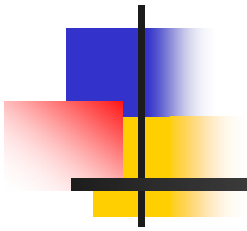
Computer Organization & Assembly Languages



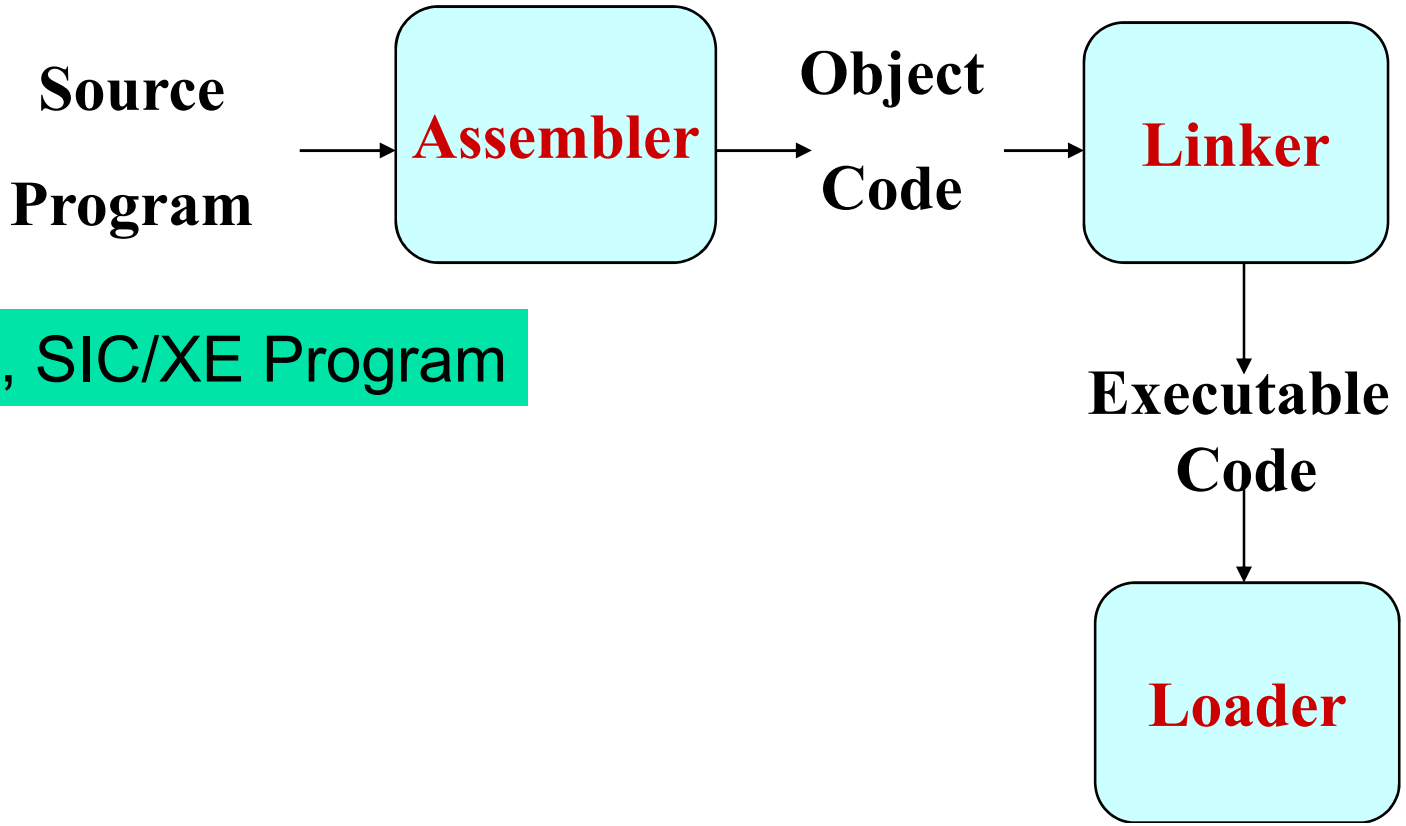
Assembler

Pu-Jen Cheng

Adapted from the slides prepared by Beck for the book,
System Software: An Intro. to Systems Programming , 3rd Ed.

Overview

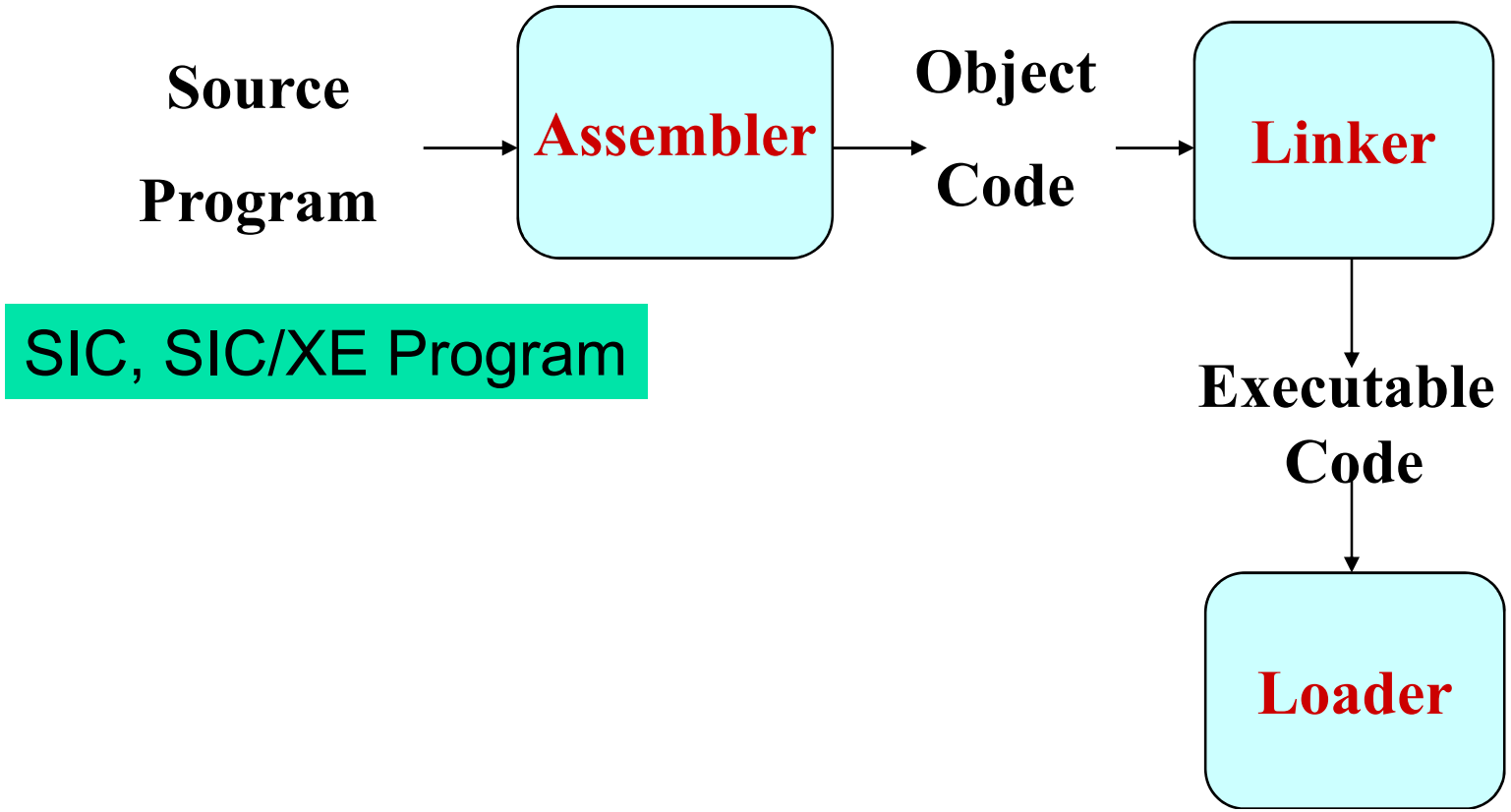
- SIC Machine Architecture
- SIC/XE Machine Architecture
- Design and Implementation of Assembler



SIC, SIC/XE Program

Overview

- **SIC Machine Architecture**
- SIC/XE Machine Architecture
- Design and Implementation of Assembler



SIC, SIC/XE Program

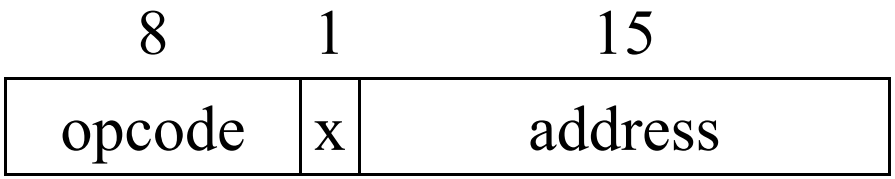
SIC Machine Architecture

- The Simplified Instructional Computer (SIC)
- Memory
  - Memory consists of 8-bit bytes
  - Any 3 consecutive bytes form a word (24 bits)
  - Total of 32768 (2^{15}) bytes in the computer memory
- Registers
  - Five 24-bits registers

Mnemonic	Number	Special use
A	0	Accumulator
X	1	Index register
L	2	Linkage register
PC	8	Program counter
SW	9	Status word

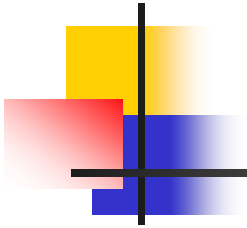
SIC Machine Architecture

- Data Formats
  - Integers are stored as 24-bit binary number
  - 2's complement representation for negative values
  - Characters are stored using 8-bit ASCII codes
  - No floating-point hardware on the standard version of SIC
- Instruction Formats
  - Standard version of SIC



Mode	Indication	Target address calculation
Direct	x=0	TA=address
Indexed	x=1	TA=address+(X)

(X): the contents of register X



SIC Machine Architecture

- Instruction Set
  - Load and store registers
    - LDA, LDX, STA, STX, etc.
  - Integer arithmetic operations
    - ADD, SUB, MUL, DIV
    - All arithmetic operations involve register A and a word in memory, with the result being left in A
  - COMP
  - Conditional jump instructions
    - JLT, JEQ, JGT
  - Subroutine linkage
    - JSUB, RSUB
  - I/O (transferring 1 byte at a time to/from the rightmost 8 bits of register A)
    - Test Device instruction (TD)
    - Read Data (RD)
    - Write Data (WD)

SIC Programming Example

```
        LDA     FIVE
        STA     ALPHA
        LDCH    CHARZ
        STCH    C1
        .
        .
        .
ALPHA   RESW    1           one-word variable
FIVE    WORD    5           one-word constant
CHARZ   BYTE    C'Z'        one-byte constant
C1      RESB    1           one-byte variable
```

SIC Programming Example

```
          LDX     ZERO          initialize index register to 0
MOVECH    LDCH    STR1,X        load char from STR1 to reg A
          STCH    STR2,X
          TIX     ELEVEN        add 1 to index, compare to 11
          JLT     MOVECH        loop if "less than"
          .
          .
          .
STR1      BYTE    C'TEST STRING'
STR2      RESB    11
ZERO      WORD    0
ELEVEN    WORD    11
```

SIC Programming Example

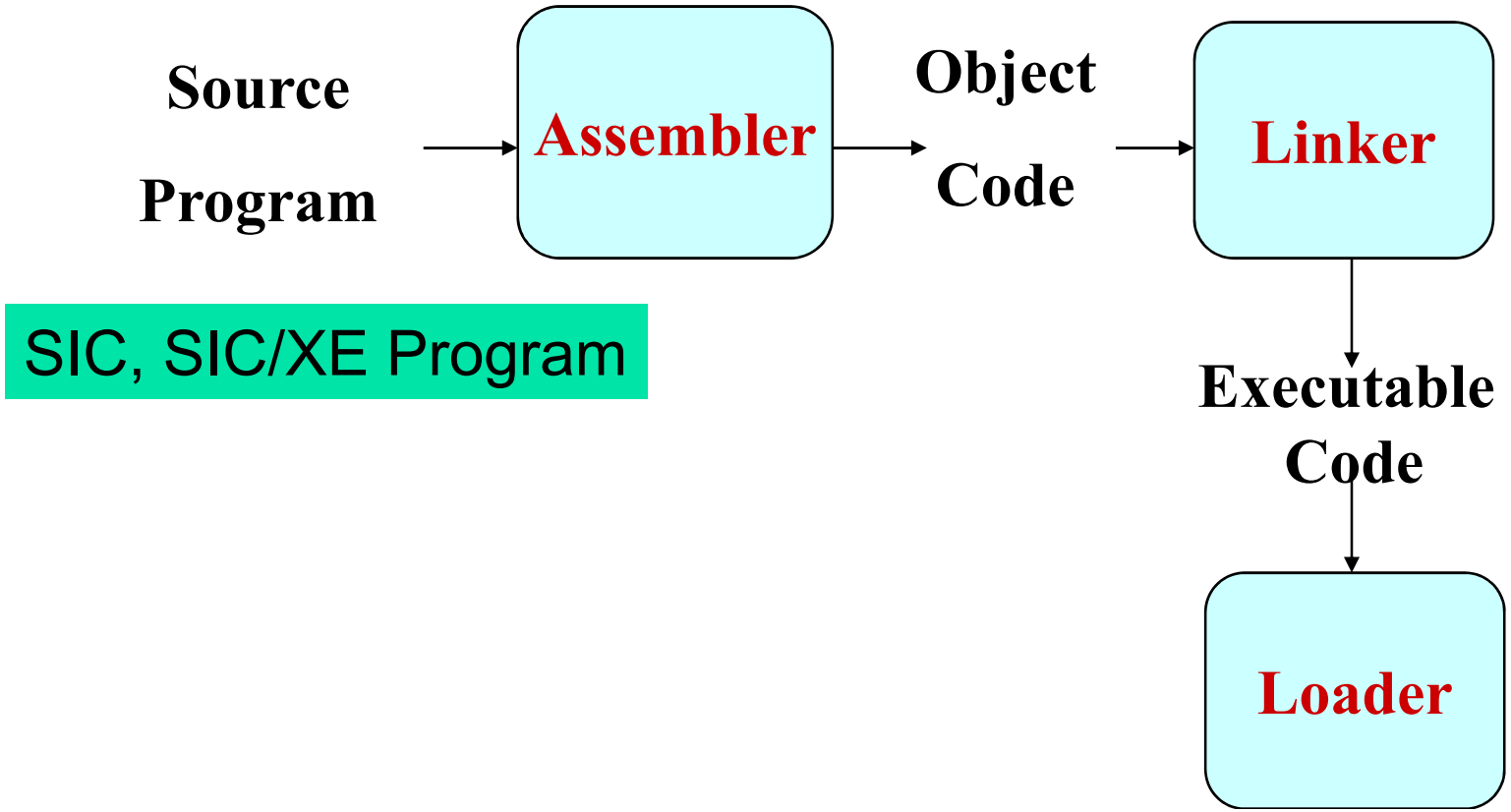
```
            LDA     ZERO        initialize index value to 0
            STA     INDEX
ADDLP       LDX     INDEX       load index value to reg X
            LDA     ALPHA,X     load word from ALPHA into reg A
            ADD     BETA,X
            STA     GAMMA,X     store the result in a word in GAMMA
            LDA     INDEX
            ADD     THREE       add 3 to index value
            STA     INDEX
            COMP    K300        compare new index value to 300
            JLT     ADDLP       loop if less than 300
            ...
            ...
INDEX       RESW    1
ALPHA       RESW    100         array variables—100 words each
BETA        RESW    100
GAMMA       RESW    100
ZERO        WORD    0           one-word constants
THREE       WORD    3
K300        WORD    300
```

SIC Programming Example

```
INLOOP  TD      INDEV   test input device
        JEQ     INLOOP  loop until device is ready (<)
        RD      INDEV   read one byte into register A
        STCH    DATA
        .
        .
OUTLP   TD      OUTDEV  test output device
        JEQ     OUTLP   loop until device is ready (<)
        LDCH    DATA
        WD      OUTDEV  write one byte to output device
        .
        .
INDEV   BYTE    X'F1'   input device number
OUTDEV  BYTE    X'05'   output device number
DATA    RESB    1
```

Overview

- SIC Machine Architecture
- **SIC/XE Machine Architecture**
- Design and Implementation of Assembler



SIC, SIC/XE Program

SIC/XE Machine Architecture

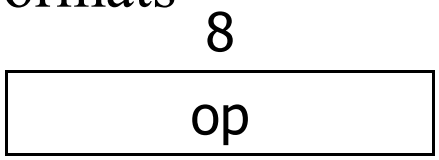
- An XE version (upward compatible)
- Memory
  - Maximum 1 megabyte (2^{20} bytes)
- Registers
  - Additional registers are provided by SIC/XE
- Support 48-bit floating-point data type

Mnemonic	Number	Special use
B	3	Base register
S	4	General working register
T	5	General working register
F	6	Floating-point accumulator (48 bits)

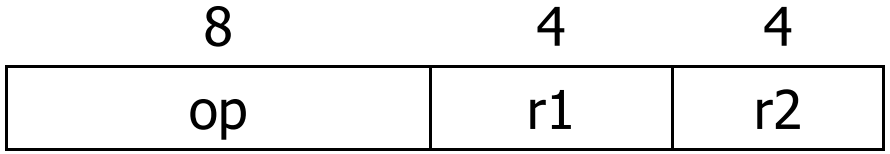
SIC/XE Machine Architecture

- Instruction Formats

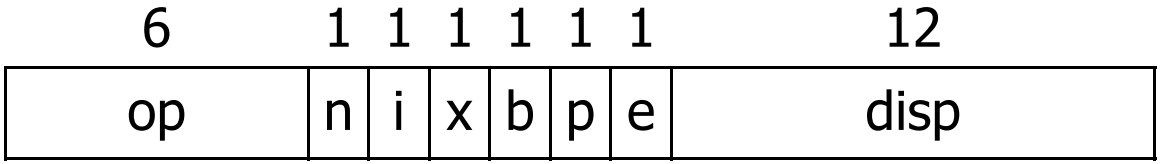
Format 1 (1 byte)



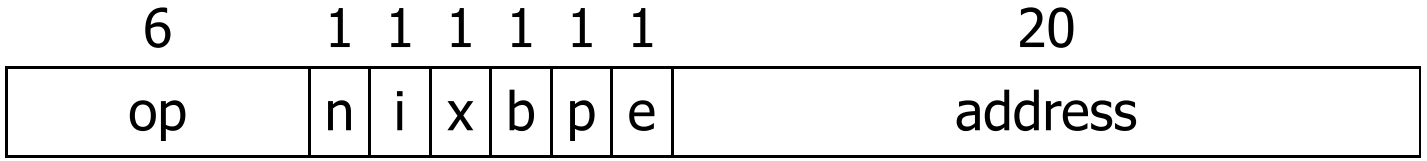
Format 2 (2 bytes)



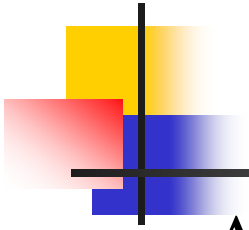
Format 3 (3 bytes)



Format 4 (4 bytes)



Formats 1 and 2 are instructions that do not reference memory at all

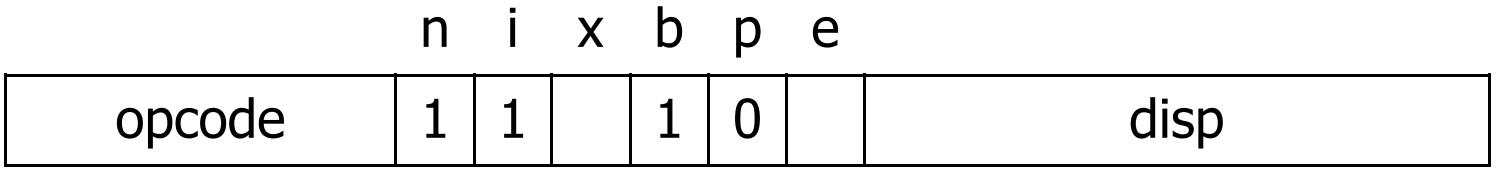


SIC/XE Machine Architecture

- Addressing modes
  - Base relative (n=1, i=1, b=1, p=0)
  - Program-counter relative (n=1, i=1, b=0, p=1)
  - Direct (n=1, i=1, b=0, p=0)
  - Immediate (n=0, i=1, x=0)
  - Indirect (n=1, i=0, x=0)
  - Indexing (both n & i = 0 or 1, x=1)
  - Extended (e=1 for format 4, e=0 for format 3)

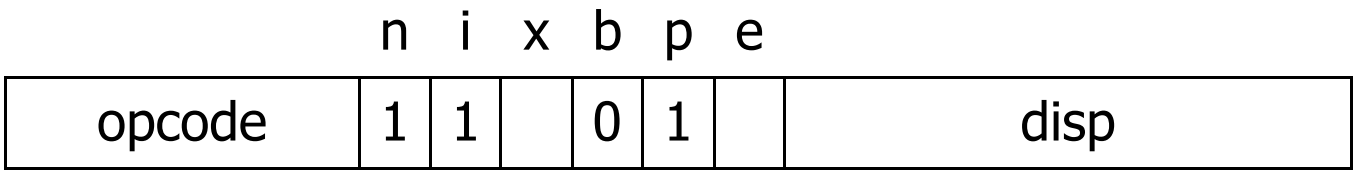
SIC/XE Machine Architecture

- Base Relative Addressing Mode



n=1, i=1, b=1, p=0, TA=(B)+disp (0≤disp ≤4095)

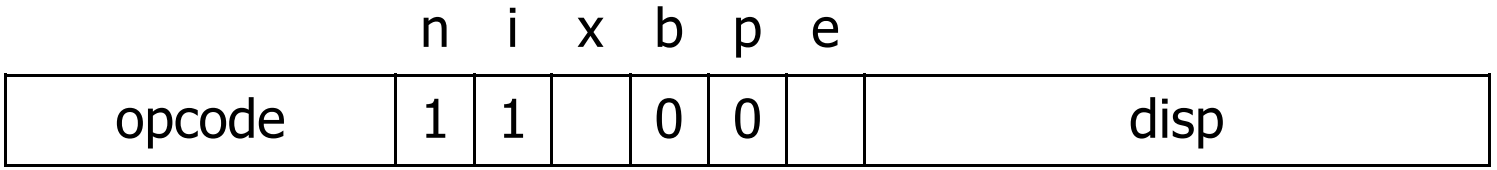
- Program-Counter Relative Addressing Mode



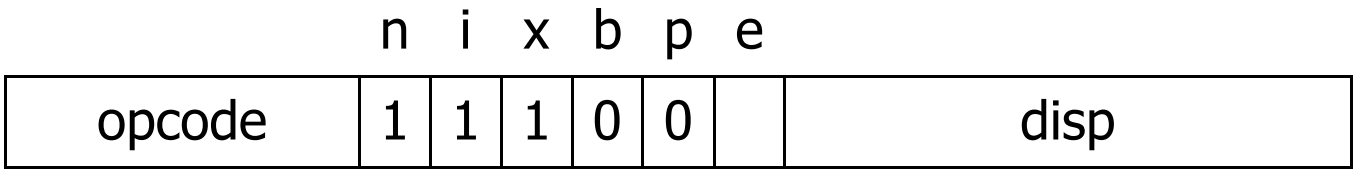
n=1, i=1, b=0, p=1, TA=(PC)+disp (-2048≤disp ≤2047)

SIC/XE Machine Architecture

- Direct Addressing Mode



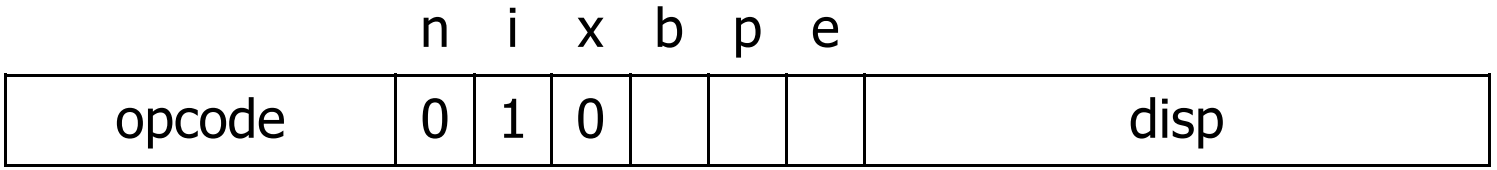
n=1, i=1, b=0, p=0, TA=disp (0≤disp ≤4095)



n=1, i=1, b=0, p=0, TA=(X)+disp
(with index addressing mode)

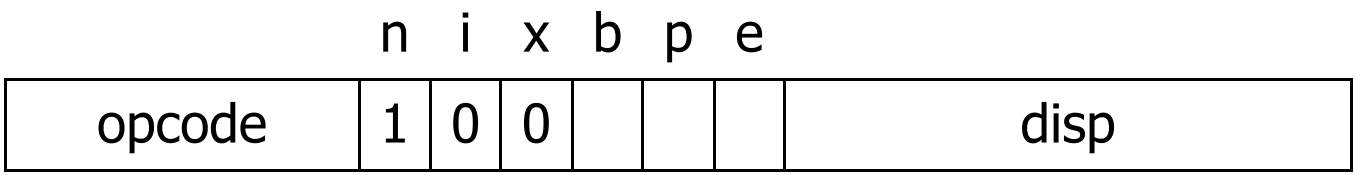
SIC/XE Machine Architecture

- Immediate Addressing Mode



n=0, i=1, x=0, operand=disp

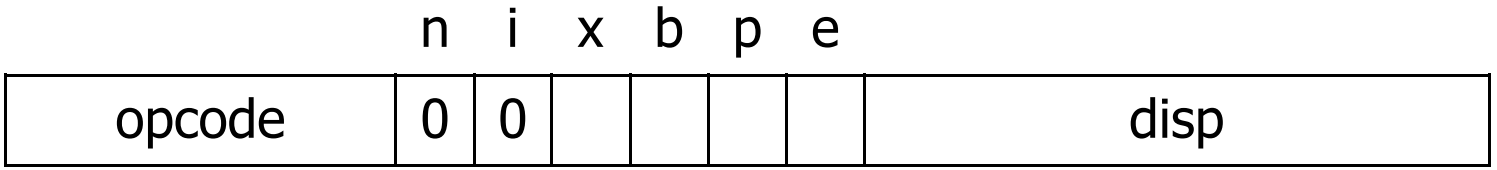
- Indirect Addressing Mode



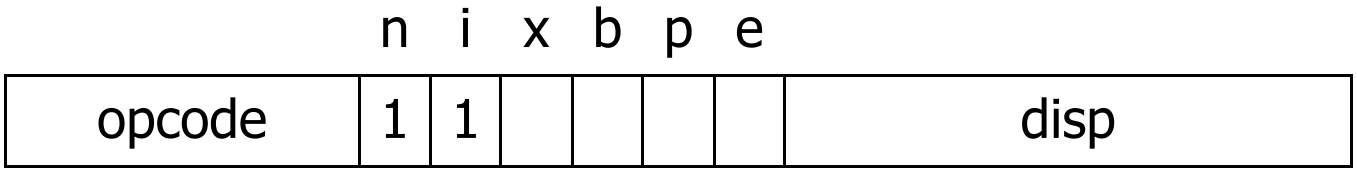
n=1, i=0, x=0, TA=(disp)

SIC/XE Machine Architecture

- Simple Addressing Mode

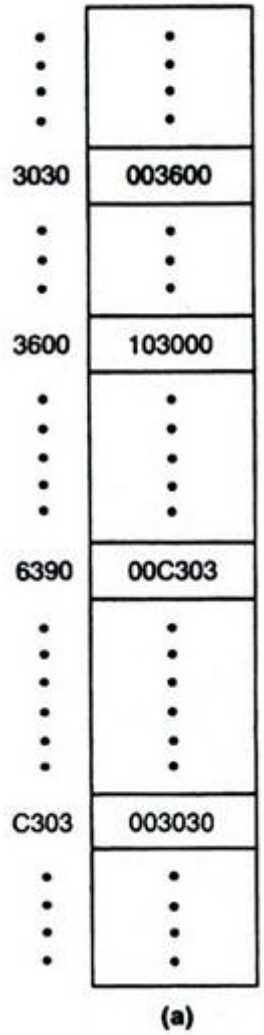


i=0, n=0, TA=bpe+disp (SIC standard)



i=1, n=1, TA=disp (SIC/XE standard)

SIC/XE Machine Architecture



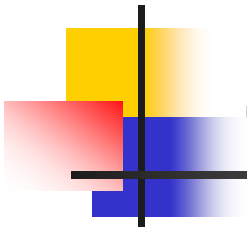
(B) = 006000
(PC) = 003000
(X) = 000090

- Instruction Format

Machine instruction									Value loaded into register A
Hex	Binary							Target address	
	op	n	i	x	b	p	e disp/address		
032600	000000	1	1	0	0	1	0 0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0 0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0 0000 0011 0000	3030	103000
010030	000000	0	1	0	0	0	0 0000 0011 0000	30	000030
003600	000000	0	0	0	0	1	1 0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1 0000 1100 0011 0000 0011	C303	003030

(b)

Figure 1.1 Examples of SIC/XE instructions and addressing modes.



SIC/XE Machine Architecture

- Instruction Set
  - Instructions to load and store the new registers
    - LDB, STB, etc.
  - Floating-point arithmetic operations
    - ADDF, SUBF, MULF, DIVF
  - Register move instruction
    - RMO
  - Register-to-register arithmetic operations
    - ADDR, SUBR, MULR, DIVR
  - Supervisor call instruction
    - SVC
  - Input and Output
    - There are I/O channels that can be used to perform input and output while the CPU is executing other instructions

SIC/XE Programming Example

SIC version

```
        LDA     FIVE
        STA     ALPHA
        LDCH    CHARZ
        STCH    C1
        .
        .
        .
ALPHA   RESW    1
FIVE    WORD    5
CHARZ   BYTE    C'Z'
C1      RESB    1
```

SIC/XE version

```
        LDA     #5
        STA     ALPHA
        LDCH    #90
        STCH    C1
        .
        .
        .
ALPHA   RESW    1
C1      RESB    1
```

SIC/XE Programming Example

```
        LDS     INCR
        LDA     ALPHA
        ADDR    S,A             A = A + S
        SUB     #1              A = A - 1
        STA     BETA
        LDA     GAMMA
        ADDR    S,A
        SUB     #1
        STA     DELTA
        ...
        ...
ALPHA   RESW    1           one-word variables
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

SIC/XE Programming Example

```
         LDT    #11        initialize register T to 11
         LDX    #0         initialize index register to 0
MOVECH   LDCH   STR1,X     load char from STR1 to reg A
         STCH   STR2,X     store char into STR2
         TIXR   T          add 1 to index, compare to 11
         JLT    MOVECH     loop if "less than" 11
         .
         .
         .
STR1     BYTE   C'TEST STRING'
STR2     RESB   11
```

SIC/XE Programming Example

```
        LDS     #3
        LDT     #300
        LDX     #0
ADDLP   LDA     ALPHA,X     load from ALPHA to reg A
        ADD     BETA,X
        STA     GAMMA,X     store in a word in GAMMA
        ADDR    S,X         add 3 to index value
        COMPR   X,T         compare to 300
        JLT     ADDLP       loop if less than 300
        ...
        ...
ALPHA   RESW    100         array variables—100 words each
BETA    RESW    100
GAMMA   RESW    100
```

SIC/XE Programming Example

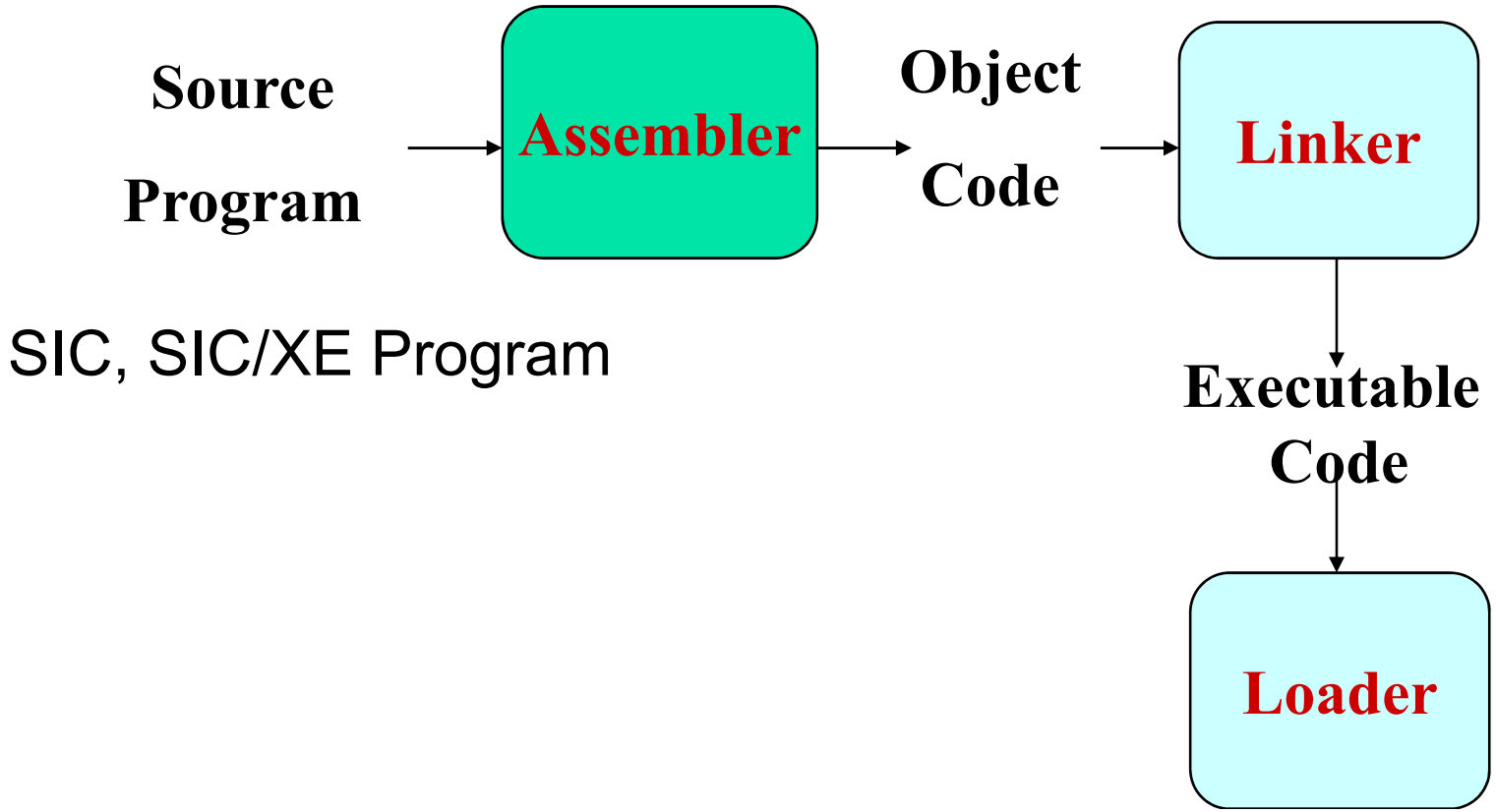
```
            JSUB    READ        CALL READ SUBROUTINE
            .
            .
            .
.                               SUBROUTINE TO READ 100-BYTE RECORD
READ        LDX     #0          INITIALIZE INDEX REGISTER TO 0
            LDT     #100        INITIALIZE REGISTER T TO 100
RLOOP       TD      INDEV       TEST INPUT DEVICE
            JEQ     RLOOP       LOOP IF DEVICE IS BUSY
            RD      INDEV       READ ONE BYTE INTO REGISTER A
            STCH    RECORD,X    STORE DATA BYTE INTO RECORD
            TIXR    T           ADD 1 TO INDEX AND COMPARE TO 100
            JLT     RLOOP       LOOP IF INDEX IS LESS THAN 100
            RSUB                EXIT FROM SUBROUTINE
            .
            .
            .
INDEV       BYTE    X'F1'       INPUT DEVICE NUMBER
RECORD      RESB    100         100-BYTE BUFFER FOR INPUT RECORD
```

(b)

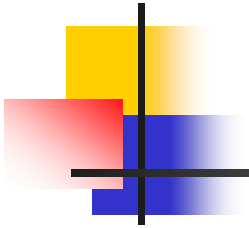
Figure 1.7 Sample subroutine call and record input operations for (a) SIC and (b) SIC/XE.

Overview

- SIC Machine Architecture
- SIC/XE Machine Architecture
- **Design and Implementation of Assembler**

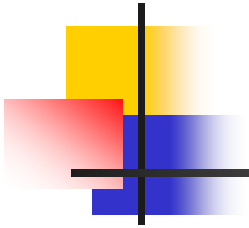


SIC, SIC/XE Program



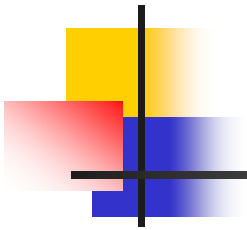
Design & Implementation of Assembler

- Functions of a Basic Assembler
- Object File
- Address Translation
- Program Relocation
- Program Block
- Control Section & Program Linking
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler



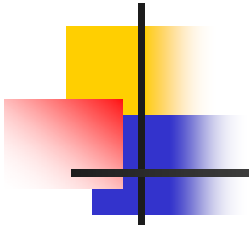
Design & Implementation of Assembler

- **Functions of a Basic Assembler**
- Object File
- Address Translation
- Program Relocation
- Program Block
- Control Section & Program Linking
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler



Functions of a Basic Assembler

- Mnemonic code (or instruction name) → opcode.
- Symbolic operands (e.g., variable names) → addresses.
- Choose the proper instruction format & addressing mode.
- Constants → Numbers.
- Output to object files and listing files.



Assembler Directives

- Pseudo-Instructions
  - Not translated into machine instructions
  - Providing information to the assembler
- Basic assembler directives
  - START
  - END
  - BYTE
  - WORD
  - RESB
  - RESW

Example Program with Object Code

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           30203D
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
```

Symbolic Operands

- Mnemonic code (or instruction name) → opcode.

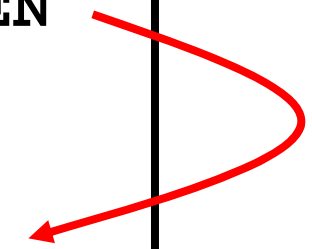
  STL 1033 → opcode 14 10 33

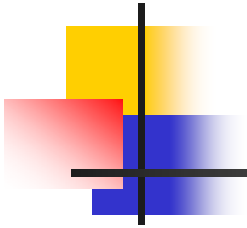
0001 0100	0	001 0000 0011 0011

- Use variable names instead of memory addresses
  - Labels (for jump instructions)
  - Subroutines
  - Constants

forward references

```
COPY  START 1000
      …
      LDA   LEN
      …
      …
LEN   RESW 1
```

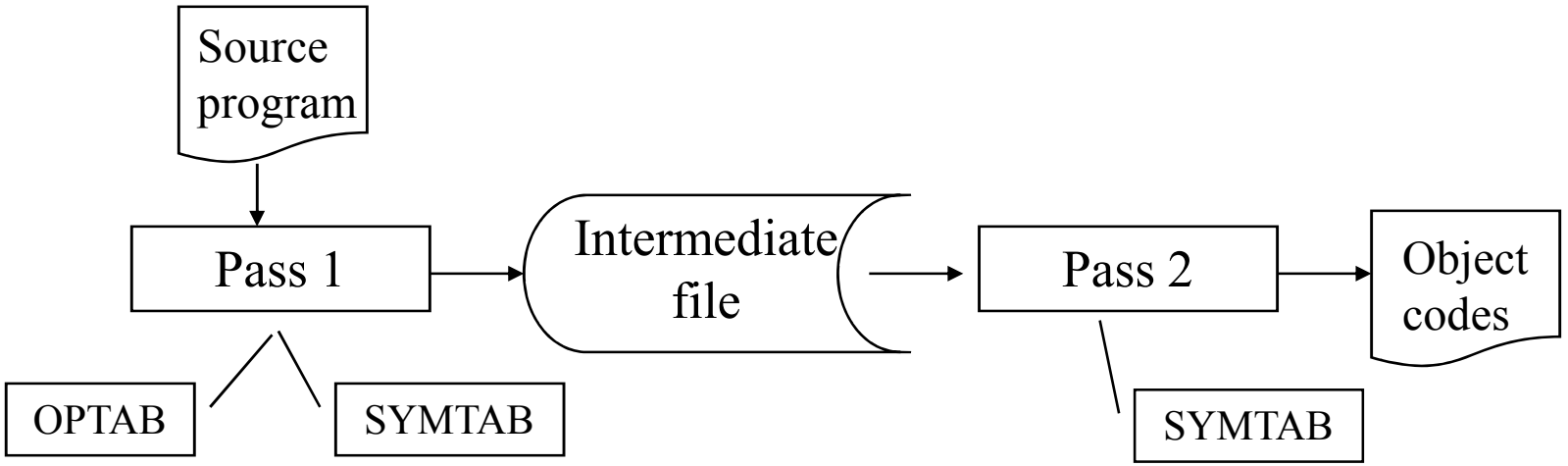




Two Pass Assembler

- Pass 1
  - Assign addresses (LOC) 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

Two Pass Assembler



Data Structures:

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

Two Pass Assembler – 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
                           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}
```

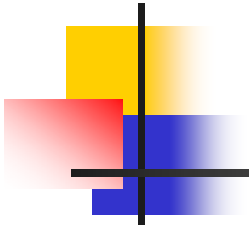
Figure 2.4(a) Algorithm for Pass 1 of assembler.

Two Pass Assembler – 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
               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.



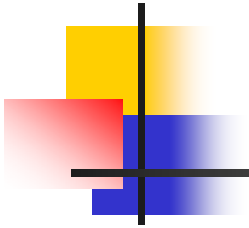
OPTAB (operation code table)

- Content
  - Mnemonic, machine code (instruction format, length) etc.
- Characteristic
  - Static table
- Implementation
  - Array or hash table, easy for search

SYMTAB (symbol table)

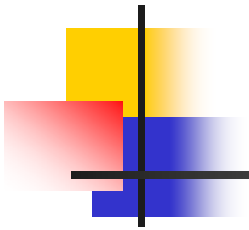
- Content
  - Label name, value, flag, (type, length) etc.
- Characteristic
  - Dynamic table (insert, delete, search)
- Implementation
  - Hash table, non-random keys, hashing function

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



Design & Implementation of Assembler

- Functions of a Basic Assembler
- **Object File**
- Address Translation
- Program Relocation
- Program Block
- Control Section & Program Linking
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler



Object Program

- Header

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

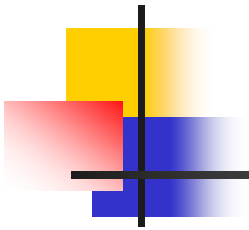
- Text

Col.1	T
Col.2~7	Starting address in this record (hex)
Col. 8~9	Length of object code in this record in bytes (hex)
Col. 10~69	Object code (69-10+1)/6=10 instructions

- End

Col.1	E
Col.2~7	Address of first executable instruction (hex) (END program_name)

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	



Object Program Example

H COPY 001000 00107A

T 001000 1E 141033 482039 001036 281030 301015 482061 ...

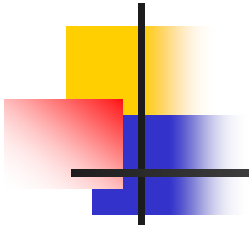
T 00101E 15 0C1036 482061 081044 4C0000 454F46 000003 000000

T 002039 1E 041030 001030 E0205D 30203F D8205D 281030 …

T 002057 1C 101036 4C0000 F1 001000 041030 E02079 302064 …

T 002073 07 382064 4C0000 05

E 001000 ←starting address



Design & Implementation of Assembler

- Functions of a Basic Assembler
- Object File
- **Address Translation**
- Program Relocation
- Program Block
- Control Section & Program Linking
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler

An SIC/XE Example

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	

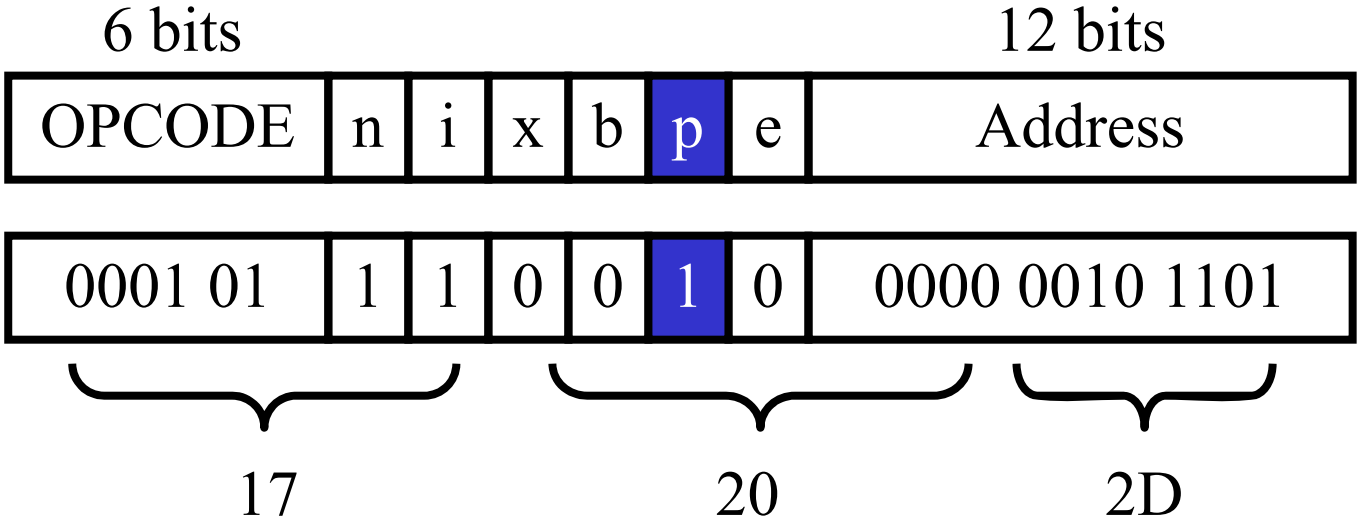
```
115           .       READ RECORD INTO BUFFER
120           .
125   1036    RDREC    CLEAR  X           B410
130   1038             CLEAR  A           B400
132   103A             CLEAR  S           B440
133   103C             +LDT   #4096       75101000
135   1040    RLOOP    TD     INPUT       E32019
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
195           .
200           .       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
              ...(omitted)
```

A Case of Object Code Generation

- Line 10

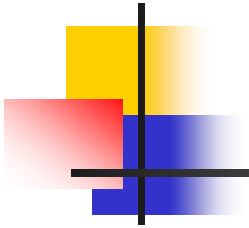
  `STL RETADR → 17 20 2D`

- The mode bit p=1, meaning PC relative addressing mode.



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)

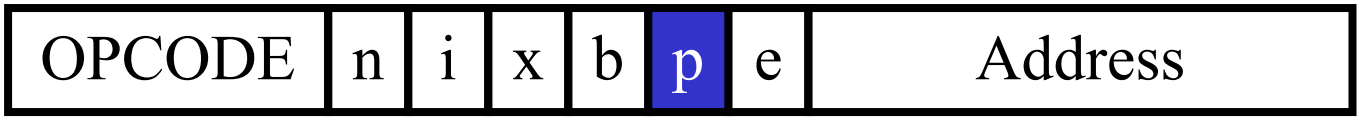


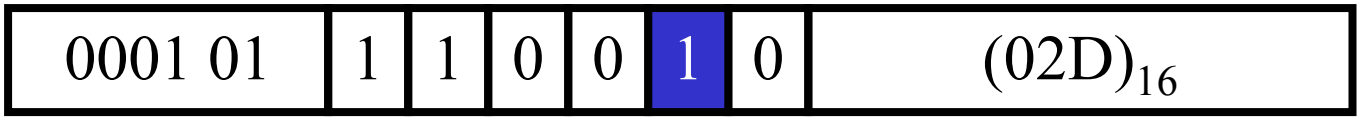
Translation

- 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

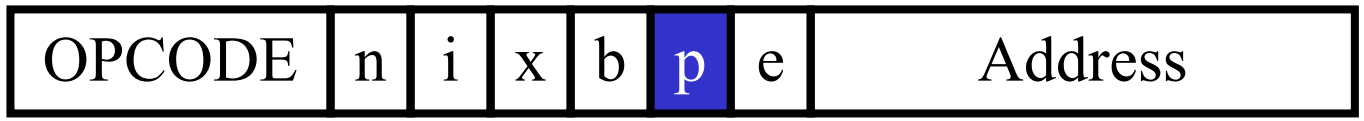
PC-Relative Addressing Mode

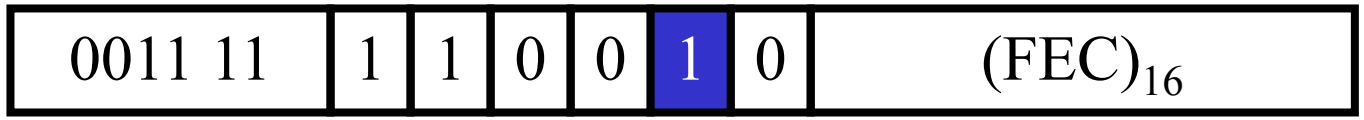
- PC-relative
  - 10 0000 FIRST STL RETADR 17202D





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





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

Base-Relative Addressing Modes

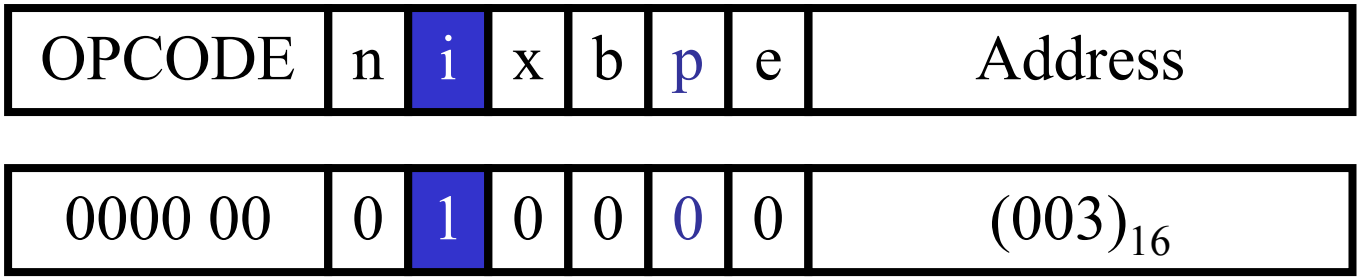
- Base-relative
  - Base register is under the control of the programmer
  - 12 LDB #LENGTH
  - 13 BASE LENGTH
  - 160 104E STCH BUFFER, X 57C003

OPCODE	n	i	x	b	p	e	Address
0101 01	1	1	1	1	0	0	$(003)_{16}$

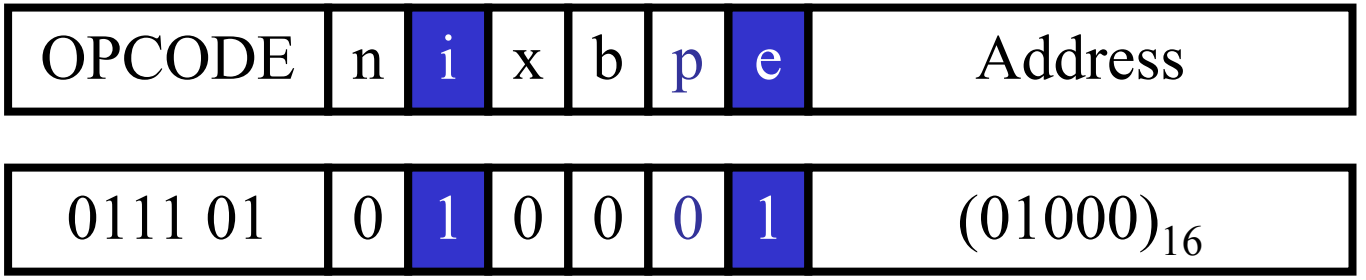
    - Displacement= BUFFER - B = 0036 - 0033 = 3
  - NOBASE is used to inform the assembler that the contents of the base register no longer be relied upon for addressing

Immediate Address Translation

- Immediate addressing
  - 55 0020 LDA #3 010003



  - 133 103C +LDT #4096 75101000



Immediate Address Translation

- Immediate addressing
  - 12 0003 LDB #LENGTH 69202D

OPCODE	n	i	x	b	p	e	Address
0110 10	0	1	0	0	1	0	$(02D)_{16}$

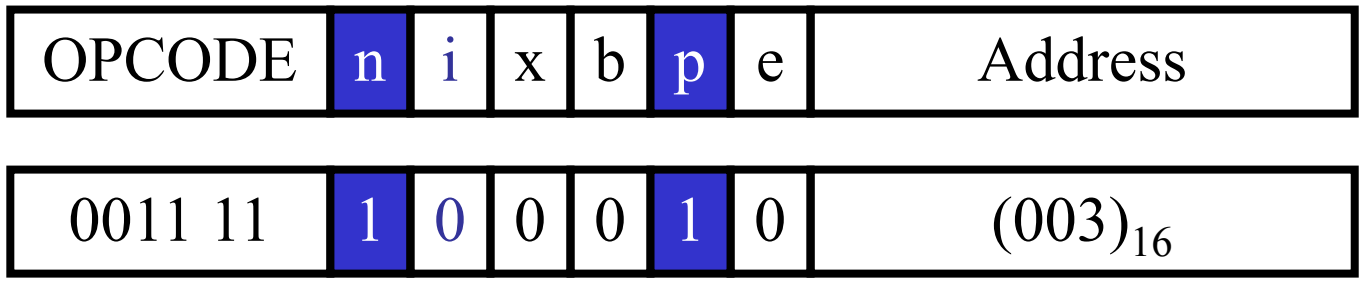
  - 12 0003 LDB #LENGTH 690033

OPCODE	n	i	x	b	p	e	Address
0110 10	0	1	0	0	0	0	$(033)_{16}$

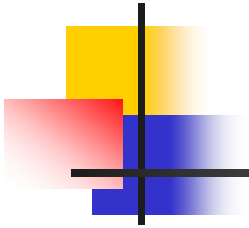
    - The immediate operand is the symbol LENGTH
    - The address of this symbol LENGTH is loaded into register B
    - LENGTH=0033=PC+displacement=0006+02D
    - If immediate mode is specified, the target address becomes the operand

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



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



Design & Implementation of Assembler

- Functions of a Basic Assembler
- Object File
- Address Translation
- **Program Relocation**
- Program Block
- Control Section & Program Linking
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler

Program Relocation

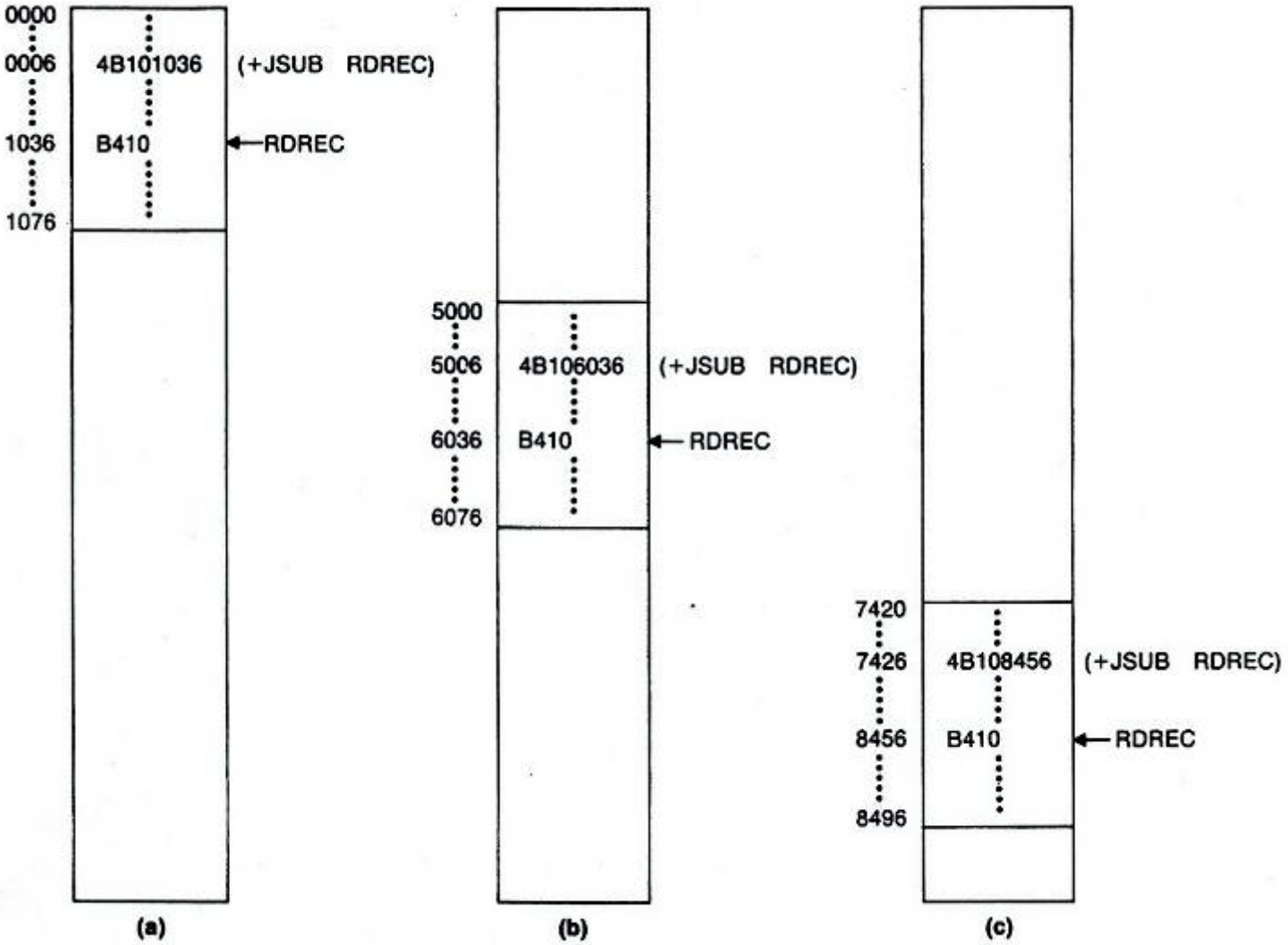
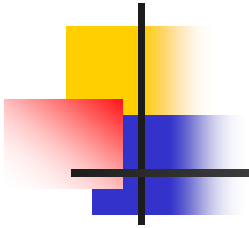


Figure 2.7 Examples of program relocation.

Examples of Program Relocation

- *Absolute program*, starting address 1000

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	WREC	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	WREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000
80	102A	EOF	BYTE	C'EOF'	454E46
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	

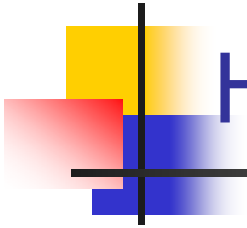


Problems of Program Relocation

- 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

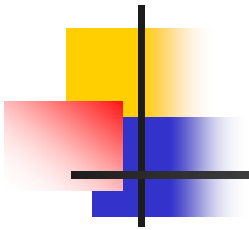
Examples of Program Relocation

5	0000	COPY	START	~~0~~ → 1000	
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	0036	BUFFER	RESB	4096	



How to Make Program Relocation Easier

- Use program-counter (PC) relative addresses
  - Did you notice that we didn't modify the addresses for **JEQ**, **JLT** and **J** instructions?
  - We didn't modify the addresses for **RETADR**, **LENGTH**, and **BUFFER**.
- Virtual memory
  - Every program pretends that it has all of memory. Therefore, Text segment always starts at a fixed address; Stack segment always resides a some huge high address.



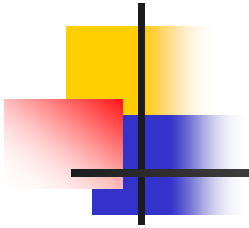
Relocatable Program

- 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 with Modification Record

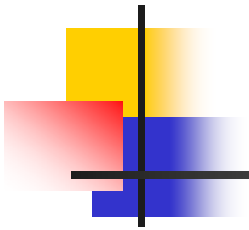
```
HCOPY  000000001077
T0000001D17202D69202D4B10103603202629000033200 74B10105D3F2FEC032010
T00001D130F20160100030F200D4B10105D3E2003454F46
T0010361DB410B400B44075101000E32019332FFADB2013A00433200857C003B850
T0010531D3B2FEA1340004F0000F1B410774000E32011332FFA53C003DF2008B850
T001070073B2FEF4F000005
M00000705
M00001405
M00002705
E000000
```

Figure 2.8 Object program corresponding to Fig. 2.6.



Design & Implementation of Assembler

- Functions of a Basic Assembler
- Object File
- Address Translation
- Program Relocation
- **Program Block**
- Control Section & Program Linking
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler



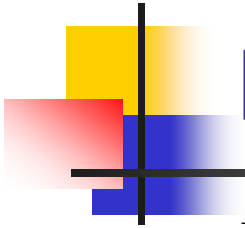
Program Blocks

- Refer to segments of code that are rearranged within a single object program unit
- **USE [blockname]**
- At the beginning, statements are assumed to be part of the unnamed (default) block
- If no USE statements are included, the entire program belongs to this single block
- Each program block may actually contain several separate segments of the source program

```
Line          Source statement
  5     COPY        START     0                 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      #0
 30                 JEQ       ENDFIL            EXIT IF EOF FOUND
 35                 JSUB      WRREC             WRITE OUTPUT RECORD
 40                 J         CLOOP             LOOP
 45     ENDFIL      LDA       =C'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
 92                 USE       CDATA
 95     RETADR      RESW      1
100     LENGTH      RESW      1                 LENGTH OF RECORD
103                 USE       CBLKS
105     BUFFER      RESB      4096              4096-BYTE BUFFER AREA
106     BUFEND      EQU       *                 FIRST LOCATION AFTER BUFFER
107     MAXLEN      EQU       BUFEND-BUFFER     MAXIMUM RECORD LENGTH
110     .                                       .
115     .           SUBROUTINE TO READ RECORD INTO BUFFER
120     .
123                 USE
125     RDREC       CLEAR     X                 CLEAR LOOP COUNTER
130                 CLEAR     A                 CLEAR A TO ZERO
132                 CLEAR     S                 CLEAR S TO ZERO
133                +LDT       #MAXLEN
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
183                 USE       CDATA
185     INPUT       BYTE      X'F1'             CODE FOR INPUT DEVICE
195     .
200     .           SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
208                 USE
210     WRREC       CLEAR     X                 CLEAR LOOP COUNTER
212                 LDT       LENGTH
215     WLOOP       TD        =X'05'            TEST OUTPUT DEVICE
220                 JEQ       WLOOP             LOOP UNTIL READY
225                 LDCH      BUFFER,X          GET CHARACTER FROM BUFFER
230                 WD        =X'05'            WRITE CHARACTER
235                 TIXR      T                 LOOP UNTIL ALL CHARACTERS
240                 JLT       WLOOP               HAVE BEEN WRITTEN
245                 RSUB                        RETURN TO CALLER
252                 USE       CDATA
253                 LTORG
255                 END       FIRST
```

Figure 2.11 Example of a program with multiple program blocks.

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

Program Blocks - Implementation

- 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

Object Code

```
HCOPY  000000001071
T0000001E1720634B2021032060290000332006 4B203B3F2FEE032055 0F2056010003
T00001E090F20484B20293E203F
T0000271DB410B400B44075101000E32038332FFADB2032A00433200857A02FB850
T000044093B2FEA13201F4F0000
T00006C01F1
T00004D19B410772017E3201B332FFA53A016DF2012B8503B2FEF4F0000
T00006D04454F4605
E000000
```

Figure 2.13 Object program corresponding to Fig. 2.11.

Loading Program Blocks

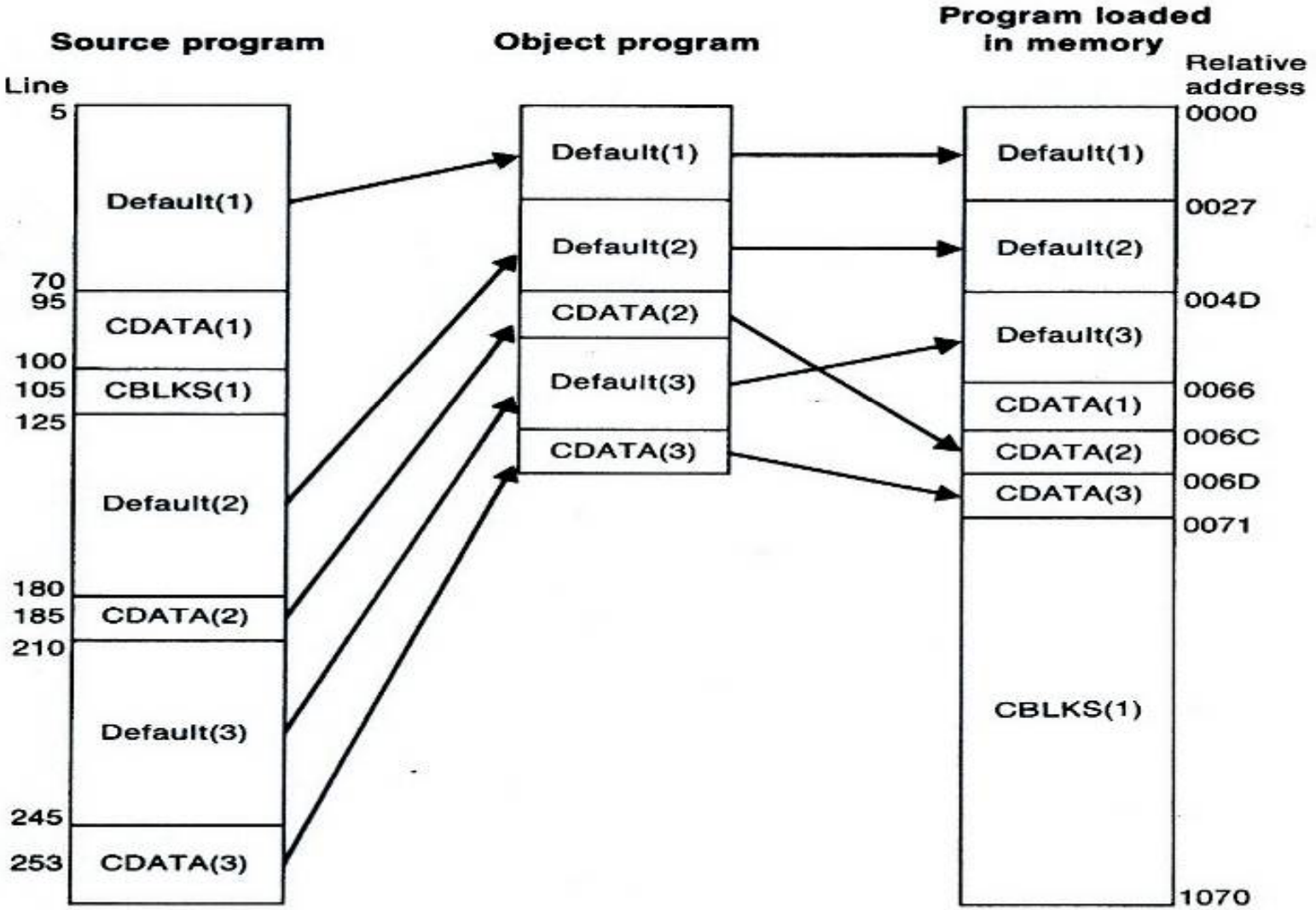
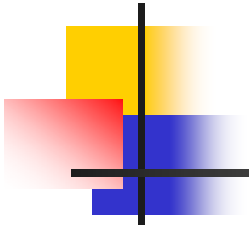
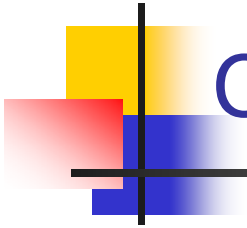


Figure 2.14 Program blocks from Fig. 2.11 traced through the assembly and loading processes.



Design & Implementation of Assembler

- Functions of a Basic Assembler
- Object File
- Address Translation
- Program Relocation
- Program Block
- **Control Section & Program Linking**
- Other Issues
  - One-pass Assembler
  - Multi-pass Assembler



Control Section

- 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

```
Line        Source statement

  5     COPY        START   0                   COPY FILE FROM INPUT TO OUTPUT
  6                 EXTDEF  BUFFER,BUFEND,LENGTH
  7                 EXTREF  RDREC,WRREC
 10     FIRST       STL     RETADR              SAVE RETURN ADDRESS
 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     =C'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
 95     RETADR      RESW    1
100     LENGTH      RESW    1                   LENGTH OF RECORD
103                 LTORG
105     BUFFER      RESB    4096                4096-BYTE BUFFER AREA
106     BUFEND      EQU     *
107     MAXLEN      EQU     BUFEND-BUFFER

109     RDREC       CSECT
110     .
115     .           SUBROUTINE TO READ RECORD INTO BUFFER
120     .
122                 EXTREF  BUFFER,LENGTH,BUFEND
125                 CLEAR   X                   CLEAR LOOP COUNTER
130                 CLEAR   A                   CLEAR A TO ZERO
132                 CLEAR   S                   CLEAR S TO ZERO
133                 LDT     MAXLEN
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
190     MAXLEN      WORD    BUFEND-BUFFER

193     WRREC       CSECT
195     .
200     .           SUBROUTINE TO WRITE RECORD FROM BUFFER
205     .
207                 EXTREF  LENGTH,BUFFER
210                 CLEAR   X                   CLEAR LOOP COUNTER
212                +LDT     LENGTH
215     WLOOP       TD      =X'05'              TEST OUTPUT DEVICE
220                 JEQ     WLOOP               LOOP UNTIL READY
225                +LDCH    BUFFER,X            GET CHARACTER FROM BUFFER
230                 WD      =X'05'              WRITE CHARACTER
235                 TIXR    T                   LOOP UNTIL ALL CHARACTERS
240                 JLT     WLOOP                  HAVE BEEN WRITTEN
245                 RSUB                        RETURN TO CALLER
255                 END     FIRST
```

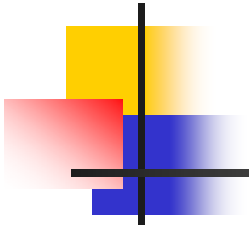
Figure 2.15 Illustration of control sections and program linking.

Line	Loc	Source statement			Object code
5	0000	COPY	START	0	
6			EXTDEF	BUFFER,BUFEND,LENGTH	
7			EXTREF	RDREC,WRREC	
10	0000	FIRST	STL	RETADR	172027
15	0003	CLOOP	+JSUB	RDREC	4B100000
20	0007		LDA	LENGTH	032023
25	000A		COMP	#0	290000
30	000D		JEQ	ENDFIL	332007
35	0010		+JSUB	WRREC	4B100000
40	0014		J	CLOOP	3F2FEC
45	0017	ENDFIL	LDA	=C'EOF'	032016
50	001A		STA	BUFFER	0F2016
55	001D		LDA	#3	010003
60	0020		STA	LENGTH	0F200A
65	0023		+JSUB	WRREC	4B100000
70	0027		J	@RETADR	3E2000
95	002A	RETADR	RESW	1	
100	002D	LENGTH	RESW	1	
103			LTORG		
	0030	*	=C'EOF'		454F46
105	0033	BUFFER	RESB	4096	
106	1033	BUFEND	EQU	*	
107	1000	MAXLEN	EQU	BUFEND-BUFFER	
109	0000	RDREC	CSECT		
110		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
122			EXTREF	BUFFER,LENGTH,BUFEND	
125	0000		CLEAR	X	B410
130	0002		CLEAR	A	B400
132	0004		CLEAR	S	B440
133	0006		LDT	MAXLEN	77201F
135	0009	RLOOP	TD	INPUT	E3201B
140	000C		JEQ	RLOOP	332FFA
145	000F		RD	INPUT	DB2015
150	0012		COMPR	A,S	A004
155	0014		JEQ	EXIT	332009
160	0017		+STCH	BUFFER,X	57900000
165	001B		TIXR	T	B850
170	001D		JLT	RLOOP	3B2FE9
175	0020	EXIT	+STX	LENGTH	13100000
180	0024		RSUB		4F0000
185	0027	INPUT	BYTE	X'F1'	F1
190	0028	MAXLEN	WORD	BUFEND-BUFFER	000000
193	0000	WRREC	CSECT		
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
207			EXTREF	LENGTH,BUFFER	
210	0000		CLEAR	X	B410
212	0002		+LDT	LENGTH	77100000
215	0006	WLOOP	TD	=X'05'	E32012
220	0009		JEQ	WLOOP	332FFA
225	000C		+LDCH	BUFFER,X	53900000
230	0010		WD	=X'05'	DF2008
235	0013		TIXR	T	B850
240	0015		JLT	WLOOP	3B2FEE
245	0018		RSUB		4F0000
255			END	FIRST	
	001B	*	=X'05'		05

Figure 2.16 Program from Fig. 2.15 with object code.

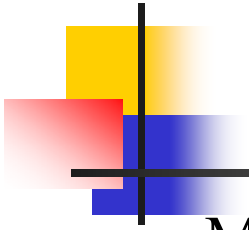
External Definition and References

- 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



Implementation

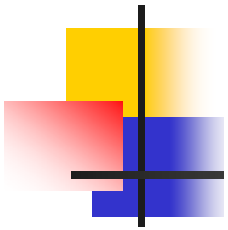
- The assembler must include information in the object program that will cause the loader to insert proper values where they are required
- Define record
  - Col. 1 D
  - Col. 2-7 Name of external symbol defined in this control section
  - Col. 8-13 Relative address within this control section (hexadeccimal)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



Modification Record

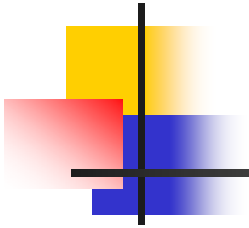
- Modification record
  - Col. 1 M
  - Col. 2-7 Starting address of the field to be modified (hexiadecimal)
  - Col. 8-9 Length of the field to be modified, in half-bytes (hexadeccimal)
  - Col.11-16 External symbol whose value is to be added to or subtracted from the indicated field
  - Note: control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

Object Code



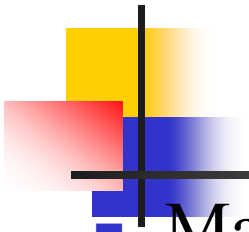
```
HCOPY  000000001033
DBUFFER000033BUFEND001033LENGTH00002D
RRDREC WRREC
T0000001D1720274B1000003202329000033200 74B1000003F2FEC0320160F2016
T00001D0D0100030F200A4B1000003E2000
T000003003454F46
M00000405+RDREC
M00001105+WRREC
M00002405+WRREC
E000000


HRDREC 000000000002B
RBUFFERLENGTHBUFEND
T0000001DB410B400B44077201FE3201B332FFADB2015A00433200957900000B850
T00001D0E3B2FE9131000004F0000F1000000
M00001805+BUFFER
M00002105+LENGTH
M00002806+BUFEND
M00002806-BUFFER
E


HWRREC 000000000001C
RLENGTHBUFFER
T0000001CB41077100000E32012332FFA53900000DF2008B8503B2FEE4F000005
M00000305+LENGTH
M00000D05+BUFFER
E
```

Figure 2.17 Object program corresponding to Fig. 2.15.



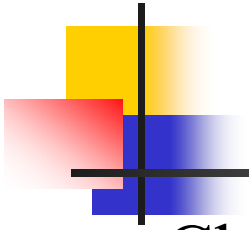
Design & Implementation of Assembler

- Functions of a Basic Assembler
- Object File
- Address Translation
- Program Relocation
- Program Block
- Control Section & Program Linking
- **Other Issues**
  - One-pass Assembler
  - Multi-pass Assembler



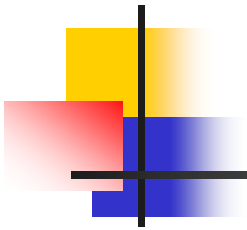
One-Pass Assemblers

- Main problem
  - Forward references
    - Data items & labels on instructions
- Solution
  - Require all such areas be defined before they are referenced
  - Labels on instructions: no good solution
- Two types of one-pass assemblers
  - Load-and-go
    - Produces object code directly in memory for immediate execution
  - The other
    - Produces usual kind of object code for later execution



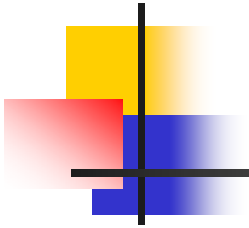
Load-and-go Assembler

- 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



Load-and-go Assembler

- Forward references handling
  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



Load-and-go Assembler

- 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

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		.			
115		.	SUBROUTINE TO READ RECORD INTO BUFFER		
120		.			
121	2039	INPUT	BYTE	X'F1'	F1
122	203A	MAXLEN	WORD	4096	001000
124		.			
125	203D	RDREC	LDX	ZERO	041006
130	2040		LDA	ZERO	001006
135	2043	RLOOP	TD	INPUT	E02039
140	2046		JEQ	RLOOP	302043
145	2049		RD	INPUT	D82039
150	204C		COMP	ZERO	281006
155	204F		JEQ	EXIT	30205B
160	2052		STCH	BUFFER,X	54900F
165	2055		TIX	MAXLEN	2C203A
170	2058		JLT	RLOOP	382043
175	205B	EXIT	STX	LENGTH	10100C
180	205E		RSUB		4C0000
195		.			
200		.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205		.			
206	2061	OUTPUT	BYTE	X'05'	05
207		.			
210	2062	WRREC	LDX	ZERO	041006
215	2065	WLOOP	TD	OUTPUT	E02061
220	2068		JEQ	WLOOP	302065
225	206B		LDCH	BUFFER,X	50900F
230	206E		WD	OUTPUT	DC2061
235	2071		TIX	LENGTH	2C100C
240	2074		JLT	WLOOP	382065
245	2077		RSUB		4C0000
255			END	FIRST	

Figure 2.18 Sample program for a one-pass assembler.

After Scanning Line 40

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	100948--	--00100C	28100630	------48--
2020	--3C2012			
•				
•				
•				

Symbol	Value		
LENGTH	100C		
RDREC	*	→	2013 | 0
THREE	1003		
ZERO	1006		
WRREC	*	→	201F | 0
EOF	1000		
ENDFIL	*	→	201C | 0
RETADR	1009		
BUFFER	100F		
CLOOP	2012		
FIRST	200F		

Figure 2.19(a) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.

After Scanning Line 160

Memory address	Contents			
1000	454F4600	00030000	00xxxxxx	xxxxxxxx
1010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
•				
•				
•				
2000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxx14
2010	10094820	3D00100C	28100630	202448——
2020	——3C2012	0010000C	100F0010	030C100C
2030	48————08	10094C00	00F10010	00041006
2040	001006E0	20393020	43D82039	28100630
2050	————5490	0F		
•				
•				
•				

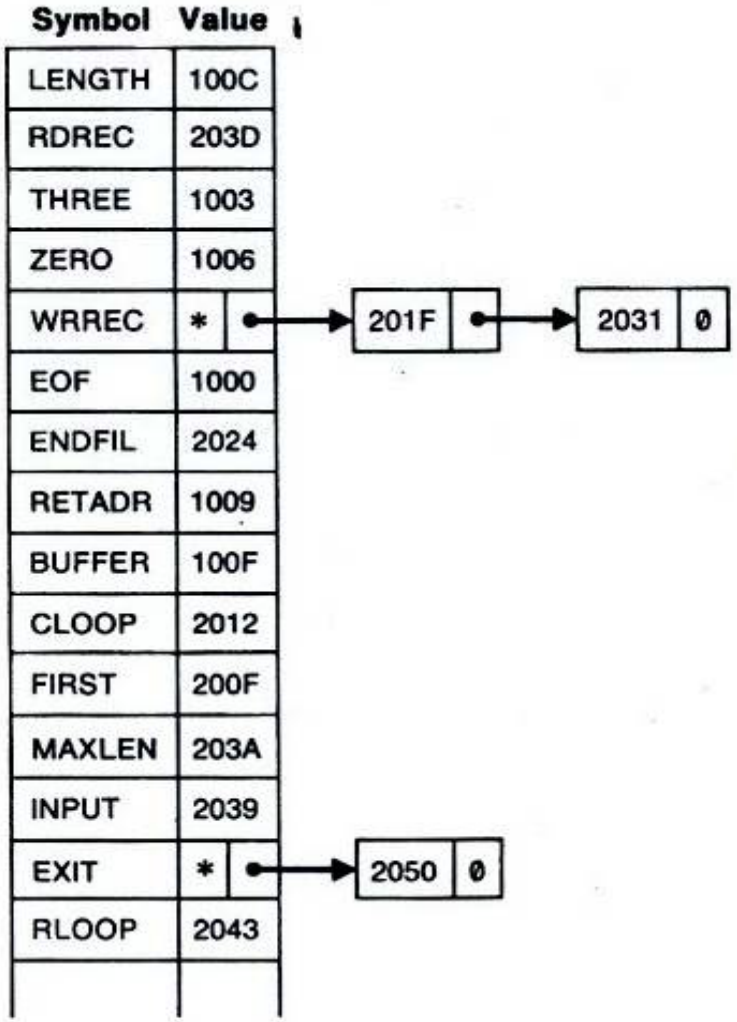
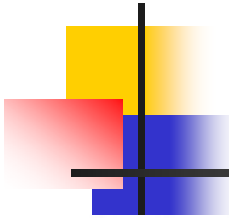


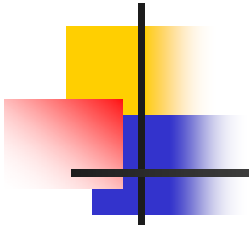
Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

Object Code



```
HCOPY  001000001007A
T001000094454F460000030000000
T00200F151410094800000100C2810063000004800003C2012
T00201C022024
T00202419001000C100F0010030C100C480000081009 4C0000F1001000
T002013022030D
T00203D1E041006001006E02039302043D82039281006300000549 00F2C203A382043
T002050022058B
T00205B0710100C4C000005
T00201F022062
T002031022062
T00206218041006E02061302065509 00FDC2061 2C100C3820654C0000
E00200F
```

Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.



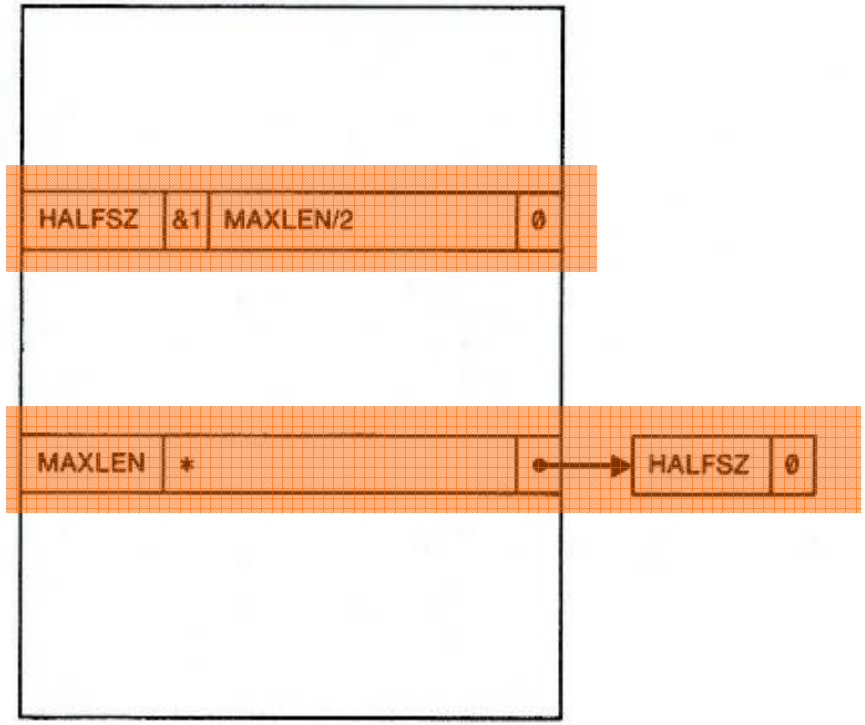
Multi-Pass Assemblers

- Restriction on EQU
  - No forward reference, since symbols' value can't be defined during the first pass
- Example
  - Use link list to keep track of whose value depend on an undefined symbol

Example of Multi-pass Assembler

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

(a)



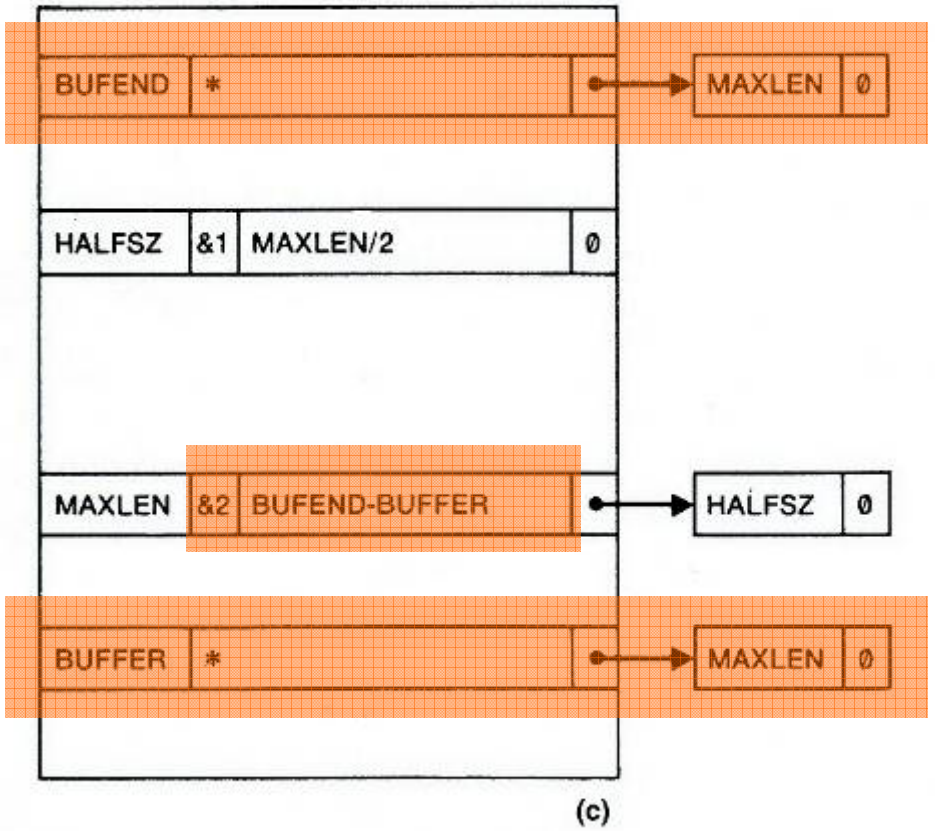
(b)

Figure 2.21 Example of multi-pass assembler operation.

Example of Multi-pass Assembler

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

(a)

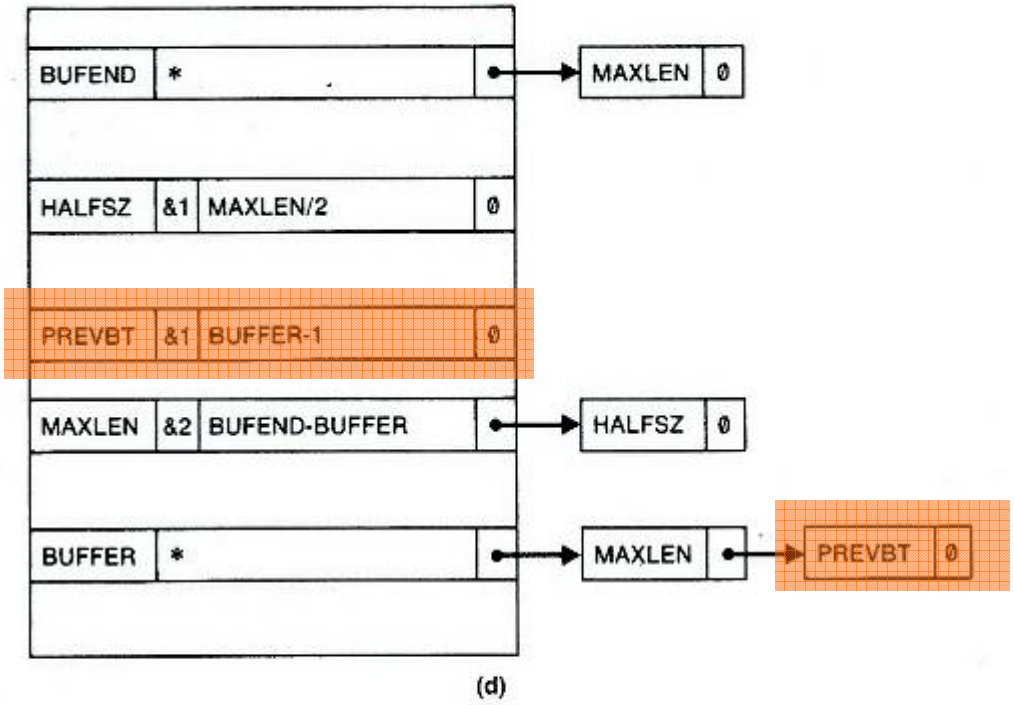


(c)

Example of Multi-pass Assembler

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

(a)



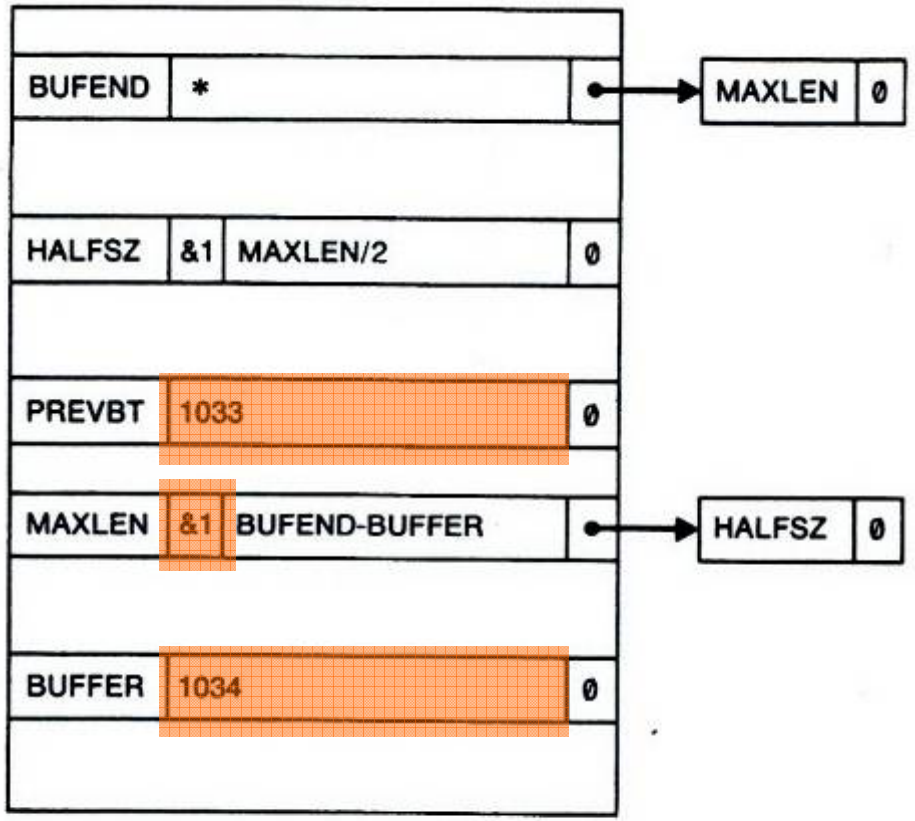
(d)

Figure 2.21 (*cont'd*)

Example of Multi-pass Assembler

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

(a)

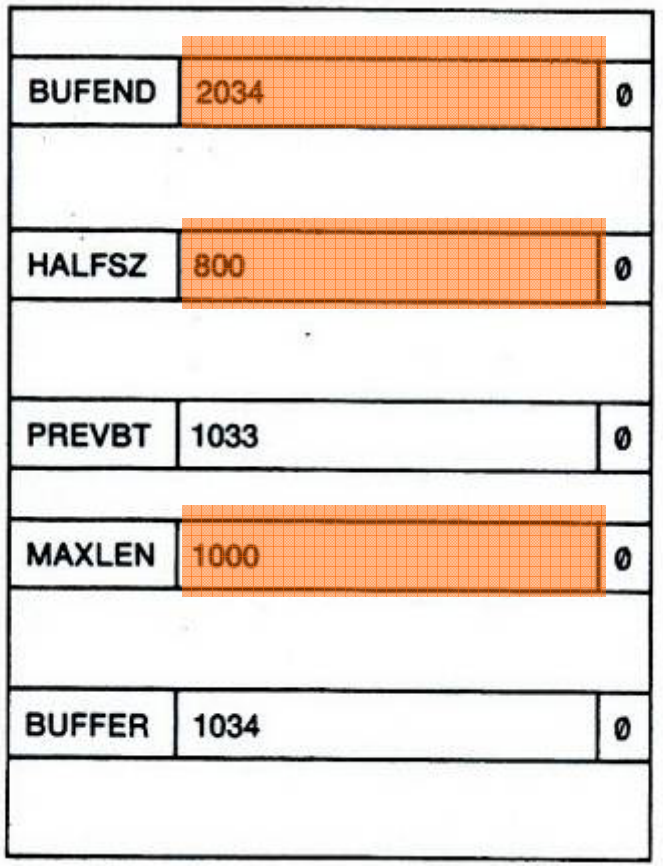


(e)

Example of Multi-pass Assembler

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

(a)



(f)

Figure 2.21 (*con'd*)