D^13^0F2016^010003^0F200D^4B10105D^3E2003^454F46
T^001036^1D^B410^B400^B440^75101000^E32019^332FFA^DB2013^A004^332008^57C003^B850
T^001053^1D^3B2FEA^134000^4F0000^F1^B410^774000^E32011^332FFA^53C003^DF2008^B850
T^001070^07^3B2FEF^4F0000^05
M^000007^05
M^000014^05
M^000027^05
E^000000
```

End of Sec 2-2

Literals
Symbol Defining Statement
Expressions
Program Blocks
Control Sections and Program Linking

## MACHINE-INDEPENDENT ASSEMBLER FEATURES

## LITERALS

- Design idea
  - Let programmers to be able to write the value of a constant operand as a part of the instruction that uses it.
  - This avoids having to define the constant elsewhere in the program and make up a label for it.
- Example
  - e.g. 45 001A ENDFIL LDA =C'EOF' 032010
  - 93 LTORG
  - 002D * =C'EOF' 454F46
  - e.g. 215 1062 WLOOP TD =X'05' E32011

## LITERALS VS. IMMEDIATE OPERANDS

- Immediate Operands
  - The operand value is assembled as part of the machine instruction
  - e.g. 55 0020 LDA #3 010003
- Literals
  - The assembler generates the specified value as a constant at some other memory location
  - e.g. 45 001A ENDFILLDA =C'EOF' 032010
- Compare (Fig. 2.6)
  - e.g. 45 001A ENDFIL LDA EOF 032010
  - 80 002D EOF BYTE C'EOF' 454F46

System programming

## LITERAL - IMPLEMENTATION (1/3)

- Literal pools
  - Normally literals are placed into a pool at the end of the program
    - see Fig. 2.10 (END statement)
  - In some cases, it is desirable to place literals into a pool at some other location in the object program
    - assembler directive LTORG
    - reason: keep the literal operand close to the instruction

System programming

## LITERAL - IMPLEMENTATION (2/3)

- Duplicate literals
  - e.g. 215 1062 WLOOP TD =X'05'
  - e.g. 230 106B WD =X'05'
  - The assemblers should recognize duplicate literals and store only one copy of the specified data value
    - Comparison of the defining expression
      - Same literal name with different value, e.g. LOCCTR=*
    - Comparison of the generated data value
      - The benefits of using generate data value are usually not great enough to justify the additional complexity in the assembler

System programming

## LITERAL - IMPLEMENTATION (3/3)

- LITTAB
  - literal name, the operand value and length, the address assigned to the operand
- Pass 1
  - build LITTAB with literal name, operand value and length, leaving the address unassigned
  - when LTORG statement is encountered, assign an address to each literal not yet assigned an address
- Pass 2
  - search LITTAB for each literal operand encountered
  - generate data values using BYTE or WORD statements
  - generate modification record for literals that represent an address in the program

System programming

## SYMBOL-DEFINING STATEMENTS

- Labels on instructions or data areas
  - the value of such a label is the address assigned to the statement
- Defining symbols
  - symbol EQU value
  - value can be: ✶ constant, ✸ other symbol, ✹ expression
  - making the source program easier to understand
  - no forward reference

System programming

## SYMBOL-DEFINING STATEMENTS

- Example 1
  - MAXLEN EQU 4096
  - +LDT #MAXLEN

| +LDT | #4096 |
|---|---|

- Example 2 (Many general purpose registers)
  - BASE EQU R1
  - COUNT EQU R2
  - INDEX EQU R3
- Example 3
  - MAXLEN EQU BUFEND-BUFFER

System programming

## ORG (ORIGIN)

- Indirectly assign values to symbols
- Reset the location counter to the specified value
  - ORG value
- Value can be: ✶ constant, ✸ other symbol, ✹ expression
- No forward reference
- Example
  - SYMBOL: 6bytes
  - VALUE: 1word
  - FLAGS: 2bytes
  - LDA VALUE, X

| | SYMBOL | VALUE | FLAGS |
|---|---|---|---|
| STAB | | | |
| (100 entries) | | | |
| | | | |
| | : | : | : |

System programming

## ORG EXAMPLE

- Using EQU statements
  - STAB RESB 1100
  - SYMBOL EQU STAB
  - VALUE EQU STAB+6
  - FLAG EQU STAB+9
- Using ORG statements
  - STAB RESB 1100
  - ORG STAB
  - SYMBOL RESB 6
  - VALUE RESW 1
  - FLAGS RESB 2
  - ORG STAB+1100

System programming

## EXPRESSIONS

- Expressions can be classified as absolute expressions or relative expressions
  - MAXLEN EQU BUFEND-BUFFER
  - BUFEND and BUFFER both are relative terms, representing addresses within the program
  - However the expression BUFEND-BUFFER represents an absolute value
- When relative terms are paired with opposite signs, the dependency on the program starting address is canceled out; the result is an absolute value

System programming

## SYMTAB

- None of the relative terms may enter into a multiplication or division operation
- Errors:
  - BUFEND+BUFFER
  - 100-BUFFER
  - 3*BUFFER
- The type of an expression
  - keep track of the types of all symbols defined in the program

| Symbol | Type | Value |
|---|---|---|
| RETADR | R | 30 |
| BUFFER | R | 36 |
| BUFEND | R | 1036 |
| MAXLEN | A | 1000 |

System programming

## EXAMPLE 2.9

SYMTAB

| Name | Value |
|---|---|
| COPY | 0 |
| FIRST | 0 |
| CLOOP | 6 |
| ENDFIL | 1A |
| RETADR | 30 |
| LENGTH | 33 |
| BUFFER | 36 |
| BUFEND | 1036 |
| MAXLEN | 1000 |
| RDREC | 1036 |
| RLOOP | 1040 |
| EXIT | 1056 |
| INPUT | 105C |
| WREC | 105D |
| WLOOP | 1062 |

LITTAB

| | | | |
|---|---|---|---|
| C'EOF' | 454F46 | 3 | 002D |
| X'05' | 05 | 1 | 1076 |

System programming

## PROGRAM BLOCKS

- Program blocks
  - refer to segments of code that are rearranged within a single object program unit
  - USE [blockname]
  - Default block
  - Example: Figure 2.11
  - Each program block may actually contain several separate segments of the source program

Source statement

```
COPY      START   0               COPY FILE FROM INPUT TO OUTPUT
FIRST     STL     RETADR          SAVE RETURN ADDRESS
CLOOP     JSUB    RDREC           READ INPUT RECORD
          LDA     LENGTH          TEST FOR EOF (LENGTH = 0)
          COMP    #0
          JEQ     ENDFIL          EXIT IF EOF FOUND
          JSUB    WRREC           WRITE OUTPUT RECORD
          J       CLOOP           LOOP
ENDFIL    LDA     =C'EOF'         INSERT END OF FILE MARKER
          STA     BUFFER
          LDA     #3              SET LENGTH = 3
          STA     LENGTH
          JSUB    WRREC           WRITE EOF
          J       @RETADR         RETURN TO CALLER
          USE     CDATA
RETADR    RESW    1
LENGTH    RESW    1               LENGTH OF RECORD
          USE     CBLKS
BUFFER    RESB    4096            4096-BYTE BUFFER AREA
BUFEND    EQU     *               FIRST LOCATION AFTER BUFFER
MAXLEN    EQU     BUFEND-BUFFER   MAXIMUM RECORD LENGTH
.
.         SUBROUTINE TO READ RECORD INTO BUFFER
.
          USE
RDREC     CLEAR   X               CLEAR LOOP COUNTER
          CLEAR   A               CLEAR A TO ZERO
          CLEAR   S               CLEAR S TO ZERO
         +LDT     #MAXLEN
RLOOP     TD      INPUT           TEST INPUT DEVICE
          JEQ     RLOOP           LOOP UNTIL READY
          RD      INPUT           READ CHARACTER INTO REGISTER A
          COMPR   A,S             TEST FOR END OF RECORD (X'00')
          JEQ     EXIT            EXIT LOOP IF EOR
          STCH    BUFFER,X        STORE CHARACTER IN BUFFER
          TIXR    T               LOOP UNLESS MAX LENGTH
          JLT     RLOOP             HAS BEEN REACHED
EXIT      STX     LENGTH          SAVE RECORD LENGTH
          RSUB                    RETURN TO CALLER
          USE     CDATA
INPUT     BYTE    X'F1'           CODE FOR INPUT DEVICE
.
.         SUBROUTINE TO WRITE RECORD FROM BUFFER
.
          USE
WRREC     CLEAR   X               CLEAR LOOP COUNTER
          LDT     LENGTH
WLOOP     TD      =X'05'          TEST OUTPUT DEVICE
          JEQ     WLOOP           LOOP UNTIL READY
          LDCH    BUFFER,X        GET CHARACTER FROM BUFFER
          WD      =X'05'          WRITE CHARACTER
          TIXR    T               LOOP UNTIL ALL CHARACTERS
          JLT     WLOOP             HAVE BEEN WRITTEN
          RSUB                    RETURN TO CALLER
          USE     CDATA
          LTORG
          END     FIRST
```

Figure 2.11 Example of a program with multiple program blocks.

System programming

## PROGRAM BLOCKS - IMPLEMENTATION

- Pass 1
  - each program block has a separate location counter
  - each label is assigned an address that is relative to the start of the block that contains it
  - at the end of Pass 1, the latest value of the location counter for each block indicates the length of that block
  - the assembler can then assign to each block a starting address in the object program
- Pass 2
  - The address of each symbol can be computed by adding the assigned block starting address and the relative address of the symbol to that block

## Figure 2.12 (a)

```
Line  Loc/Block       Source statement                     Object code

  5   0000  0   COPY     START   0
 10   0000  0   FIRST    STL     RETADR                    172063
 15   0003  0   CLOOP    JSUB    RDREC                     4B2021
 20   0006  0            LDA     LENGTH                    032060
 25   0009  0            COMP    #0                        290000
 30   000C  0            JEQ     ENDFIL                    332006
 35   000F  0            JSUB    WRREC                     4B203B
 40   0012  0            J       CLOOP                     3F2FEE
 45   0015  0   ENDFIL   LDA     =C'EOF'                   032055
 50   0018  0            STA     BUFFER                    0F2056
 55   001B  0            LDA     #3                        010003
 60   001E  0            STA     LENGTH                    0F2048
 65   0021  0            JSUB    WRREC                     4B2029
 70   0024  0            J       @RETADR                   3E203F
 92   0000  1            USE     CDATA
 95   0000  1   RETADR   RESW    1
100   0003  1   LENGTH   RESW    1
103   0000  2            USE     CBLKS
105   0000  2   BUFFER   RESB    4096
106   1000  2   BUFEND   EQU     *
107   1000      MAXLEN   EQU     BUFEND-BUFFER
110             .
115             .        SUBROUTINE TO READ RECORD INTO BUFFER
```

There is no block number for MAXLEN. This is because MAXLEN is an absolute symbol.

```
120             .
123   0027  0            USE
125   0027  0   RDREC    CLEAR   X                         B410
130   0029  0            CLEAR   A                         B400
132   002B  0            CLEAR   S                         B440
133   002D  0           +LDT     #MAXLEN                   75101000
135   0031  0   RLOOP    TD      INPUT                     E32038
140   0034  0            JEQ     RLOOP                     332FFA
145   0037  0            RD      INPUT                     DB2032
150   003A  0            COMPR   A,S                       A004
155   003C  0            JEQ     EXIT                      332008
160   003F  0            STCH    BUFFER,X                  57A02F
165   0042  0            TIXR    T                         B850
170   0044  0            JLT     RLOOP                     3B2FEA
175   0047  0   EXIT     STX     LENGTH                    13201F
180   004A  0            RSUB                              4F0000
183   0006  1            USE     CDATA
185   0006  1   INPUT    BYTE    X'F1'                     F1
195             .
200             .        SUBROUTINE TO WRITE RECORD FROM BUFFER
205             .
208   004D  0            USE
210   004D  0   WRREC    CLEAR   X                         B410
212   004F  0            LDT     LENGTH                    772017
215   0052  0   WLOOP    TD      =X'05'                    E3201B
220   0055  0            JEQ     WLOOP                     332FFA
225   0058  0            LDCH    BUFFER,X                  53A016
230   005B  0            WD      =X'05'                    DF2012
235   005E  0            TIXR    T                         B850
240   0060  0            JLT     WLOOP                     3B2FEF
245   0063  0            RSUB                              4F0000
252   0007  1            USE     CDATA
253                      LTORG
      0007  1   *        =C'EOF'                           454F46
      000A  1   *        =X'05'                            05
255                      END     FIRST
```

0063+3

System programming

## FIGURE 2.12

- Each source line is given a relative address assigned and a block number

| Block name | Block number | Address | Length |
|---|---|---|---|
| (default) | 0 | 0000 | 0066 |
| CDATA | 1 | 0066 | 000B |
| CBLKS | 2 | 0071 | 1000 |

- For absolute symbol, there is no block number
  - line 107
- Example
  - 20 0006 0 LDA LENGTH 032060
  - LENGTH=(Block 1)+0003= 0066+0003= 0069
  - LOCCTR=(Block 0)+0009= 0009

System programming

## PROGRAM READABILITY

- Program readability
  - No extended format instructions on lines 15, 35, 65
  - No needs for base relative addressing (line 13, 14)
  - LTORG is used to make sure the literals are placed ahead of any large data areas (line 253)
- Object code
  - It is not necessary to physically rearrange the generated code in the object program
  - see Fig. 2.13, Fig. 2.14

System programming

Figure 2.13

```
H^COPY  ^000000^001071
T^000000^1E^172063^4B2021^032060^290000^332006^4B203B^3F2FEE^032055^0F2056^010003
T^00001E^09^0F2048^4B2029^3E203F
T^000027^1D^B410^B400^B440^75101000^E32038^332FFA^DB2032^A004^332008^57A02F^B850
T^000044^09^3B2FEA^13201F^4F0000
T^00006C^01^F1
T^00004D^19^B410^772017^E3201B^332FFA^53A016^DF2012^B850^3B2FEF^4F0000
T^00006D^04^454F46^05
E^000000
```

## Figure 2.14

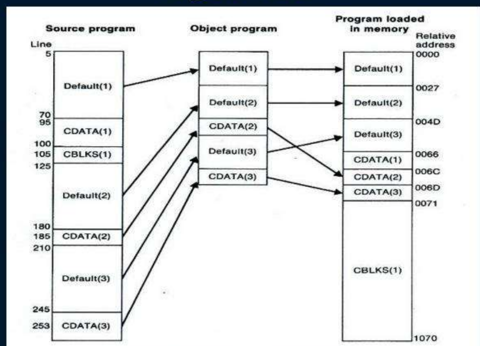



**Figure 2.14** Program blocks from Fig. 2.11 traced through the assembly and loading processes.

---

System programming

## CONTROL SECTIONS AND PROGRAM LINKING

- Control Sections
  - are most often used for subroutines or other logical subdivisions of a program
  - the programmer can assemble, load, and manipulate each of these control sections separately
  - instruction in one control section may need to refer to instructions or data located in another section
  - because of this, there should be some means for linking control sections together
  - Fig. 2.15, 2.16

---

System programming

## EXTERNAL DEFINITION AND REFERENCES

- External definition
  - EXTDEF name [, name]
  - EXTDEF names symbols that are defined in this control section and may be used by other sections
- External reference
  - EXTREF name [,name]
  - EXTREF names symbols that are used in this control section and are defined elsewhere
- Example
  - 15 0003 CLOOP +JSUB RDREC 4B100000
  - 160 0017 +STCH BUFFER,X 57900000
  - 190 0028 MAXLEN WORD BUFEND-BUFFER 000000

---

System programming

## IMPLEMENTATION

- The assembler must include information in the object program that will cause the loader to insert proper values where they are require
- Define record
  - Col. 1 D
  - Col. 2-7 Name of external symbol defined in this control section
  - Col. 8-13 Relative address within this control section (hexadeccimal)
  - Col.14-73 Repeat information in Col. 2-13 for other external symbols
- Refer record
  - Col. 1 R
  - Col. 2-7 Name of external symbol referred to in this control section
  - Col. 8-73 Name of other external reference symbols

---

System programming

## MODIFICATION RECORD

- Modification record
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified (hexiadecimal)
  - Col. 8-9 Length of the field to be modified, in half-bytes (hexadeccimal)
  - Col.11-16 External symbol whose value is to be added to or subtracted from the indicated field
  - Note: control section name is automatically an external symbol, i.e. it is available for use in Modification records.
- Example
  - Figure 2.17
  - M00000405+RDREC
  - M00000705+COPY

---

System programming

## EXTERNAL REFERENCES IN EXPRESSION

- Earlier definitions
  - required all of the relative terms be paired in an expression (an absolute expression), or that all except one be paired (a relative expression)
- New restriction
  - Both terms in each pair must be relative within the same control section
  - Ex: BUFEND-BUFFER
  - Ex: RDREC-COPY
- In general, the assembler cannot determine whether or not the expression is legal at assembly time. This work will be handled by a linking loader.

System programming

One-pass assemblers
Multi-pass assemblers
Two-pass assembler with overlay structure

# ASSEMBLER DESIGN OPTIONS

System programming

## TWO-PASS ASSEMBLER WITH OVERLAY STRUCTURE

- For small memory
  - pass 1 and pass 2 are never required at the same time
  - three segments
    - root: driver program and shared tables and subroutines
    - pass 1
    - pass 2
  - tree structure
  - overlay program

System programming

## ONE-PASS ASSEMBLERS

- Main problem
  - forward references
    - data items
    - labels on instructions
- Solution
  - data items: require all such areas be defined before they are referenced
  - labels on instructions: no good solution
- It is possible, although inconvenient, to do so for data items.
- Forward jump to instruction items cannot be easily eliminated.
  - Insert (label, *address_to_be_modified*) to SYMTAB
  - Usually, *address_to_be_modified* is stored in a linked-list

System programming

## FORWARD REFERENCE IN ONE-PASS ASSEMBLER

- For any symbol that has not yet been defined
  1. omit the address translation
  2. insert the symbol into SYMTAB, and mark this symbol undefined
  3. the address that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry
  4. when the definition for a symbol is encountered, the proper address for the symbol is then inserted into any instructions previous generated according to the forward reference list

| Name | Type | location | Size | Section id | Is-global |
|---|---|---|---|---|---|
| | | | | | |

| Name | Offset(hex) to be corrected |
|---|---|
| | |

System programming

## ONE-PASS ASSEMBLERS

- Two types of one-pass assembler
  - load-and-go
    - produces object code directly in memory for immediate execution
  - the other
    - produces usual kind of object code for later execution

System programming

## LOAD-AND-GO ASSEMBLER

- Characteristics
  - Useful for program development and testing
  - Avoids the overhead of writing the object program out and reading it back
  - Both one-pass and two-pass assemblers can be designed as load-and-go.
  - However one-pass also avoids the over head of an additional pass over the source program
  - For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program

System programming

## LOAD-AND-GO ASSEMBLER (CONT.)

- At the end of the program
  - any SYMTAB entries that are still marked with * indicate undefined symbols
  - search SYMTAB for the symbol named in the END statement and jump to this location to begin execution
- The actual starting address must be specified at assembly time
- Example
  - Figure 2.18, 2.19

System programming

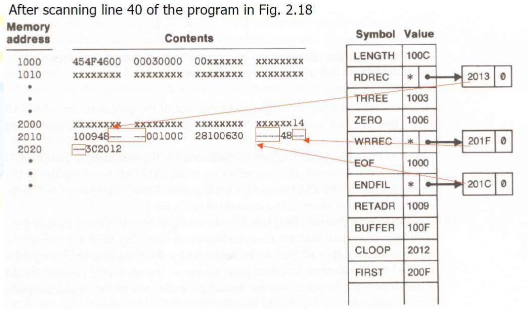
## EXAMPLE PROGRAM FIGURE 2.18

| Line | Loc | Source statement | | | Object code |
|---|---|---|---|---|---|
| 0 | 1000 | COPY | START | 1000 | |
| 1 | 1000 | EOF | BYTE | C'EOF' | 454F46 |
| 2 | 1003 | THREE | WORD | 3 | 000003 |
| 3 | 1006 | ZERO | WORD | 0 | 000000 |
| 4 | 1009 | RETADR | RESW | 1 | |
| 5 | 100C | LENGTH | RESW | 1 | |
| 6 | 100F | BUFFER | RESB | 4096 | |
| 9 | | . | | | |
| 10 | 200F | FIRST | STL | RETADR | 141009 |
| 15 | 2012 | CLOOP | JSUB | RDREC | 48203D |
| 20 | 2015 | | LDA | LENGTH | 00100C |
| 25 | 2018 | | COMP | ZERO | 281006 |
| 30 | 201B | | JEQ | ENDFIL | 302024 |
| 35 | 201E | | JSUB | WRREC | 482062 |
| 40 | 2021 | | J | CLOOP | 302012 |
| 45 | 2024 | ENDFIL | LDA | EOF | 001000 |
| 50 | 2027 | | STA | BUFFER | 0C100F |
| 55 | 202A | | LDA | THREE | 001003 |
| 60 | 202D | | STA | LENGTH | 0C100C |
| 65 | 2030 | | JSUB | WRREC | 482062 |
| 70 | 2033 | | LDL | RETADR | 081009 |
| 75 | 2036 | | RSUB | | 4C0000 |
| 110 | | . | | | |

System programming

## OBJECT CODE IN MEMORY AND SYMTAB FOR LOAD-AND-GO

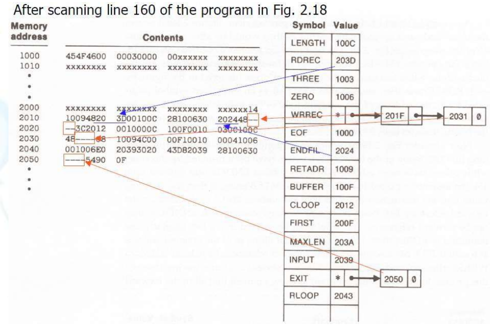
After scanning line 40 of the program in Fig. 2.18



System programming

## OBJECT CODE IN MEMORY AND SYMTAB FOR LOAD-AND-GO

After scanning line 160 of the program in Fig. 2.18



System programming

## PRODUCING OBJECT CODE

- When external working-storage devices are not available or too slow (for the intermediate file between the two passes
- Solution:
  - When definition of a symbol is encountered, the assembler must generate another Tex record with the correct operand address
  - The loader is used to complete forward references that could not be handled by the assembler
  - The object program records must be kept in their original order when they are presented to the loader
- Example: Figure 2.20

System programming

## ONE-PASS WITH OUTPUT FILE

```
H^COPY  ^001000^00107A
T^001000^09^454F46^000003^000000
T^00200F^15^141009^480000^00100C^281006^300000^480000^3C2012
T^00201C^02^2024
T^002024^19^001000^0C100F^001003^0C100C^480000^081009^4C0000^F1^001000
T^002013^02^203D
T^00203D^1E^041006^001006^E02039^302043^D82039^281006^300000^54900F^2C203A^382043
T^002050^02^205B
T^00205B^07^10100C^4C0000^05
T^00201F^02^2062
T^002031^02^2062
T^002062^18^041006^E02061^302065^50900F^DC2061^2C100C^382065^4C0000
E^00200F
```

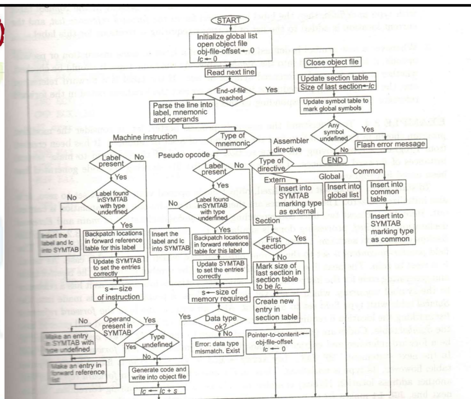


FIGURE 3.15 Single-pass assembler.

System programming

## MULTI-PASS ASSEMBLERS

- For a two pass assembler, forward references in symbol definition are not allowed:

```
ALPHA   EQU    BETA
BETA    EQU    DELTA
DELTA   RESW   1
```

  - Symbol definition must be completed in pass 1.
- Prohibiting forward references in symbol definition is not a serious inconvenience.
  - Forward references tend to create difficulty for a person reading the program.

System programming

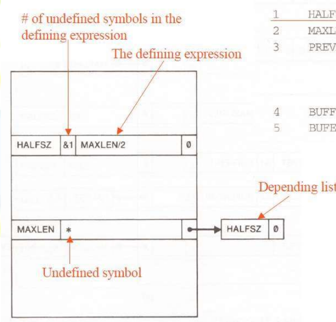
## MULTI-PASS IMPLEMENTATION

- For a forward reference in symbol definition, we store in the SYMTAB:
  - The symbol name
  - The defining expression
  - The number of undefined symbols in the defining expression
- The undefined symbol (marked with a flag *) associated with a list of symbols depend on this undefined symbol.
- When a symbol is defined, we can recursively evaluate the symbol expressions depending on the newly defined symbol.

System programming

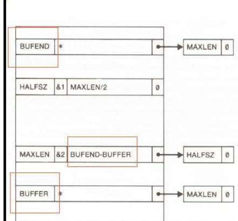
## MULTI-PASS EXAMPLE, FIGURE 2.21

```
1   HALFSZ   EQU    MAXLEN/2
2   MAXLEN   EQU    BUFEND-BUFFER
3   PREVBT   EQU    BUFFER-1
                    .
                    .
                    .
4   BUFFER   RESB   4096
5   BUFEND   EQU    *
```

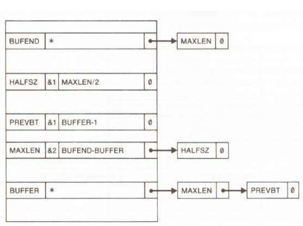



System programming

## MULTI-PASS EXAMPLE: FIGURE 2.21



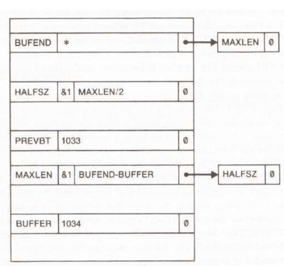
2 MAXLEN EQU BUFEND-BUFFER



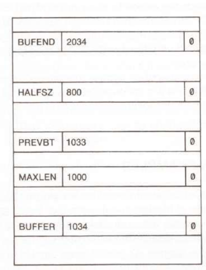
3 PREVBT EQU BUFFER-1

System programming

## MULTI-PASS EXAMPLE: FIGURE 2.21



4 BUFFER RESB 4096



5 BUFEND EQU *