18CS61 (M1)

Topics

Introduction to System Software

Machine Architecture of SIC and SIC/XE

Assemblers:

Basic Assembler functions

Machine dependent assembler features

Machine independent assembler features

Assembler design options.

Basic Loader Functions.

18CS61 (M1 PYQs)

1. Explain in detail SIC/XE machine architecture
2. Write an SIC/XE program to calculate DELTA = ALPHA + BETA * GAMMA - 10
3. Write an algorithm for Pass-1 of an assembler
4. Generate the object code for the following SIC/XE source program.

```
SUM     START   0
FIRST   CLEAR   X
        LDA     #0
        LDB     #TOTAL
        BASE    TOTAL
LOOP    ADD     TABLE, X
        TIX     COUNT
        JLT     LOOP
        STA     TOTAL
COUNT   RESW    1
TABLE   RESW    2000
TOTAL   RESW    1
        END     FIRST
```

Mnemonic	ADD	JLT	LDA	LDB	LDX	RSUB	STA	TIC	JSUB
Opcode	18	38	00	68	04	4C	0C	2C	08

J	LDT	CLEAR
3C	74	B4

5- Define System Software. Distinguish b/w system software and application software.

6. Explain the data structures and pass-1 algorithm of SIC assembler.

7. Write the SIC/XE pgm for a bootstrap loader with suitable comments. Explain in brief the algorithm of a bootstrap loader.

8. What is a loader? Mention its advantages and disadvantages. What are the basic operations loader has to perform.

9. What are the control sections? How they are processed?

10. What are the basic functions of an assembler?

11. Write the following formats:

i) Header ii) Text iii) End

12. What are program blocks? Explain a program with multiple program blocks.

13. Explain multipass assembler.

14. Generate the machine code for the following:

(i) 0000 +JSUB RDREC

(ii) 0004 STL RETADR

(iii) 0008 LDB #LENGTH

(iv) 000A CLEAR X

Assume the opcodes are:

JSUB = 48_H

STL = 14_H

LDB = 60_H

CLEAR = $B4_H$

The LC value for RDREC = 1036_H

RETADR = 0030_H, LENGTH = 0033_H

The mnemonics values for registers are

A = 0, X = 1, L = 2, B = 3, S = 4, T = 5, F = 6, PC = 8

SW = 9

15. Write a SIC/XE program to copy the string "COMPUTER SCIENCE ENGINEERING" from STR1 to another string STR2.

16. List the various machine independent/dependent assembler features. Explain the control sections, how the assembler convert them into object code.

↓

17. Generate the complete object program for the following SIC/XE assembly program

```
WRREC   START   4050
        CLEAR   X
        LDT     LENGTH
WLOOP   TD      OUTPUT
        JEQ     WLOOP
        LDCH    BUFFER,X
        WD      OUTPUT
        TIXR    T
        JLT     WLOOP
        RSUB
OUTPUT  BYTE    X '05'
        END
```

Address of BUFFER : 4033

Address of length : 4036

Op codes:

CLEAR - B4; JEQ - 30; WD - DC; JLT - 38;

LDT - 74; LDCH - 50; TIXR - B8; RSUB - 4C;

Flow of video

① Difference b/w System Software and Appln Software

② SIC architecture and SIC/XE architecture
- memory, Register ... I/O

③ Writing SIC/XE programs

④ Assembler : ① Its use, adv/disadv, Assembler directives
basics of assembler; SIC → op code → op pgm
→ Algorithms (pass1 & pass2) brief.
→ SYMTAB / OPTAB Basics

② Machine dependent Assembler features
→ Detailed 2pass Algo's
→ Numerical (convert to op code & op pgm)
→ Pgm relocation & modification code

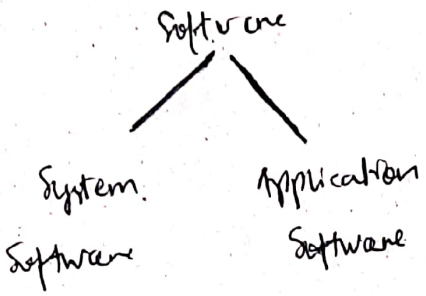
③ Machine independent Assembler features
→ literals
→ Symbol defining statements (SYMTAB)
→ Expressions
→ Program blocks
→ control section & Pgm Linking

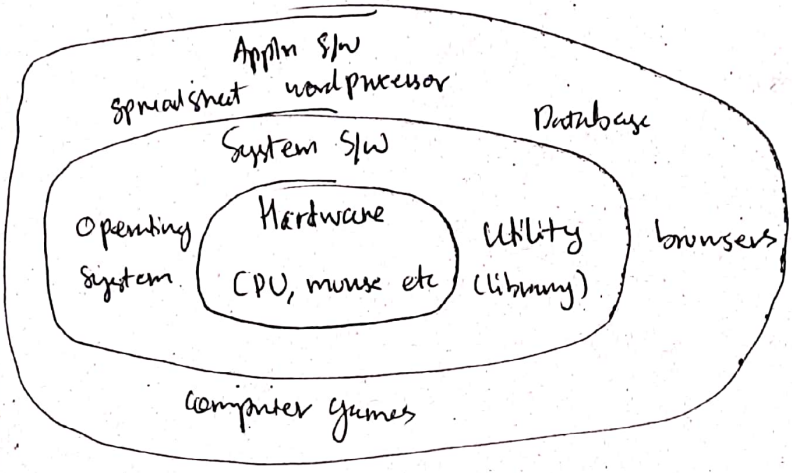
⑤ Assembler types: One pass - 2pass
→ Theory & its numericals

⑥ Loaders: Basics, Architecture, Algo of Bootstrap loader

What is "Software"?

"Software" refers to the set of electronic program instructions or data a computer processor reads in order to perform a task.





- Difference b/w System Software and Application Software

System Software	Application Software
1. Set of programs to manage the computer	1. Set of programs to permit user to perform group of functions.
2. Its written in a low-level language	2. Its written in a high level language.

3. Starts running when the system is turned on	3. Runs only when user executes
4. A system cant run without system software	4. A system does not require Appln software to run
5. It is general purpose	5. It is specific purpose
6. Eg:- Operating system, assembler, compiler, loader, linker etc.	6. Eg:- Web browser, word processing, spreadsheet, database, Adobe etc.
7. Machine Dependent	7. Not Machine Dependent.

C. System software and machine architecture

-> Machine architecture differs in:

- Machine code
- Instruction formats
- Addressing mode
- Registers

-> Machine independency of system software

General design and logic is basically the same:

- Code optimization
- Subprogram linking

SIC (Simplified Instructional Computer) Architecture

Every machine architecture includes

a) memory
b) Registers
c) Data formats
d) Instruction formats
e) Addressing modes
f) Instruction set
g) Input and output

a) Memory

- Memory consists of 8 bit bytes
- Any 3 consecutive bytes form a word (24 bits)
- All addresses on SIC are byte addresses.
- Total of 32768 (2^{15}) bytes in the computer memory

b) Registers

- Five registers, all of which have special uses
- Each register is 24 bits in length
- The table shows the mnemonic, number and uses of each register.

Mnemonic	Number	Uses
A Accumulator	0	Used for arithmetic operations
X Index Register	1	Used for addressing
L Linkage Register	2	JSUB - Jump to Sub Routine instruction stores return address
PC Program Counter	8	Contains the address of next instruction to be fetched for execution
SW Status Word	9	Contains a variety of info including a condition code in comp instruction

C) Data Formats

- Integers are stored as 24-bit binary number

$(0 - 2^{24}-1 \Rightarrow 0 \text{ to } 16G-1)$

- Negative values are represented as 2's complement

eg:- -24 is represented as

24 = 00011000

1's complement = 11100111

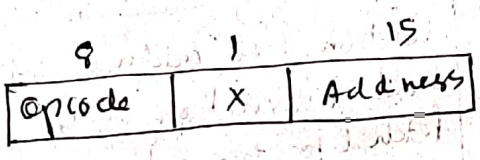
+ 1

11101000 => 232

- Characters are stored using their 8 bit ASCII code
- There is no floating point on the std. version of SIC.

d) Instruction Formats

→ All machine instructions on the standard version of SIC have 24-bit format.



X → indicates indexed addressing mode.

e) Addressing mode

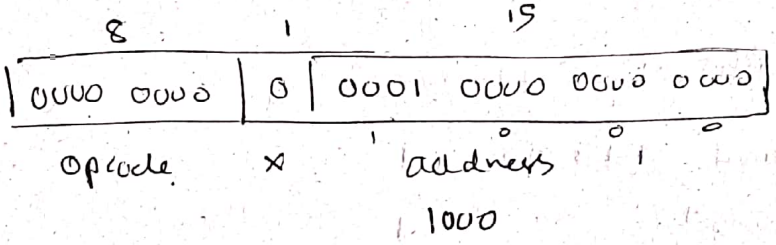
→ Two addressing modes based on X bit

- Direct Addressing
- Indexed Addressing

Direct addressing

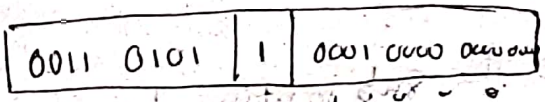
- X=0 ⇒ TA = address (Target Address)

eg:-



TA = 1000 is the content of address was loaded into Accumulator (A).

Indexed Addressing



TA = address + (x)

f> Instruction set

i) Load and Store : LDA, LDX, STA, STX

ii) Integer Arithmetic operations : ADD, SUB, MUL, DIV

eg:- ADD WORD;

A ← A + WORD

// Word is added to the existing register A

iii) Comparison Operations : COMP

eg:- COMP WORD; Compares A's contents with WORD and sets condition code as < = >

iv) Conditional Jump instructions : JLT, JEQ, JGT

These instructions will tell the setting of CC and jump accordingly.

(v) Subroutine Linkage Instructions: JSUB, RSUB

- JSUB - jumps to the subroutine by placing the return address in register L
- RSUB - returns by jumping to the address contained in register L

g) Input and Output

→ Input and Output is performed by transferring byte data at a time to or from the rightmost 8 bits of register A.

→ Each device is assigned a unique 8-bit code.

→ There are three I/O instructions which specify the device code as an operand.

i) TD (Test Device): Checks whether the addressed device is ready to send or recieve a byte of data. CC (condition code in SW register).

It is set accordingly (< =)

< -> device is ready to send/recieve

z → device is not ready.

→ The program has to wait until the device is ready and then perform Read Data (RD) or Write Data (WD).

RD: It transfers a byte of data from I/P device into rightmost byte of register A.

WD: It loads the data from rightmost byte of reg. A and writes to the Output Device.

Some SIC programs

1) SIC instructions for data movement operations (no memory-memory move instructions)

```
LDA   FIVE    ; Load constant 5 into register A
STA   ALPHA   ; Store in Alpha
LDCH  CHAR2   ; load character '2' into reg A
STCH  C1      ; store in character variable C1

ALPHA RESW  1   ; one-word variable
FIVE  WORD  5   ; one-word constant
CHAR2 BYTE  C '2' ; one-byte constant
C1    RESB  1   ; one-byte variable
```

2) SIC instructions for arithmetic operations

BETA = ALPHA + INCR - 1

DELTA = GAMMA + INCR - 1

```
LDA   ALPHA   ; Loads Alpha into register A
ADD   INCR    ; A ← (A) + (INCR)
SUB   ONE     ; A ← (A) - 1
STA   BETA    ; BETA ← (A)
LDA   GAMMA   ; A ← (GAMMA)
ADD   INCR    ; A ← (A) + (INCR)
```

```
SUB   ONE    ;   A ← (A) - 1
STA   DELTA  ;   DELTA ← (A)
```

```
ONE     WORD  1
ALPHA   RESW  1
BETA    RESW  1
GAMMA   RESW  1
DELTA   RESW  1
INCR    RESW  1
```

3) SIC instructions for looping and indexed operations (program to copy 11-byte character string to another string)

```
// LDX  ZERO;  initialize index register
   j=0
for (i=0; $1.[i] != '\0'; i++)
      S2[j++] = S1[i];
S2[j] = '\0';
```

```
LDX   ZERO ;  initialize index register to 0
LOOP  LDCH  STR1, X ;  copies the first character of
                       STR1 to reg A
                       (A = content of first byte of str1)
```

STCH STR2, X ; Store the first character into STR2

TIX ELEVEN ; Add 1 to index & compare to 11

X = 0+1 = 1 ; 1 ⟷ 11 cc will be set as <

JLT LOOP ; repeats if index is < 11
(X)

STR1 BYTE C 'HELLO WORLD'

STR2 RESB 11

ZERO WORD 0

ELEVEN WORD 11

4) Program to add 2 arrays of 100 words each and store it in another array. Each word is 3 bytes

∴ 100 words = 3×100
= 300 bytes

G = A+B

ADD LOOP	LDX	INDEX ;	initialize index value X = 0
	LDA	ALPHA, X ;	A ← (ALPHA)
	ADD	BETA, X ;	A ← (A) + [(BETA) at 0th byte (index)]
	STA	GAMMA, X ;	G ← [(A) at 0th byte]
	LDA	INDEX ;	A ← 0
	ADD	THREE ;	A ← (A)+3 = 3 → m

```
STA   INDEX    ; INDEX = 3
COMP  K300     ; A ↔ K300  i.e 3 ↔ 300  CC = <
JLT   ADDLOOP  ; repeat loop, now X = 3rd byte

K300   WORD   300
INDEX  RESW   1
ALPHA  RESW   100
THREE  WORD   3
BETA   RESW   100
GAMMA  RESW   100
```

5) To read one byte of data from input device and copies it to device OS.

```
INLOOP  TD    INDEV   ; Tests input device
        JEQ   INLOOP  ; CC := then loop until
                        device ready
        RD    INDEV   ; Once ready, read a byte
                        into reg A.
        STCH  DATA    ; Store it in data(memory)

        ;

OUTLOOP TD    OUTDEV  ; Test Output Device
        JEQ   OUTLOOP ; CC = then loop until
                        device ready
```

```
LDCH    DATA    ; Load data byte into reg. A
WD      OUTDEV  ; write one byte to output device

INDEV   BYTE    X 'F1'
OUTDEV  BYTE    X '05'
DATA    RESB    1
```

6) Subroutine call to read a 100-byte record from an input device into memory

```
        JSUB    READ    ; call Read subroutine wherein
                        ; it stores this return address
                        ; in linkage register.
        .
        .
        .
READ    LDX     ZERO    ; X ← 0
RLOOP   TD      INDEV   ; TEST input device
        JEQ     RLOOP   ; CC:= Loop until device is
                        ;      ready
        RD      INDEV   ; read one byte into reg A
        STCH    RECORD,X; store it into Record at
                        ;      Xth address
        TIX     K100    ; X = (X) + 1 & 1 ⟷ 100
                        ;      compare
        JLT     RLOOP   ; CC: < then Loop back
        RSUB            ; ~~Bat~~ Exit from subroutine;
                        ; it returns to the address
                        ; stored in linkage register.
```

```
INDEX BYTE X'F1'
RECORD RESB 100
ZERO WORD 0
HUNDRED WORD 100
```

SIC/XE machine architecture

→ SIC/XE: Simple Instructional Computer with Extra Equipments.

a) memory
b) Registers
c) Data formats
d) Instruction formats
e) Addressing modes
f) Instruction set.
g) Input/Output.

a) memory

→ memory consists of 8 bit bytes
→ 3 consecutive bytes form a word (24 bits).
→ All addresses are byte addresses.
→ Total of 1MB (2^{20} bytes) in the memory.

b) Registers

→ There are 9 registers.
→ Each register is 24 bits in length except floating point reg.

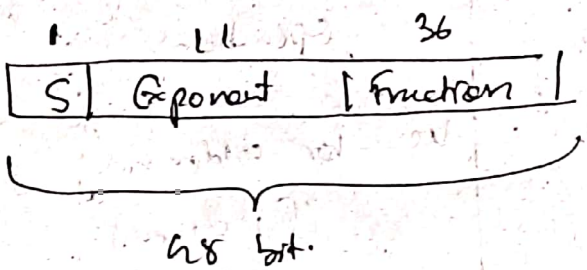
The registers are A, X, L, B, S, T, F, PC & SW

Mnemonic	number	Uses
A Accumulator	0	Used for arithmetic operations
X Index Register	1	Used for addressing (indexed)
L Linkage Register	2	Used to store the return address for JSUB instruction.
B Base Register	3	Used for addressing
S General Register	4	General working register no spl use
T General Register	5	General working register no spl use.
F Floating pt accumulator	6	Floating point accumulator (48 bits)
PC Program Counter	8	contains address of next instruction to be executed
SW Status word	9	Contains a variety of information including condition code (cc).

01) Data Formats

-> Integers are stored using 24-bit binary numbers.

=> Negative values are represented as 2's complement.



eg- 48 bit representation

100. ?

0.89 x 2

0.78 x 2 → 1

0.56 x 2 → 1

0.12 x 2 → 1

0.24 x 2 → 0

⋮

i.e. 100. 11000 ...

or

0. 100 111 000 ... x 2^{3} → Exponent. 011

d) Instruction Formats

→ There are two possible options.

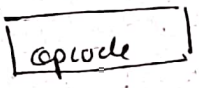
(i) Either use some form of relative addressing

(ii) Extend the address field to 20 bits.

→ If e = 0, then format 3

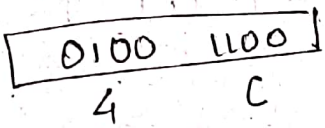
→ If e = 1, then format 4

1) Format 1 (1 byte)



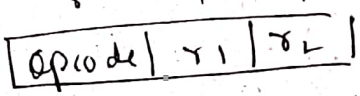
ex: RSUB → 4C (return to subroutine)

⇒ it returns to the address stored in Linkage register.



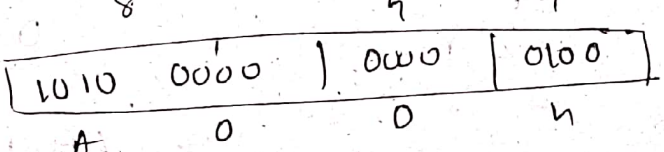
→ object code

2) Format 2 (2 bytes)



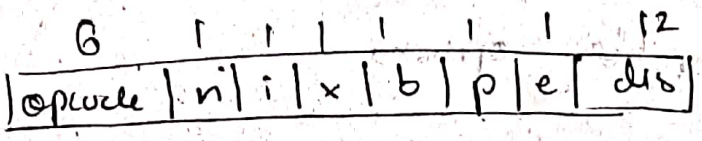
eg: COMPR A,S (compare the contents of registers A & S)

opcode of COMPR = A0



→ obj. code.

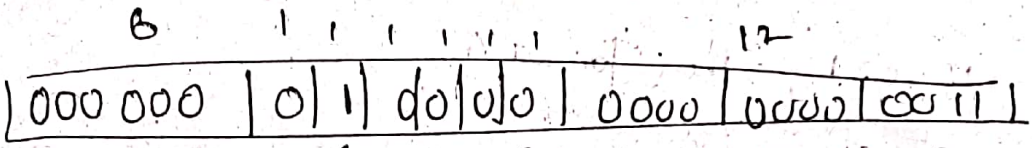
3) Format 3 (3 bytes)



n = indirect bit
i = immediate bit
x = index bit
b = base bit
p = pc relative bit
e = extended bit

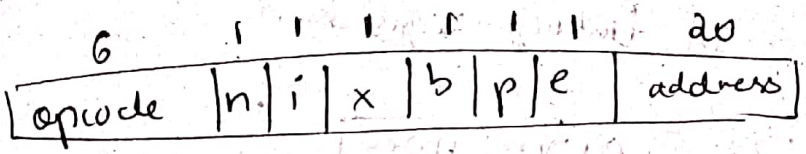
Note: e=0

eg:- LDA #3 (load 3 to A)



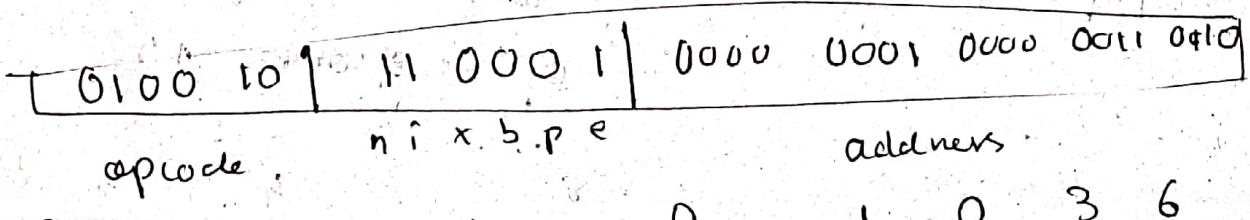
0 1003 → object code.

4) Format 4 (4 bytes)



eg:- + JSUB RDREC (Jumps to Address 1036)

opcode JSUB - 48



Object code = 4B 10 1036

e) Addressing modes

	Mode	Indication	Target Address Calculation
1.	Base Relative	b=1, p=0	TA = (B) + displacement (0 <= disp <= 4095)
2.	Program counter relative	b=0, p=1	TA = (P) + displacement (-2048 ≤ disp ≤ 2047)
3.	Direct Addressing	b=0, p=0 (for format 3/4)	TA = displacement/ address
4.	Base Relative Indexed Addressing	b=1, p=0 x=1	TA = (B) + (x) + displacement
5.	Program Relative Indexed Addressing	b=0; p=1 x=1	TA = (P) + (x) + displacement
6.	Immediate Addressing	i=1, n=0	Target Address itself is used TA = operand value (no memory reference)
7	Indirect addressing	i=0, n=1	TA = displacement value
8.	Simple addressing	i=0, n=0 or i=1, n=1	TA = location of operand

Special Symbols indication

1) #: Immediate address
2) @: Indirect address
3) +: Format 4
4) *: The current value of PC.
5) C'': Character String
6) op m, x: x - denotes the index register
7) base: Base-register.

Note:

Immediate addressing (i=1, n=0)

Target address itself is used as the operand value (no memory reference is performed)

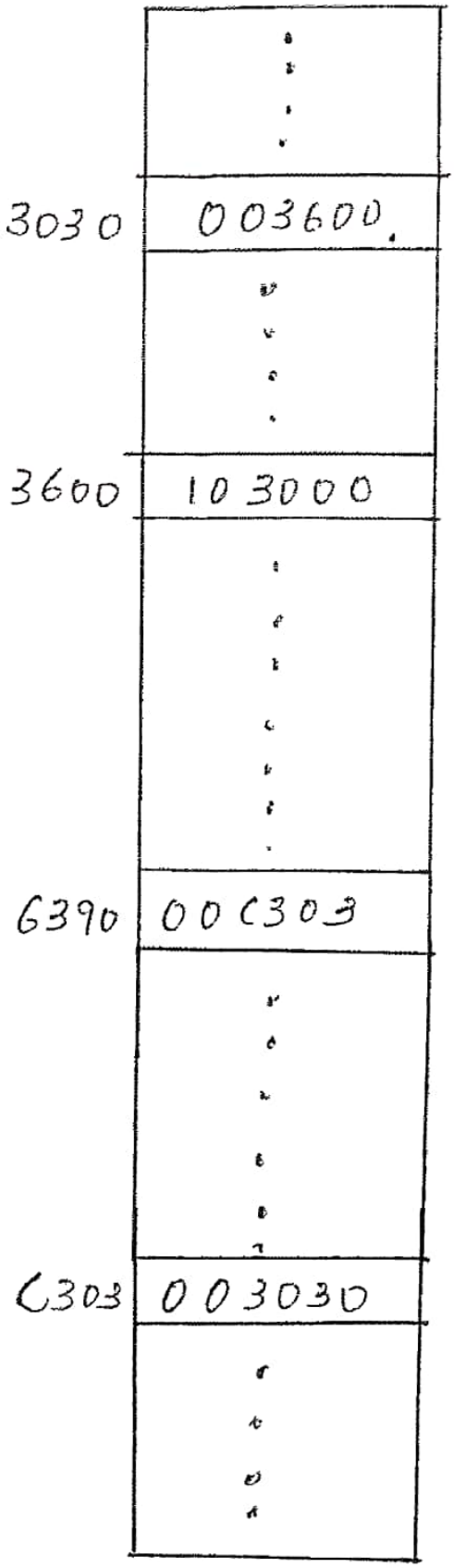
Indirect addressing (i=0, n=1)

The value in the address is taken as a address which is fetched and the value in that address is the real value.

Examples of SIC/XE instructions and addressing modes

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

Hex	Machine instruction (Binary)								Target address	Value loaded into Reg. A	Mode
	6 opcode	n	i	x	b	p	e	12/20 disp/address			
032600	000000 (0)	1 (3)	1	0	0	1 (2)	0	0110 0000 0000 (6 0 0)	3600	103000	Program Counter Relative TA = (PC) + displacement = 003000 + 600 = 3600
03C300	0000 00	1	1	1	1	0	0	0011 0000 0000	6390	00C303	Base relative Indexed TA = (B) + (X) + disp = 006000 + 000090 + 300 = 6390
022030	0000 00	1	0	0	0	1	0	0000 0011 0000	3030 (contain) 3600) (contain)	103000	Indirect + Program relative TA = (PC) + disp = 003000 + 030 = 3030
010030	0000 00	0	1	0	0	0	0	0000 0011 0000	30	000030	Immediate addressing TA is used as operand value
003600	0000 00	0	0	0	0	1	1	0110 0000 0000	3600	103000	PC relative TA = (PC) + disp = 003000 + 600 = 3600
0310C303	0000 00	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030	Simple addressing TA = location of operand



(B) = 006000

(PC) = 003000

(X) = 000090

Fig: contents of registers B, PC and X & memory locations

f) Instruction Set

* Load and Store instruction: LDA, LDX, STA, STX, LDB, STB

* Integer and Floating point arithmetic operations:

ADD, SUB, MUL, DIV, ADDF, SUBF, MULF, DIVF

* Register move instructions (RMO) => register to register operations such as ADDR, SUBR, MULR, DIVR.

* A special supervisor call (SVC) instruction is provided. Executing this instruction generates an interrupt that can be used for communication with the operating system.

- Comparison instruction: COMP, COMPR, COMPF
- Conditional jump instructions: JLT, JEQ, JGT
- Subroutine linkage instruction: JSUB, RSUB

g) Input and Output

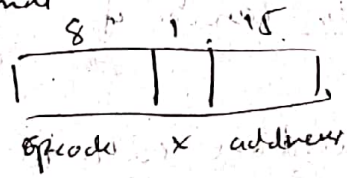
-> Same as SIC, transfer of 1 byte from or to the rightmost 8 bits of register A.

-> It has each device's unique id

-> It uses Test Device, < ready
= not ready

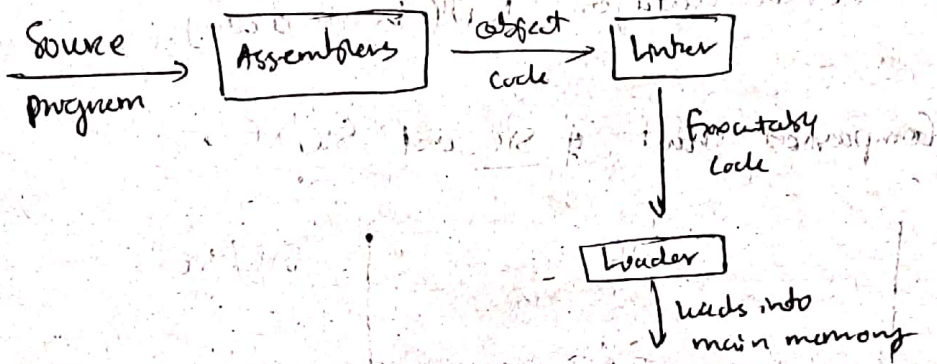
-> It does RD (read data) or WD (write data).

Comparison Chart of SIC and SIC/XE

Specification	SIC	SIC/XE
Memory	Word size: 3 bytes (24 bits) Total size: 32,768 bytes	Word size: 3 bytes (24 bits) Total Size: 32,768 bytes
Register	Total Registers: 5	Total Registers: 9
Instruction format	only one instruction format 	4 instruction format (format 1, format 2, format 3, format 4)
Addressing modes	Two possible modes Direct (x=0) Indexed (x=1)	Five possible modes Direct (x=0, b=0, p=0) n=i=0 or 1

	Relative (b/p = 1 other 0)
	Immediate (i=1, n=0)
	Indirect (i=0, n=1)
	Indexed (x=1)

Assemblers



Assembler does two functions:

1) It converts the mnemonic operation codes into their machine language equivalent

2) Convert symbolic labels into their machine addresses.

The design of assembler can be of:

1. Convert mnemonic operation codes to their machine language equivalent.
2. Convert symbolic operands to their equivalent machine addresses

3. Build the machine instructions in the proper format

4. Convert the data constants specified in the source program into their internal machine representation.

5. Write the object program and the assembly listing.

Different data structures for assemblers

1. Operation code Table (OPTAB)
2. Symbol Table (SYMTAB)
3. Location Counter Variable (LOCCTR)

I. Operation code Table (OPTAB)

a) Contents

- Mnemonic operation codes
- Machine language equivalent
- Instruction format and length

b) During pass-1

- Validates op codes
- Find instruction length to increase location Counter value (LOCCTR)

c) During pass-2

- Determines the instruction format (3 or 4)
- Translates the operation codes to their machine language equivalent.

d) Implementation

- Static hash table, easy for searching

Mnemonic Name	Opcode	Format
ADD m	18	3/4

II Symbol Table (SYMTAB)

a) Contents

- label name
- label address
- Flags to indicate error conditions
- Data type or length.

b) During pass-1

- Store label name and assigned address (from LOCCTR) in SYMTAB.

c) During pass-2

- Symbols used as operands are looked up in SYMTAB.

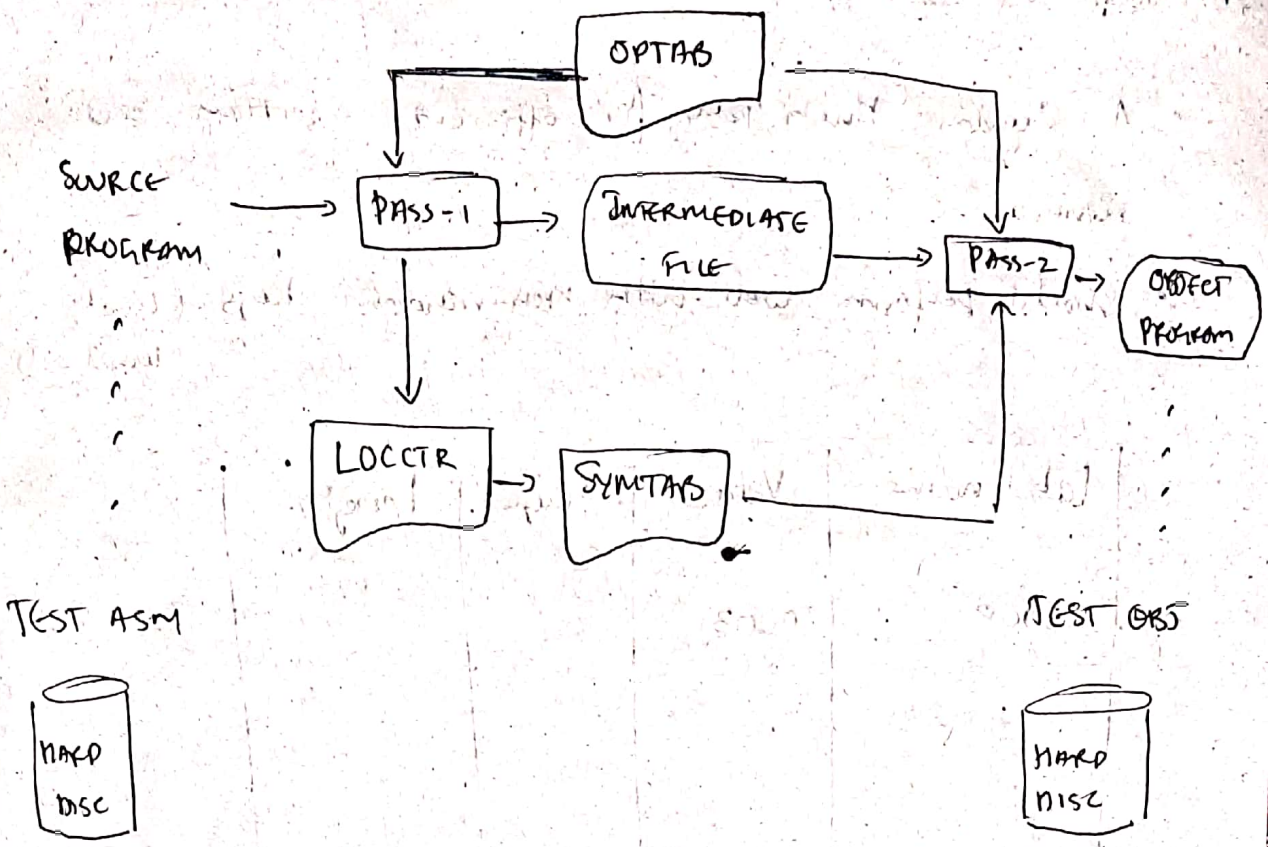
d) Implementation

- A dynamic Hash Table for efficient insertion and retrieval
- Should perform well with non-random keys (Loop1, Loop2...)

Label name	Value	Flags	Length
CLOOP	0003		

III LOCATION COUNTER VARIABLE (LOCCTR)

- Variable accumulated for address assignment.
  i.e. LOCCTR gives the address of the associated labels.
- LOCCTR is initialized at the beginning address specified in the "START" statement.
- After each source stmt is processed during pass-1, the instruction length or data area is added to LOCCTR.



Assembler Directives

1) START — Specifies name and starting address for the program.

2) END — Indicates the end of the source program

3) BYTE — Generate character or hexadecimal constant occupying as many bytes as needed to represent the constant.

4) WORD — Generate one-word integer constant.

5) RESB — Reserve the indicated number of bytes for a data area.

6) RESW Reserve the indicated no. of words for a data area.

7) LTORG Create a literal pool that contains all the literal operands used since the previous LTORG.

8) EQU Establishes symbolic names that can be used for improved readability inplace of numeric values.

9) ORG Used to indirectly assign values to symbols

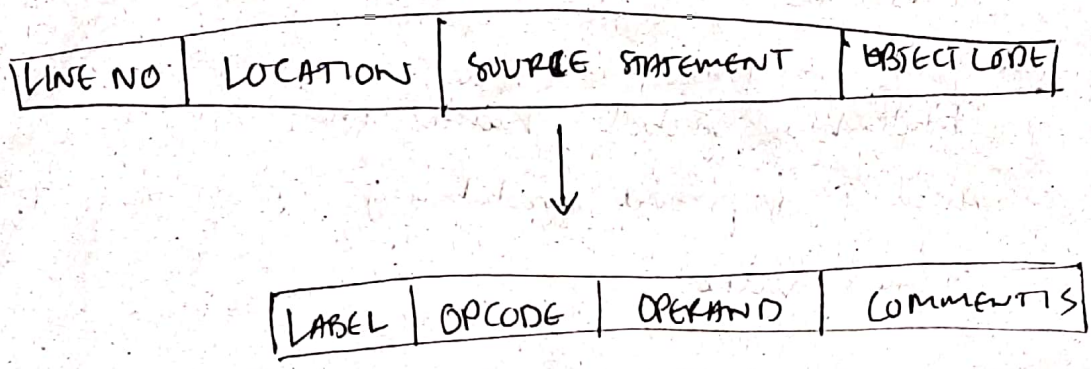
10) USE Indicates which portion of the source program belong to the various blocks

11) BASE Indicates that the base register will contain the addresses of operand.

12) NOBASE Indicates that the contents of the base register can no longer be relied upon for addressing.

A Simple SIC Assembler

The normal format to represent the assembly language program for SIC machine with generated assembly code.



Where

→ LABEL: An identifier and optional labels are used to reduce reliance upon programmers remembering.

Ex: FIRST STL #4096

→ OPCODE: Is a machine code instruction. It may require additional information like operands (optional)

Ex: COMP ZERO ;

→ OPERAND: Is an additional data or information that the opcode requires.

SIC: SIC/XE programs

1) Data movement operations

```
        LDA    #5       ; loads value 5 into register A
        STA    ALPHA    ; store in alpha : A ← (A)+(ALPHA)
                                              5 + (ALPHA)
        LDA    #90      ; load ascii code for '2' into A
        STCH   C1       ; store in charactr variable C1
        :
ALPHA   RESW   1        ; one-word variable
C1      RESB   1        ; one byte variable
```

2) Arithmetic operations (Beta = alpha + incr - 1)

```
        LDS    INCR     ; load value of incr to reg S
        LDA    ALPHA    ; load value of alpha to A
        ADDR   S, A     ; A ← (A)+(S)
               source destination
        SUB    #1       ; A ← (A)-1
        STA    BETA     ; BETA ← (A)
        :
```

3) Looping and indexed operations

GAMMA = ALPHA + BETA where ALPHA and BETA are arrays of 100 words each. (100 x 3 = 300 bytes)

```
        LDS    #3       ; S=3
        LDT    #300     ; T=300
        LDX    #0       ; X=0 → which specifies index value
ADDLOOP LDA    ALPHA,X  ; A ← (ALPHA) at the specified address of
                                       index register
        ADD    BETA,X   ; A ← (A)+(BETA)
        STA    GAMMA,X  ; GAMMA ← (A) at specified index value (of)
        ADDR   S,X      ; X ← (X)+(S) = 0+3 = 3 (index register
                                                 address is 3)
        COMPR  X,T      ; (X) ↔ (T) compared i.e 3 < 300, CC: <
        JLT    ADDLOOP  ; repeat loop till 300 = 300
```

```
        :
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
```

5) To read one byte of data from input device F1 and copies it to device 05

```
INLOOP  TD      INDEV
        JEQ     INLOOP
        RD      INDEV
        STCH    DATA
        :
OUTLOOP TD      OUTDEV
        JEQ     OUTLOOP
        LDCH    DATA
        WD      OUTDEV
        :
INDEV   BYTE    X'F1'
OUTDEV  BYTE    X'05'
DATA    RESB    1
```

6) subroutine call to read 100-byte record from an input device into memory

```
        JSUB    READ
        :
READ    LDX     #0
        LDT     #100
RLOOP   TD      INDEV
        JEQ     RLOOP
        RD      INDEV
```

```
        STCH    RECORD,X
        TIXR    T
        JLT     RLOOP
        RSUB
        :
INDEV   BYTE    X'F1'
RECORD  RESB    100
```

Write a SIC and SIC/XE program to copy 'SYSTEM SOFTWARE' to another string.

a) SIC program

```
        LDX     ZERO
LOOP    LDCH    STR1,X
        STCH    STR2,X
        TIX     FIFTEEN
        JLT     LOOP
        :
STR1    BYTE    C'SYSTEM SOFTWARE'
STR2    RESB    15
ZERO    WORD    0
FIFTEEN WORD    15
```

b) SIC/XE program

```
        LDX     #0
        LDT     #15
LOOP    LDCH    STR1,X
        STCH    STR2,X
        TIXR    T
        JLT     LOOP
        :
STR1    BYTE    C'SYSTEM SOFTWARE'
STR2    RESB    15
```

Exercises 1.3

1. Write a sequence of instructions for SIC to set ALPHA equal to the product of BETA and GAMMA. Assume ALPHA, BETA and GAMMA are 1 word (ALPHA = BETA * GAMMA)

```
LDA   BETA
MUL   GAMMA
STA   ALPHA
:
ALPHA RESW  1
BETA  RESW  1
GAMMA RESW  1
```

2. Write a sequence of instructions for SIC/XE to set ALPHA equal to H * BETA - 9. ALPHA, BETA and GAMMA are 1 word. Use immediate addressing for the constants (A = H * B - 9)

```
LDA   BETA
LDS   #H
MULR  S, A
SUB   #9
STA   ALPHA
:
ALPHA RESW  1
```

3. Write SIC instruction to swap the values of ALPHA and BETA.

```
LDA   ALPHA
STA   GAMMA
LDA   BETA
STA   ALPHA
LDA   GAMMA
STA   BETA
  :
ALPHA   RESW  1
BETA    RESW  1
GAMMA   RESW  1
```

4. Write a sequence of instructions for SIC to set ALPHA equal to the integer portion of BETA ÷ GAMMA. ALPHA, BETA, GAMMA are 1 word each

```
LDA   BETA
DIV   GAMMA
STA   ALPHA
  :
ALPHA   RESW  1
BETA    RESW  1
GAMMA   RESW  1
```

OP

1. Write a SIC program to copy string 'SYSTEM SOFTWARE' to another string.

```
        LDX   ZERO      ; Initialize X to 3ero
MOVECH  LDCH  STR1,X    ; X specific indexing
        STCH  STR2,X
        TIX   FIFTEEN   ; increment X and compare
                        with 15
        JLT   MOVECH
        .
STR1    BYTE  C 'SYSTEM SOFTWARE'
STR2    RESB  15
ZERO    WORD  0
FIFTEEN WORD  15
```

MACHINE DEPENDENT ASSEMBLER FEATURES (SIC)

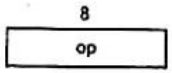
Mnemonic	Format	Opcode	Effect	Notes
ADD m	3/4	18	A ← (A) + (m..m+2)	
ADDF m	3/4	58	F ← (F) + (m..m+5)	X F
ADDR r1,r2	2	90	r2 ← (r2) + (r1)	X
AND m	3/4	40	A ← (A) & (m..m+2)	
CLEAR r1	2	B4	r1 ← 0	X
COMP m	3/4	28	(A) : (m..m+2)	C
COMPF m	3/4	88	(F) : (m..m+5)	X F C
COMPR r1,r2	2	A0	(r1) : (r2)	X C
DIV m	3/4	24	A ← (A) / (m..m+2)	
DIVF m	3/4	64	F ← (F) / (m..m+5)	X F
DIVR r1,r2	2	9C	r2 ← (r2) / (r1)	X
FIX	1	C4	A ← (F) [convert to integer]	X F
FLOAT	1	C0	F ← (A) [convert to floating]	X F
HIO	1	F4	Halt I/O channel number (A)	P X
J m	3/4	3C	PC ← m	
JEQ m	3/4	30	PC ← m if CC set to =	
JGT m	3/4	34	PC ← m if CC set to >	
JLT m	3/4	38	PC ← m if CC set to <	
JSUB m	3/4	48	L ← (PC); PC ← m	
LDA m	3/4	00	A ← (m..m+2)	
LDB m	3/4	68	B ← (m..m+2)	X
LDCH m	3/4	50	A [rightmost byte] ← (m)	
LDF m	3/4	70	F ← (m..m+5)	X F
LDL m	3/4	08	L ← (m..m+2)	
LDS m	3/4	6C	S ← (m..m+2)	X
LDT m	3/4	74	T ← (m..m+2)	X
LDX m	3/4	04	X ← (m..m+2)	
LPS m	3/4	D0	Load processor status from information beginning at address m (see Section 6.2.1)	P X
MUL m	3/4	20	A ← (A) * (m..m+2)	

Mnemonic	Format	Opcode	Effect	Notes
MULF m	3/4	60	F ← (F) * (m..m+5)	X F
MULR r1, r2	2	98	r2 ← (r2) * (r1)	X
NORM	1	C8	F ← (F) [normalized]	X F
OR m	3/4	44	A ← (A) | (m..m+2)	
RD m	3/4	D8	A [rightmost byte] ← data from device specified by (m)	P
RMO r1,r2	2	AC	r2 ← (r1)	X
RSUB	3/4	4C	PC ← (L)	
SHIFTL r1,n	2	A4	r1 ← (r1); left circular shift n bits. [In assembled instruction, r2 = n–1]	X
SHIFTR r1,n	2	A8	r1 ← (r1); right shift n bits, with vacated bit positions set equal to leftmost bit of (r1). [In assembled instruction, r2 = n-1]	X
SIO	1	F0	Start I/O channel number (A); address of channel program is given by (S)	P X
SSK m	3/4	EC	Protection key for address m ← (A) (see Section 6.2.4)	P X
STA m	3/4	0C	m..m+2 ← (A)	
STB m	3/4	78	m..m+2 ← (B)	X
STCH m	3/4	54	m ← (A) [rightmost byte]	
STF m	3/4	80	m..m+5 ← (F)	X F
STI m	3/4	D4	Interval timer value ← (m..m+2) (see Section 6.2.1)	P X
STL m	3/4	14	m..m+2 ← (L)	
STS m	3/4	7C	m..m+2 ← (S)	X
STSW m	3/4	E8	m..m+2 ← (SW)	P
STT m	3/4	84	m..m+2 ← (T)	X
STX m	3/4	10	m..m+2 ← (X)	
SUB m	3/4	1C	A ← (A) – (m..m+2)	
SUBF m	3/4	5C	F ← (F) – (m..m+5)	X F

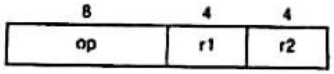
Mnemonic	Format	Opcode	Effect	Notes
SUBR r1,r2	2	94	r2 ← (r2) – (r1)	X
SVC n	2	B0	Generate SVC interrupt. [In assembled instruction, r1 = n]	X
TD m	3/4	E0	Test device specified by (m)	P C
TIO	1	F8	Test I/O channel number (A)	P X C
TIX m	3/4	2C	X ← (X) + 1; (X): (m..m+2)	C
TIXR r1	2	B8	X ← (X) + 1; (X): (r1)	X C
WD m	3/4	DC	Device specified by (m) ← (A) [rightmost byte]	P

Instruction Formats

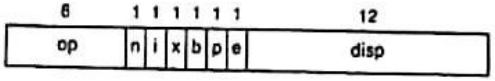
Format 1 (1 byte):



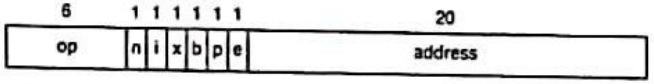
Format 2 (2 bytes):



Format 3 (3 bytes):



Format 4 (4 bytes):



Addressing Modes

The following addressing modes apply to Format 3 and 4 instructions. Combinations of addressing bits not included in this table are treated as errors by the machine. In the description of assembler language notation, *c* indicates a constant between 0 and 4095 (or a memory address known to be in this range); *m* indicates a memory address or a constant value larger than 4095. Further information can be found in Section 1.3.2.

The letters in the Notes column have the following meanings:

- 4 Format 4 instruction
- D Direct-addressing instruction
- A Assembler selects either program-counter relative or base-relative mode
- S Compatible with instruction format for standard SIC machine. Operand value can be between 0 and 32,767 (see Section 1.3.2 for details).

Addressing type	Flag bits n i x b p e	Assembler language notation	Calculation of target address TA	Operand	Notes
Simple	110000	op c	disp	(TA)	D
	110001	+op m	addr	(TA)	4 D
	110010	op m	(PC) + disp	(TA)	A
	110100	op m	(B) + disp	(TA)	A
	111000	op c,X	disp + (X)	(TA)	D
	111001	+op m,X	addr + (X)	(TA)	4 D
	111010	op m,X	(PC) + disp + (X)	(TA)	A
	111100	op m,X	(B) + disp + (X)	(TA)	A
	000- - -	op m	b/p/e/disp	(TA)	D S
	001- - -	op m,X	b/p/e/disp + (X)	(TA)	D S
Indirect	100000	op @c	disp	((TA))	D
	100001	+op @m	addr	((TA))	4 D
	100010	op @m	(PC) + disp	((TA))	A
	100100	op @m	(B) + disp	((TA))	A
Immediate	010000	op #c	disp	TA	D
	010001	+op #m	addr	TA	4 D
	010010	op #m	(PC) + disp	TA	A
	010100	op #m	(B) + disp	TA	A

Line	Source statement			
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
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	.			
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	

Figure 2.1 Example of a SIC assembler language program.

ASCII CODE
A = 65
a = 97

Line	Loc	Length	Source statement			Object code
			LABEL	OPCODE	OPERAND	
5	1000		COPY	START	1000	
10	1000	3	FIRST	STL	RETADR	141033
15	1003	3	CLOOP	JSUB	RDREC	482039
20	1006	3		LDA	LENGTH	001036
25	1009	3		COMP	ZERO	281030
30	100C	3		JEQ	ENDFIL	301015
35	100F	3		JSUB	WRREC	482061
40	1012	3		J	CLOOP	3C1003
45	1015	3	ENDFIL	LDA	EOF	00102A
50	1018	3		STA	BUFFER	0C1039
55	101B	3		LDA	THREE	00102D
60	101E	3		STA	LENGTH	0C1036
65	1021	3		JSUB	WRREC	482061
70	1024	3		LDL	RETADR	081033
75	1027	3		RSUB		4C0000
80	102A	3	EOF	BYTE	C'EOF'	454F46
85	102D	3	THREE	WORD	3	000003
90	1030	3	ZERO	WORD	0	000000
95	1033	3	RETADR	RESW	1	
100	1036	3	LENGTH	RESW	1	
105	1039	1000	BUFFER	RESB	4096	
110			.			
115			.	SUBROUTINE TO READ RECORD INTO BUFFER		
120			.			
125	2039	3	RDREC	LDX	ZERO	041030
130	203C	3		LDA	ZERO	001030
135	203F	3	RLOOP	TD	INPUT	E0205D
140	2042	3		JEQ	RLOOP	30203F
145	2045	3		RD	INPUT	D8205D
150	2048	3		COMP	ZERO	281030
155	204B	3		JEQ	EXIT	302057
160	204E	3		STCH	BUFFER,X	549039
165	2051	3		TIX	MAXLEN	2C205E
170	2054	3		JLT	RLOOP	38203F
175	2057	3	EXIT	STX	LENGTH	101036
180	205A	3		RSUB		4C0000
185	205D	1	INPUT	BYTE	X'F1'	F1
190	205E	3	MAXLEN	WORD	4096	001000
195			.			
200			.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205			.			
210	2061	3	WRREC	LDX	ZERO	041030
215	2064	3	WLOOP	TD	OUTPUT	E02079
220	2067	3		JEQ	WLOOP	302064
225	206A	3		LDCH	BUFFER,X	509039
230	206D	3		WD	OUTPUT	DC2079
235	2070	3		TIX	LENGTH	2C1036
240	2073	3		JLT	WLOOP	382064
245	2076	3		RSUB		4C0000
250	2079	1	OUTPUT	BYTE	X'05'	05
255	207A			END	FIRST	

Figure 2.2 Program from Fig. 2.1 with object code.

210	2061	3	WRREC	LDX	ZERO	041030
215	2064	3	WLOOP	TD	OUTPUT	E02079
220	2067	3		JEQ	WLOOP	302064
225	206A	3		LDCH	BUFFER,X	509039
230	206D	3		WD	OUTPUT	DC2079
235	2070	3		TIX	LENGTH	2C1036
240	2073	3		JLT	WLOOP	382064
245	2076	3		RSUB		4C0000
250	2079	1	OUTPUT	BYTE	X'05'	05
255	207A			END	FIRST	

Figure 2.2 Program from Fig. 2.1 with object code.

The following program contains a main routine that reads records from an input device (code: F1) and copies them to output device (code: 05). 6

main function calls subroutine RDREC to read a record into a buffer and subroutine WRREC to write record from the buffer to output device.

Each subroutine transfers one record one character at a time because only I/O instructions available are RD and WD.

Since the I/O rates of two devices (disk and a printing terminal) may be different, a buffer is used. The end of each record is marked with a null character i.e 00 (in hexadecimal). If a record is longer than length of buffer (4096 bytes) then only the first 4096 bytes are copied The end of file to be copied is indicated by a zero length record.

The program indicate EOF (End of File) on output device when the zero length record (ie end of file) is detected. The program terminate by executing the RSUB instruction since it was called by JSUB instruction.

Procedure to generate object code and object Program (Intermediate File).

note: we have assumed that the program starts at address 1000.

Procedure to generate object code and object Program (Intermediate File).

Note: We have assumed that the program starts at address 1000.

1) First and foremost write the LOCCTR addresses
   - → START 1000
   - → Add 3 byte for each instructions. (∵ instruction format for SIC m/c is 24bits. ie 3 byte)
   - → BYTE C 'EOF' : count the length of constant and add those many byte
   - → RESW 2000 : then it should be 2000x3 byte = 6000B = 1770 (H) added to previous address
   - → RESW 1 : add just 3 byte
   - → RESB 2000 : convert 2000 to hexadecimal (~~7D0~~ ie 7D0 byte and add

     RESB 4096 : $4096 \rightarrow 1000_{(16)}$ is added to previous value
   - → WORD 3 or WORD 0 → 3 byte added.

2) Start creating the object code.
   - → Convert mnemonic operation codes to their machine language equivalent. Ex:- STL to 14
   - → Convert symbolic operands to their equivalent machine address Ex BETADR to 1033 (forward reference)

   - → Build machine instruction in proper format
     - a) Direct addressing : X=0 : TA : address
     - b) Indexed addressing : X=1 : TA = address+(X)
       - → indicated by symbol 'X'
       - Ex:- STCH BUFFER, X :→ Line no. 160
   - → Convert the data constants into their machine representation. Ex:- EOF to 454F46 (Line no 80)

2) Start creating

→ Convert mnemonic operation codes to their machine language equivalent. Ex:- STL to 14

→ Convert symbolic operands to their equivalent machine address Ex BUFADR to 1033 (forward reference)

→ Build machine instruction in proper format

a) Direct addressing : X=0 : TA : address

b) Indexed addressing : X=1 : TA = address+(X)

→ indicated by symbol 'X'

Ex:- STCH BUFFER, X :→ Line no. 160

→ Convert the data constants into their machine representation. Ex:- EOF to H54F46 (Line no 80)

(A = 65, a = 97)

↳ $(41)_{16}$ ↳ $(61)_{16}$

3) Write the object program (Intermediate File)

→ object program contains three type of records.

a) Header Record b) Text Record c) End Record.

a) Header Record : Contains program name, starting address and length of program.

column 1	H
col. 2-7	Program name
col. 8-13	starting address of object Program (Hexadecimal)
col. 14-19	Length of object program in byte (Hexadecimal)

Ex:- 5 COPY START 1000 → name of program, ↳ starting address

:

255 207A END

Length of program = last address - startingaddress

= 207A - 1000 = 107A

:. H^COPY^001000^001074 (Header Record)

---

b) Text Record :

Text record contains the translated instructions (machine code) and data of the program together with an indication of addresses where these are to be loaded.

col. 1	T

Ex:- 5 COPY START 1000 → name of program, → starting address

⋮

25B 207A END

Length of program = last address - starting address

= 207A - 1000 = 107A

∴ H^COPY^001000^00107A (Header Record)

b) Text Record :

Text record contains the translated instructions (machine code) and data of the program together with an indication of addresses where they are to be loaded.

col. 1	T
col. 2-7	starting address for object code in this record (Hexadecimal)
col 8-9	Length of object code in this record in bytes (hexadecimal)
col 10-69	object code represented in hexadecimal (2 columns per byte of object code)

↓ note.

60 columns

⇒ 10 words ⇒ 30 bytes ⇒ $(1E)_{16}$ ← Length of object code

Ex:- 10 10000 FIRST STL RETADDR 141033 (3 byte each) } 10 words

⋮

55 101B LDA THREE 001020

Text record →

T^001000^1E^141033^ . . . . . ^00102D

↳ marker for separation

55 101B LDA THREE 001030

Text record →

T^001000^1E^141033^ . . . . . ^00102D

↳ marker for separation

c) End Record :

End record marks the end of the object program and specifies the address in the program where execution is to begin. If no operand is specified then the address of the first executable instruction is used.

col 1	E
col 2-7	Address of first Executable instruction in object program (hexadecimal)

Ex: 10 1000 FIRST STL RETADR 141033

⋮

255 END FIRST

End record → E^001000.

Let us start for the given program in Fig. 2.1

Given opcodes

STL - 14 J - 3C LDX - 04 JLT - 38 F - 56

LDCH - 50 TD - E0

Let us start for the given program in Fig. 2.1

Given opcodes

STL - 14	J - 3C	LDX - 04	JLT - 38	F - 56
JSUB - 48	STA - 0C	TD - E0	LDCH - 50	
LDA - 00	STX - 10	RD - D8	WD - DC	
COMP - 28	LDL - 08	STCH - 54	E - 45	
JEQ - 30	RSUB - 4C	TIX - 2C	0 - 4F	

① start incrementing LOCCTR
Initially it is 1000.
→ start adding 3 byte each time from line no. 5 to 105
→ 105 1039 BUFFER RESB 4096
↳ convert to Hexadecimal i.e $(4096)_{16} = 1000$
∴ add 1000 byte to 1039 = 2039
∴ line no. 125 starts at 2039th address continue till line no. 185.
→ 185 205D INPUT BYTE X 'F1' F1 (1 byte)
∴ add only 1 byte to 205D ⇒ 205E at line no. 190.
→ 190 205E MAXLEN WORD 4096 001000
↳ word of 3 byte not 1000 byte
∴ 3 byte added to 205E ⇒ 2061 at line no. 210
→ 210 2061 WRREC LDX ZERO 041030.
continue the same till end.

→ 255 207A END FIRST

③ object code for each line. 9

↳ Every line is object addressing except line no 160 and 225

	Line	LocCTR	label	opcode	operand	object code
(i)	5	1000	COPY	START	1000	
	10	1000	FIRST	STL	RETADR	14 1033
	⋮					
	95	1033	RETADR	RESW	1	

mnemonic code

③ object code for each line.

↳ Every line is direct addressing except line no 160 and 225

	Line	Loc CTR	label	opcode	operand	object code
(i)	5	1000	COPY	START	1000	
	10	1000	FIRST	STL	RETADR	14 1033
	⋮					
	95	1033	RETADR	RESW	1	

mnemonic code

(ii) 75 1027 RSUB HC0000

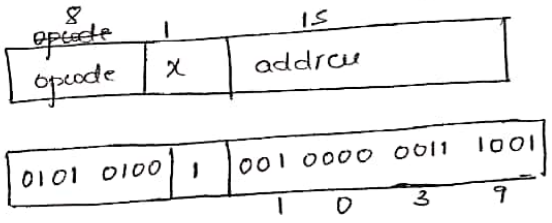
no operand.

mnemonic code

(iii) 160 204E STCH BUFFER, X ↳ indicate indexed addressing

STCH → 54

BUFFER, X → address of buffer = 1039



5 4 9 0 3 9

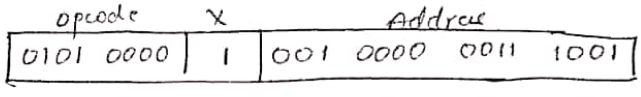
∴ 160 204E STCH BUFFER, X 549039

(iv) Same for line number 225

225 206A LDCH BUFFER, X

LDCH → 50

BUFFER, X → 1039



5 0 9 0 3 9

∴ 225 206A LDCH BUFFER, X 509039

(iv) Same for line number 225

225 206A LDCH BUFFER, X

LDCH → 50 BUFFER, X → 1039

opcode	x	Address
0101 0000	1	001 0000 0011 1001
5 0		9 0 3 9

∴ 225 206A LDCH BUFFER, X 509039

(b) Object program for Fig 3.2

H^COPY. ^001000^001077

T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^···^00102D

T^00101E^15^0C1036^482061^081033^4C0000^4F4F4F^000003^000000

T^002039^1E^0E1030^001030^E02050^302037^D82050^281030^302057^···^382031F

T^002057^1C^101036^4C0000^F1^001000^0E1036^E02079^302064^···^2C1036

T^002073^07^382064^4C0000^05

E^001000.

(c)

SYMBOL TABLE

Symbol name	value of the symbol
FIRST	1000
CLOOP	1003
ENDFIL	1015
EOF	102A
THREE	102D
ZERO	1030
RETADR	1033
LENGTH	1036
BUFFER	1039
RDREC	2039
RLOOP	203F

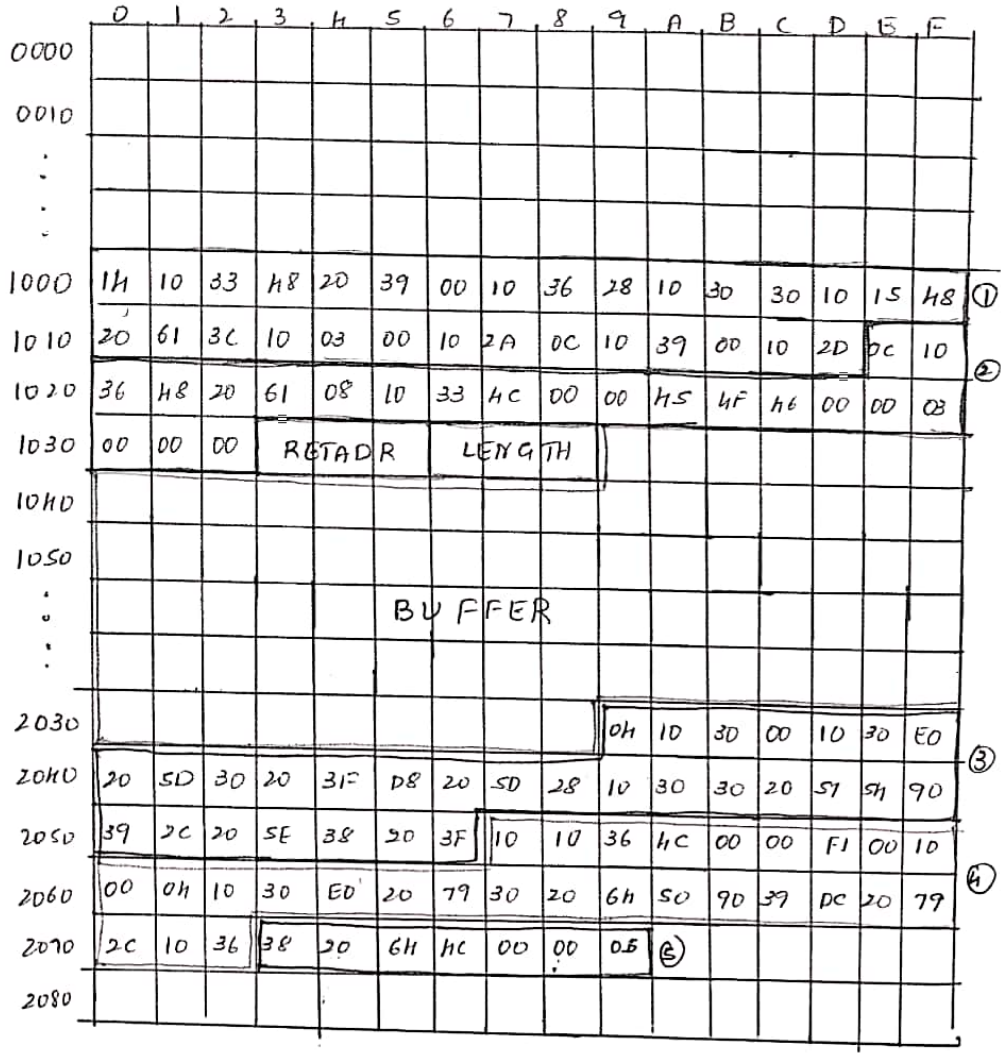
Symbol Name	Value
EXIT	2057
INPUT	2050
MAXLEN	205E
WRREC	2061
WLOOP	2064
OUTPUT	2079

10

loader loads into main memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000																	
0010																	
⋮																	
1000	1H	10	33	H8	20	39	00	10	36	28	10	30	30	10	15	H8	①
1010	20	61	3C	10	03	00	10	2A	0C	10	39	00	10	2D	0C	10	②
1020	36	H8	20	61	08	10	33	4C	00	00	H5	4F	H6	00	00	03	
1030	00	00	00	RETADR			LENGTH										

loader loads into main memory



Functions of Pass-I and Pass-II

Pass 1 :

- → Assign addresses to all statements in the program
- → Save the values (addresses) assigned to all labels for use in Pass 2
- → Perform some processing of assembler directives (includes processing that affects address assignment, the length of data areas

Functions of Pass-I and Pass-II

Pass 1 :

- → Assign addresses to all statements in the program
- → Save the values (addresses) assigned to all labels for use in Pass 2
- → Perform some processing of assembler directives (includes processing that affects address assignment, such as determining the length of data areas defined by BYTE, RESW etc)

Pass 2 :

- → Assemble instructions (Translating operation codes and looking up addresses)
- → Generate data values defined by BYTE, WORD etc
- → Perform processing of assembler directives not done during pass-1
- → Write the object program and the assembly listing.

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
```

Pass 1:

```
begin
  read first input line
  if OPCODE = 'START' then
     begin
        save #[OPERAND] as starting address
        initialize LOCCTR to starting address
        write line to intermediate file
        read next input line
     end {if START}
  else
     initialize LOCCTR to 0
  while OPCODE ≠ 'END' do
     begin
        if this is not a comment line then
           begin
              if there is a symbol in the LABEL field then
                 begin
                    search SYMTAB for LABEL
                    if found then
                       set error flag (duplicate symbol)
                    else
                       insert (LABEL,LOCCTR) into SYMTAB
                 end {if symbol}
              search OPTAB for OPCODE
              if found then
                 add 3 {instruction length} to LOCCTR
              else if OPCODE = 'WORD' then
                 add 3 to LOCCTR
              else if OPCODE = 'RESW' then
                 add 3 * #[OPERAND] to LOCCTR
              else if OPCODE = 'RESB' then
                 add #[OPERAND] to LOCCTR
              else if OPCODE  = 'BYTE' then
                 begin
                    find length of constant in bytes
                    add length to LOCCTR
                 end {if BYTE}
              else
                 set error flag (invalid operation code)
           end {if not a comment}
        write line to intermediate file
        read next input line
     end {while not END}
  write last line to intermediate file
  save (LOCCTR - starting address) as program length
end {Pass 1}
```

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

Pass 2:

```
begin
  read first input line {from intermediate file}
  if OPCODE = 'START' then
     begin
       write listing line
       read next input line
     end {if START}
  write Header record to object program
  initialize first Text record
  while OPCODE ≠ 'END' do
     begin
       if this is not a comment line then
          begin
            search OPTAB for OPCODE
            if found then
               begin
                  if there is a symbol in OPERAND field then
                     begin
                       search SYMTAB for OPERAND
                       if found then
                          store symbol value as operand address
                       else
                          begin
                            store 0 as operand address
                            set error flag (undefined symbol)
                          end
                     end {if symbol}
                  else
                     store 0 as operand address
                  assemble the object code instruction
               end {if opcode found}
            else if OPCODE = 'BYTE' or 'WORD' then
               convert constant to object code
            if object code will not fit into the current Text record then
               begin
                  write Text record to object program
                  initialize new Text record
               end
            add object code to Text record
          end {if not comment}
       write listing line
       read next input line
     end {while not END}
  write last Text record to object program
  write End record to object program
  write last listing line
end {Pass 2}
```

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

12

2.3 Machine Dependent Assembler Features

- Here we consider an example of SIC/XE machine.

→ As we know already, SIC/XE has

a) Registers : A X L B S T F PC SW
(0 1 2 3 4 5 6 8 9)

b) Dataformats : Integer : 3 byte
Character : 1 byte
Float : 6 byte

MACHINE DEPENDENT ASSEMBLER FEATURES (SIC/XE)

2.2 Machine Dependent Assembler Features

• Here we consider an example of SIC/XE machine.

→ As we know already, SIC/XE has

a) Registers : A X L B S T F PC SW
(0 1 2 3 4 5 6 8 9)

b) Dataformats : Integer : 3 byte
Character : 1 byte
Float : 6 byte

c) Instruction Formats :

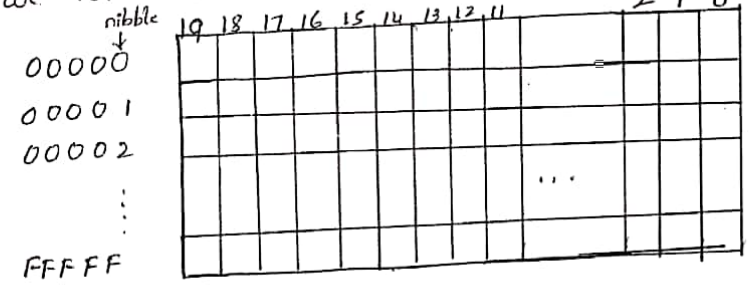
Format 1 : 1 byte — 8: opcode — Ex: FLOAT, FIX

Format 2 : 2 byte — 8: opcode | 4: r1 | 4: rx — Ex: ADDR A,X

Format 3 : 3 byte — 6: op | 1: n | 1: i | 1: x | 1: b | 1: p | 1: e | 12: disp — Ex: STL RETADR

Format 4 : 4 byte — 6: op | 1: n | 1: i | 1: x | 1: b | 1: p | 1: e | 20: address — Ex: +JSUB RDREC

→ we have 20 address lines ∴ we can have 2^{20} addresses



d) Addressing modes are determined based on 6 bits

n i x b p e

(i) →

n	i	x	Addressing mode
1	0		Indirect addressing
0	1		Immediate
1	1		not immediate, not indirect
0	0	↓	Simple addressing
		1	Indexed addressing
		0	Direct addressing

**

d) Addressing modes are determined based on 6 bits

n i x b p e

(i) →

n	i	x	Addressing mode
1	0		Indirect addressing
0	1		Immediate
** 1	1		not immediate, not indirect
0	0	↓	Simple addressing
		1	Indexed addressing
		0	Direct addressing

(ii)

b	p	e	Addressing mode
0	1		Program Counter Relative
1	0		Base relative
** 1	1		Invalid (can't be set)
0	0	↓	NO PC relative, no base relative
		1	Format 4 instruction
		0	Format 3 instructions

Different addressing mode notations

1) Indirect Addressing : @
2) Immediate Addressing : #
3) Extended Format : +
4) Indexed Addressing : operand, X
5) character string : C' '
6) Base - Register : BASE
7) Current value of PC : *

→ The addressing priority are as follows

a) PC relative addressing : $-2048 \le disp \le 2047$
(FFFFF800 ≤ disp ≤ 7FF)

b) Base relative addressing : $0 \le disp \le 4095$
(0 ≤ disp ≤ FFF)

c) Extended Instruction Format :

Note: Negative numbers are represented in 2's compliment

Procedure to create object code for SIC/XE program

→ The addressing priority are as follows

a) PC relative addressing : $-2048 \le disp \le 2047$
($FFFFF800 \le disp \le 7FF$)

b) Base relative addressing : $0 \le disp \le 4095$
($0 \le disp \le FFF$)

c) Extended Instruction Format :

Note: Negative numbers are represented in 2's compliment.

Procedure to create object code for SIC/XE program

1) Write the LOCCTR addresses for each instruction in the program.
   → if operand field is
      (i) memory address → Format 3 ⇒ Add 3 byte
      (ii) Register - Register → Format 2 ⇒ Add 2 byte
      (iii) + before operand → Format 4 ⇒ Add 4 byte
   → if it is RESW 2000
      • 2000 X 3 byte $= (6000)_d = (1770)_H$ ⇒ Add these many byte to previous address.
      • Multiplication by 3 ∵ each word is 3 byte
   → RESW 1 ⇒ Add just 3 byte
   → RESB 2000 ⇒ Add 2000 byte in hexadecimal
      ie $(2000)_d = (7D0)_H$

→ RESB 4096
   • $(4096)_d = (1000)_H$ ⇒ Add 1000 byte
→ BYTE C 'EOF' ⇒ Count the length of constant and add those many byte
→ Enter the labels onto SYMTAB (pass 1)

2) Once we are done with LOCCTR calculation and then finding program length = End Address − start address

→ RESB 4096

$(4096)_d = (1000)_H$ ⇒ Add 1000 bytes

→ BYTE C 'EOF' ⇒ Count the length of constant and add those many bytes

→ Enter the labels onto SYMTAB (pass 1)

2) Once LOCs are done with LOCCTR calculation and then finding program length = EndAddress − startaddress

3) Now start creating the object code (Pass 2) based on different addressing mode and set corresponding bits and calculate displacement

→ For extended format, displacement = address

→ For Reg-to-Reg instruction, write the opcode address followed by register numbers.

Ex: CLEAR X ⇒ B410 (Format 2)

(1) → number of X register in the list

→ For PC relative, disp = TA − PC

→ For Base relative, disp = TA − (B)

Line	Source statement			
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

Line	Source statement			
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	.			
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	.			
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	

Figure 2.5 Example of a SIC/XE program.

Line	Loc		Source statement			Object code
5	0000		COPY	START	0	
10	0000	3	FIRST	STL	RETADR	17202D
12	0003	3		LDB	#LENGTH	69202D
13				BASE	LENGTH	
15	0006	4	CLOOP	+JSUB	RDREC	4B101036
20	000A	3		LDA	LENGTH	032026
25	000D	3		COMP	#0	290000
30	0010	3		JEQ	ENDFIL	332007
35	0013	4		+JSUB	WRREC	4B10105D
40	0017	3		J	CLOOP	3F2FEC
45	001A	3	ENDFIL	LDA	EOF	032010
50	001D	3		STA	BUFFER	0F2016
55	0020	3		LDA	#3	010003
60	0023	3		STA	LENGTH	0F200D
65	0026	4		+JSUB	WRREC	4B10105D
70	002A	3		J	@RETADR	3E2003
80	002D	3	EOF	BYTE	C'EOF'	454F46
95	0030	3	RETADR	RESW	1	
100	0033	3	LENGTH	RESW	1	
105	0036	1000	BUFFER	RESB	4096	
110			.			
115			.	SUBROUTINE TO READ RECORD INTO BUFFER		
120			.			
125	1036	2	RDREC	CLEAR	X	B410
130	1038	2		CLEAR	A	B400
132	103A	2		CLEAR	S	B440
133	103C	4		+LDT	#4096	75101000
135	1040	3	RLOOP	TD	INPUT	E32019
140	1043	3		JEQ	RLOOP	332FFA
145	1046	3		RD	INPUT	DB2013
150	1049	2		COMPR	A,S	A004
155	104B	3		JEQ	EXIT	332008
160	104E	3		STCH	BUFFER,X	57C003
165	1051	2		TIXR	T	B850
170	1053	3		JLT	RLOOP	3B2FEA
175	1056	3	EXIT	STX	LENGTH	134000
180	1059	3		RSUB		4F0000
185	105C	1	INPUT	BYTE	X'F1'	F1
195			.			
200			.	SUBROUTINE TO WRITE RECORD FROM BUFFER		
205			.			
210	105D	2	WRREC	CLEAR	X	B410
212	105F	3		LDT	LENGTH	774000
215	1062	3	WLOOP	TD	OUTPUT	E32011
220	1065	3		JEQ	WLOOP	332FFA
225	1068	3		LDCH	BUFFER,X	53C003
230	106B	3		WD	OUTPUT	DF2008
235	106E	2		TIXR	T	B850
240	1070	3		JLT	WLOOP	3B2FEF
245	1073	3		RSUB		4F0000
250	1076	1	OUTPUT	BYTE	X'05'	05
255	1077			END	FIRST	

Figure 2.6 Program from Fig. 2.5 with object code.

Consider the example of figure 2.5

1) Add the length of each instruction and add it to LOCCTR and find the program length

Program length = End start address − start address

= 1077 − 0000 = 1077

pass 1

2) Create the symbol table

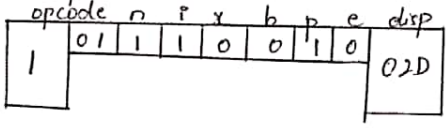
Symbol name	PC value
FIRST	0000
CLOOP	0006
ENDFIL	001A
EOF	002D
RETADR	0030
LENGTH	0033
BUFFER	0036
RDREC	1036
RLOOP	1040
EXIT	1056
INPUT	105C
WRREC	105D
WLOOP	1062
OUTPUT	1076

WRREC	105D
WLOOP	1062
OUTPUT	1076

3) start creating object code (pass-2)

10 0000 FIRST STL RETADR (Format 3 ∵ oprd is memory address)
1H 0030

By default assembler uses PC relative addressing

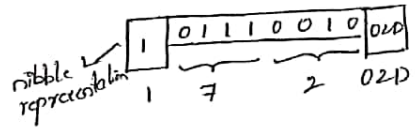


→ TA = PC + disp
↳ Displacement = TA − PC
= RETADR − LOCCTR (location of next instn to be executed)
= 0030 − 0003 = 002D
∵ format 3 displacement is 12 bits

↳ 02D is within range of $-2048 \le disp \le 2047$
↳ opcode is 6 bits (4+2 ⇒ 1 nibble + 2 bits)
⇒ last 2 bits can be represented by 4 bits but always last 2 bits are "zero"

Ex:- 4 ⇒ 01 00
only 2 bits
C ⇒ 11 00

↳ not immediate, not indirect so set n=1, i=1
↳ not indexed x=0, not base relative b=0 but it is pc relative p=1, not format 4 e=0
↳ write the whole instruction's object code ie



∴ STL RETADR 17202D

12 0003 LDB #LENGTH
68 0033

→ opcode for LDB = 68

∴ STL RETADR 172020

12 0003 LDB #LENGTH
68 0033

→ opcode for LDB = 68

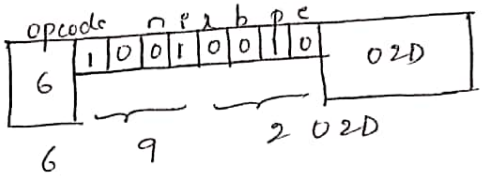
→ # is imm calculate disp.

TA = PC + disp

disp = TA − PC = 0033 − 0006 = 02D

→ Pc relative ∵ operand is memory address

→ it is immediate so i=1



LDB #LENGTH ⇒ 69202D

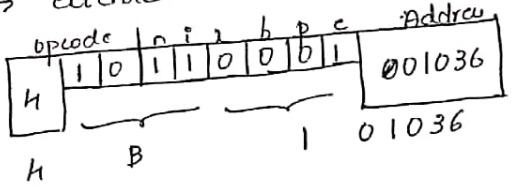
15 0006 CLOOP +JSUB RDREC → Format 4
48 0030

→ disp = (operand) ∵ it is extended format (F4

= 001036 (20 bits)

→ not immediate, not indirect n=1, i=1.

→ extended e=1



CLOOP +JSUB RDREC ⇒ 4B101036

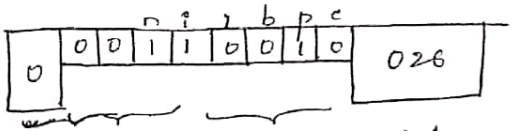
20 000A LDA LENGTH → Format 3
00 0033

→ Pc relative, disp = TA − PC

P=1 = 0033 − 000D = 026

→ not immediate, not indirect, not indexed so

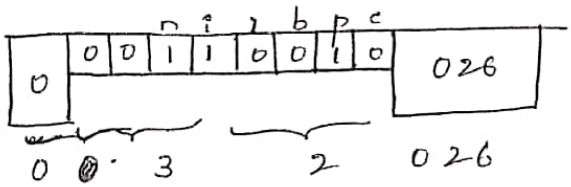
n=1, i=1, x=0



20 000A LDA LENGTH → Format 3
00 0033

→ PC relative, disp = TA − PC
P=1 = 0033 − 000D = 026

→ not immediate, not indirect, not indexed so
n=1, i=1, x=0

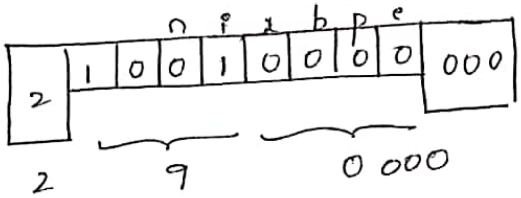


LDA LENGTH ⇒ 032026

25 000D COMP #0
28

→ ~~immediate~~ not PC relative because operand is direct value but not memory address.
∴ displacement = operand = 000

→ immediate addressing n=0, i=1, b=0, p=0



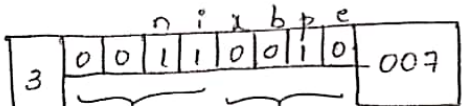
COMP #0 ⇒ 290000

---

30 0010 JEQ EXDFIL ⇒ Format 3
30 001A

• PC relative ∴ disp = TA − PC
= 001A − 0013 = 007
within range

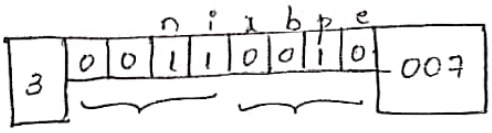
• not immediate, not indirect n=1, i=1



30 0010 JEQ EXDFIL ⇒ Format 3
(JEQ = 30, EXDFIL = 001A)

• Pc relative ∴ disp = TA − PC
= 001A − 0013 = 007
within range

• not immediate, not indirect n=1, i=1

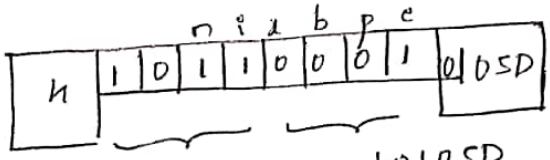


3 3 2 007

∴ JEQ EXD FIL ⇒ 332007

35/65 0013 +JSUB WRREC ⇒ Format 4
(JSUB = 48, WRREC = 105D)

• displacement = address of operand
= 0105D
~~within~~



4 B 1 0105D

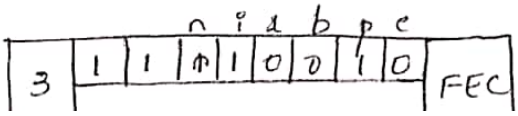
+JSUB WRREC ⇒ 4B1010SD

40 0017 J CLOOP ⇒ Format 3
(J = 3C, CLOOP = 0006)

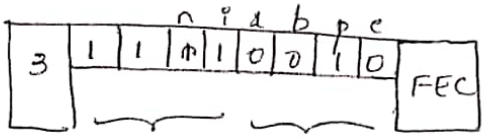
disp = TA − PC = 0006 − 001A
= −14 (it take 2's complement)
= FEC

• it is Pc relative P=1

• not immediate, not indirect n=0, i=1



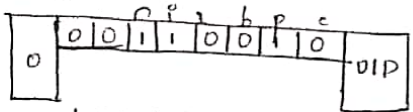
- it is Pc relative p=1
- not immediate, not indirect n=0, i=1



objeed code : 3 F 2FEC

45 001A EOFAL LDA EOF ⇒ Format 3 (Pc relative)
00 001D

disp = TA − PC = 002D − 001D = 0010

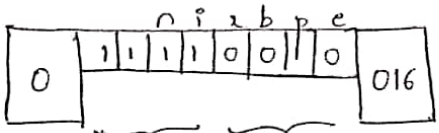


0 3 2 010

LDA EOF ⇒ 032010

50 001D STA BUFFER ⇒ Format 3 (Pc relative)
0C 0036

disp = TA − PC = 0036 − 0020 = 0016

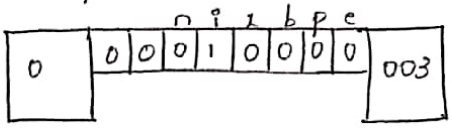


0 F 2016

∴ STA BUFFER ⇒ 0F2016

55 0020 LDA #3 ⇒ immediate address

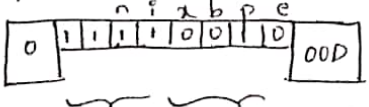
disp = 003



LDA #3 ⇒ 010003

60 0023 STA LENGTH ⇒ Format 3 (Pc relative)
0C 0033

disp = TA − PC = 0033 − 0026 = 00D

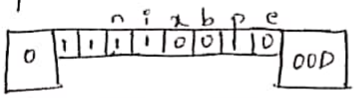


0 F 200D ⇒ 0F200D

— RSTADR ⇒ Format-3 - Indirect

60 0023 STA LENGTH ⇒ Format 3 (pc relative)
0C 0033

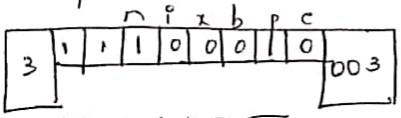
disp = TA - PC = 0033 - 0026 = #00D



0 F 200D ⇒ 0F200D

70 002A J @RETADR ⇒ Formal-3 - Indirect
3C 0030

disp = TA - PC = 0030 - 002A = #006



3 E 2 003 ⇒ 3E2003

80 002D EOF BYTE C 'EOF'

⇒ Convert EOF to heiodcuimal ascii value

E — 45
O - 4F
F — 46

125 1036 RDREC CLEAR X A X L B S T F PC SW
BH 0 1 2 3 4 5 6 8 9

⇒ BH10 ↳ this is not accumulator

⇒ only 2 bytes since it is register-to-register mode

130 1038 CLEAR A ⇒ BH00
Bh

132 103A CLEAR S ⇒ BH40
Bh

150 10A9 COMPR A,S ⇒ A004
A0

165 10S1 TIXR T ⇒ B850
B8

235 106E TIXR T ⇒ B850
B8

130 — B4

132 1039 CLEAR S ⇒ B410
(CLEAR: B4)

150 1049 COMPR A,S ⇒ A004
(COMPR: A0)

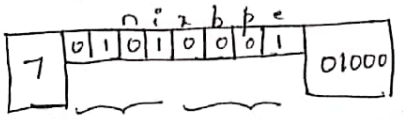
165 1051 TIXR T ⇒ B850
(TIXR: B8)

235 106E TIXR T ⇒ B850
(TIXR: B8)

210 1050 LDRREG CLEAR X ⇒ B410
(CLEAR: B4)

133 103C +LDT #4096 ⇒ Format 4 Immediate addressing

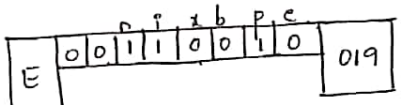
disp = $(4096)_{16}$ = 01000



7 5 101000 ⇒ 75101000

135 1040 RLOOP TD INPUT ⇒ Format 3 + PC relative
(TD: E0) (INPUT: 105C)

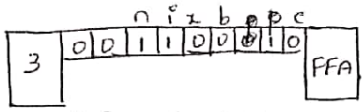
disp = TA − PC = 105C − 1043 = 019



E 3 2019 ⇒ E32019

140 1043 JEQ RLOOP ⇒ Format 3 + PC relative
(JEQ: 30) (RLOOP: 1040)

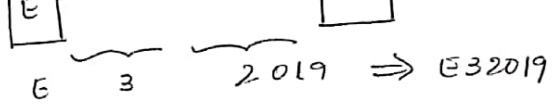
disp = TA − PC = 1040 − 1046 = −6



2's complement ⇓ FFA

3 3 2 FFA ⇒ 332FFA

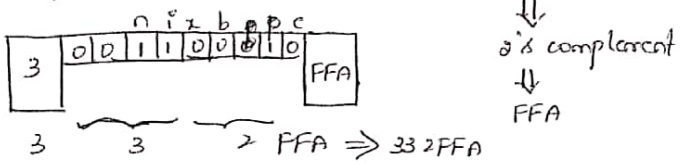
1CC 104B JEQ EXIT ⇒ Format 3 + PC relative



1H0 1046 JEQ RLOOP ⇒ Format 3 + Pc relative

JEQ → 30 RLOOP → 1040

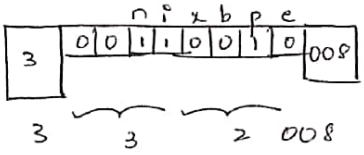
disp = TA − PC = 1040 − 1046 = −6



155 104B JEQ EXIT ⇒ Format 3 + Pc relative

JEQ → 30 EXIT → 1056

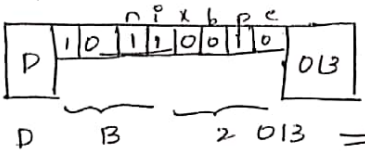
disp = TA − PC = 1056 − 104E = 008



1H5 1046 RD INPUT ⇒ Format 3 + Pc relative

RD → D8 INPUT → 105C

disp = TA − PC = 105C − 1049 = 013



160 104E STCH BUFFER, X ⇒ Indexed + Pc relative

** STCH → 54 BUFFER, X → 0036

disp = TA − PC = 0036 − 1051 = −101B

= EFE5 (H123)$_{10}$ > 2047

160 1041 STCH BUFFER, X ⇒ Indexed + pc relative

** STCH → 54, BUFFER → 0036

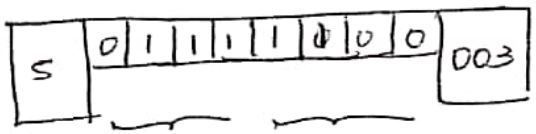
disp = TA − PC = 0036 − 1051 = −101B

= EFEF (H123)$_{10}$ > 2047

∴ it is not pc relative, go for base relative

disp = TA − B = BUFFER − B (length is stored in base register at 0033)

= 0036 − 0033 = 0003

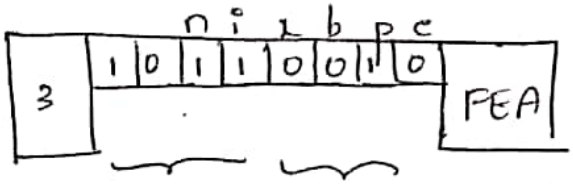


5 7 C 003 ⇒ 57C003

170 1053 JLT RLOOP ⇒ Format-3 + pc relative

JLT → 38, RLOOP → 1040

disp = TA − PC = 1040 − 1056 = FEA



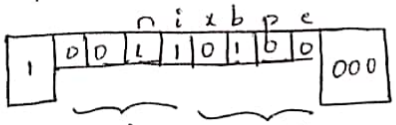
3 B 2 FEA ⇒ 3B2FEA

175 1056 EXIT STX LENGTH ⇒ Format 3 + PC relative
** 10 0033

disp = TA − PC = 0033 − 1059 = EFDA > 2047

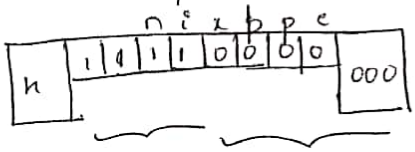
∴ go for base relative mode

disp = TA − (B) = 0033 − 0033 = 0000



1 3 4000 ⇒ 134000

180 1059 RSUB ⇒ Format 3
4C
no displacement ∵ no operand.



4 F 0000 ⇒ 4F0000

185 105C INPUT BYTE X'F1'
⇒ character string ∴ store as it is

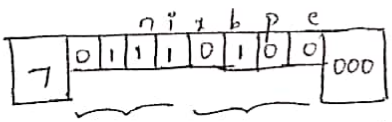
912 105F LDT LENGTH ⇒ Format 3
** 74 0033

disp = TA − PC = 0033 − 1062 = EFD1
4443 > 2047

∴ go for base relative

---

disp = TA − (B) = 0033 − 0033 = 0000

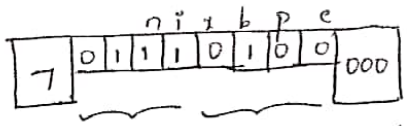


7 7 4000 ⇒ 774000

915 1062 WLOOP TD OUTPUT ⇒ Format 3 + PC relative
E0 1076

disp = TA − PC = 1076 − 1065 = 011

disp = TA − (B) = 0033 − 0033 = 0000

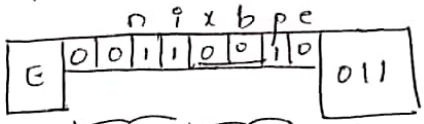


7 7 4000 ⇒ 774000

915 1062 WLOOP TD OUTPUT ⇒ Format 3 + PC relati

TD → E0 OUTPUT → 1076

disp = TA − PC = 1076 − 1065 = 011

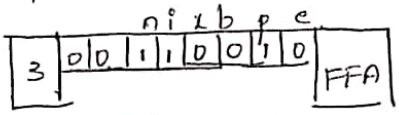


E 3 2011 ⇒ E32011

920 1065 JEQ LOOP ⇒ Format 3 + PC relati

JEQ → 30 LOOP → 1062

disp = TA − PC = 1062 − 1068 = FFA



3 3 2FFA ⇒ 332FFA

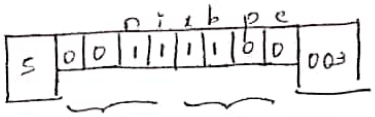
925 1068 LDCH BUFFER, X ⇒ indexed

LDCH → 50 BUFFER → 0036

disp = TA − PC = 0036 − 106B = −1035 > 2047

∴ go for base relative mode

disp = TA − B = 0036 − 0033 = 0003

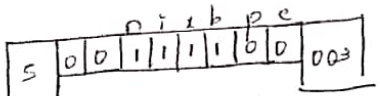


5 3 C003 ⇒ 53C003

930 106B LOD OUTPUT ⇒ F3 + PC relative

LOD → PC OUTPUT → 1076

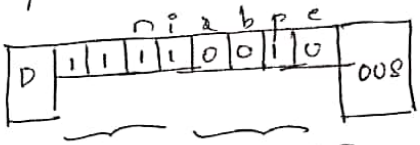
disp = TA − PC = 1076 − 106E = 0008



5 3 C 003 ⇒ 53C003

230 · 106B LOD OUTPUT ⇒ F3 + PC relative
PC 1076

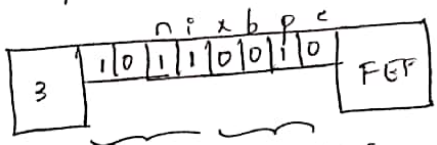
disp = TA - PC = 1076 - 106E = 0008



D F 2008 ⇒ DF2008

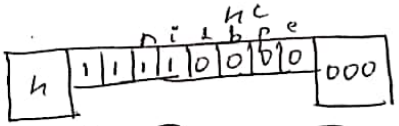
240 1070 JLT LOOP ⇒ Format 3 + PC relative
38 1062

disp = TA - PC = 1062 - 1073 = FEF



3 B 2 FEF ⇒ 3B2FEF

245 1073 RSUB ⇒ Format 3
HC



4 F 0000 ⇒ 4F0000

250 1076 OUTPUT BYTE X '05'

⇒ Character string ∴ store as it is ⇒ 05

4) object program

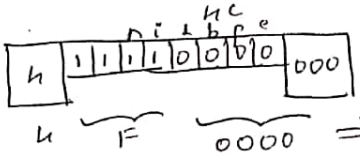
H^COPY ^000000^001077

T^000000^1D^172D2D^69202D^4B1D1036^032026^290000^332007^4B1010SD^
3F2FEC^032010

T^00001D^13^0F2016^010003^0F202D^4B1010SD^3E2003^454F46

T^001036^1D^B410^1300^B440^751010000^E32019^332FFA^DB2013^A004^
332FFA^DB2013^A004^332008^57C003^B850

2H5 1073 RSUB ⇒ Format 3



4 F 0000 ⇒ 4F0000

250 1076 OUTPUT BYTE X '05'

⇒ Characte string ∴ store as it is ⇒ 05

---

4) object program

H^COPY ^000000^001077

T^000000^1D^172D2D^69202D^4B1D1036^032026^290000^332007^4B1015D^3F2FEC^032010

T^00001D^13^0F2016^010003^0F202D^4B101D5D^3E2003^454F46

T^001036^1D^B410^1300^B400^751010000^E32019^332FFA^DB2013^A004^332FFA^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

E^000000

Loading into memory

Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	17	20	2D	69	20	2D	4B	10	10	36	03	20	26	29	00	00	①
0010	33	20	07	4B	10	10	5D	3F	2F	EC	03	20	10	0F	20	16	
0020	01	00	03	0F	20	0D	4B	10	10	5D	3E	20	03	45	4F	46	③
0030	RETADR			LENGTH													
							BUFFER										
1030							Bh	10	Bh	00	Bh	A0	75	10	10	00	
10H0	E3	20	19	33	2F	FA	DB	20	13	A0	0A	33	20	08	57	C0	③
1050	03	B8	50	3B	2F	EA	13	40	00	4F	00	00	F1	B4	10	77	
1060	40	00	E3	20	11	33	2F	FA	53	C0	03	DF	20	08	B8	50	④
1070	3B	2F	EF	4F	00	00	05	⑤									

22

. Generate the complete object program for the

1. Generate the complete object program for the following assembly level program.

CLEAR - B4, LDA - 00, LDB - 68, ADD - 18, TIX - 8C,

JLT - 38 STA - 0C

PASS-I	LENGTH	LABEL	OPCODE	OPERAND	PASS-II
0000		SUM	START	0	
0000	2		CLEAR	X	B410
0002	3		LDA	#0	010000
0005	4		+LDB	#TOTAL	69101789
			BASE	TOTAL	
0009	3	LOOP	ADD	TABLE,X	1BA00D
000C	3		TIX	COUNT	2F2007
000F	3		JLT	LOOP	3F2FF7
0012	4		+STA	TOTAL	0F101789
0016	3	COUNT	RESW	1	
0019	1770	TABLE	RESW	2000 $(1770)_H$	
1789	3	TOTAL	RESW	1	
178C			END	FIRST	

RESW 2000 $\Rightarrow$ $2000 \times 3 = (6000)_{dbytes} = (1770)_H$

$\therefore$ 0019 + 1770 = 1789

Program Length = 178C - 0000 = 178C

---

1) 0000 CLEAR X (Register-to-Register)

$\Rightarrow$ directly opcode with register numbers

$\Rightarrow$ B410

2) 0003 LDA #0 $\Rightarrow$ Immediate Addressing

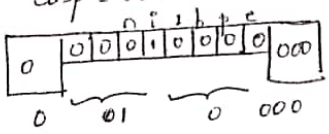
2) 0000 CLEAR X (Register - to - Register)

⇒ directly opcode with register numbers

⇒ B410

3) 0002 LDA #0 ⇒ Immediate Addressing

disp = 000

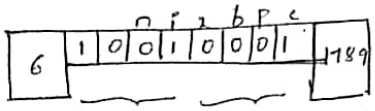


0 01 0 000 ⇒ 010000

3) 0005 +LDB #TOTAL ⇒ Extended & Immediate – Format 4 + immediate with PC relative

disp = operand address

= 1789



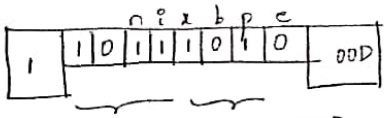
6 9 1 01789 ⇒ 69101789

4) 0009 LOOP ADD TABLE,X ⇒ indexed with PC relative

TA = PC + disp

disp = TA – PC

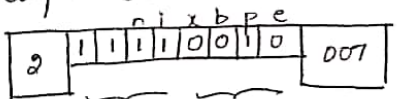
= 0019 – 000C = 00D



1 B A 00D ⇒ 1BA00D

5) 000C TIX COUNT ⇒ Format-3

disp = 0016 – 000F = 007



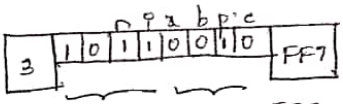
2 F 2 007 ⇒ 2F2007

6) 000F JLT LOOP ⇒ Format-3 PC relative

disp = TA – PC

= 0009 – 0012 = FF7

within range ⇒ –2048 ≤ FF7 ≤ 2047



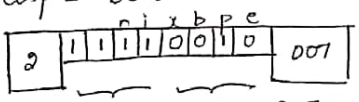
3 B 2 FF7 ⇒ 3F2FF7

7) 0012 +STA TOTAL ⇒ Format 4

disp = operand value = 1789

5) 000C TIX COUNT ⇒ Format 3

disp = 0016 − 000F = 007



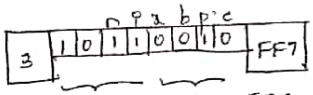
2 F 2 007 ⇒ 2F2007

6) 000F JLT LOOP ⇒ Format-3 PC relative

disp = TA − PC

= 0009 − 0012 = FF7

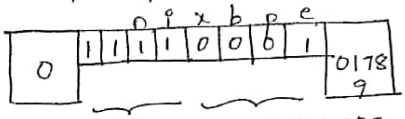
Within range ⇒ −2048 ≤ FF7 ≤ 2047



3 B 2 FF7 ⇒ 3F2FF7

7) 0012 +STA TOTAL ⇒ Format 4

disp = operand value = 1789



0 F 1 01789 ⇒ 0F101789

Object program (Pass-2)

H^ Sum ^0000. ^00178C

T^ 000000^16^BH1D^010000^691017897^1BA0DD^2F2007^3F2FF7^0F101789

E^ 000000

SYMTAB (pass-1) →

SYMBOL NAME	VALUE
LOOP	0009
COUNT	0016
TABLE	0019
TOTAL	1789

9. Generate the complete object program for the following assembly level program. Also indicate the content of symbol table at the end. Assume standard SIC model and assume the following opcodes in HEX

LDA = 00 STA = 0C TIX = 2C JLT = 38

LDX = 04 ADD = 18 RSUB = 4C

2. Generate the complete object program for the following assembly level program. Also indicate the contents of symbol table at the end. Assume standard SIC model and assume the following mnemonic codes in HEX

LDA = 00 STA = 0C TIX = 2C JLT = 38

LDX = 04 ADD = 18 RSUB = 4C

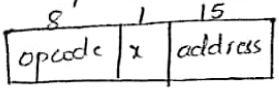
LOCCTR (PASS-1)	LENGTH	LABEL	OPCODE	OPERAND	OBJECT CODE
		SUM	START	4000	
4000	3	FIRST	LDX	ZERO	045788
4003	3		LDA	ZERO	005788
4006	3	LOOP	ADD	TABLE,X	18C015
4009	3		TIX	COUNT	2C5785
400C	3		JLT	LOOP	384006
400F	3		STA	TOTAL	0C578B
4012	3		RSUB		4C0000
4015	1770	TABLE	RESW	2000 (1770)H	
5785	3	COUNT	RESW	1	
5788	3	ZERO	WORD	0	000000
578B	3	TOTAL	RESW	1	
578E			END	FIRST	

Program length = END address - starting address

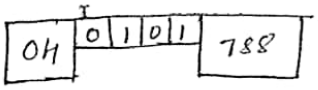
= 578E - 4000 = 178E

→ Since it is SIC program, we have two addressing mode — direct addressing (x=0), indexed addressing (x=1)

- So directly put opcode with operand address



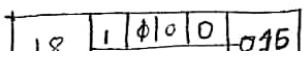
1) 4000 FIRST LDX ZERO



⇒ 045788

2) 4006 ADD TABLE,X ⇒ indexed addressing

18 (ADD) 4015 (TABLE)

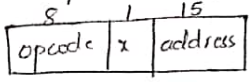


			END	FIRST

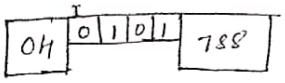
Program length = End address - starting address
= 578E - H000 = 178E

→ Since it is SIC program, we have two addressing mode < direct addressing (x=0) / indexed addressing (x=1)

- So directly put opcode with operand address

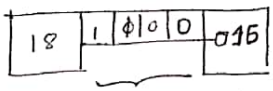


1) H000 FIRST LDX ZERO



⇒ 0h5788

2) H006 ADD TABLE, X ⇒ indexed addressing
18 H015



18 C 015 ⇒ 18C015

SYMTAB

NAME	VALUE
FIRST	H000
LOOP	H006
TABLE	H015
COUNT	5785
ZERO	5788
TOTAL	578B

object program

H^ sum ^00H000 ^00178E

T^ 00H000 ^ 15 ^ 0H5788 ^ 005788 ^ 18C015 ^ 2C5785 ^ 38 H006 ^ 0C5788 ^ H0000

T^ 005788 ^ 03 ^ 000000

E^ 00H000

LOADING INTO MAIN MEMORY

	0	1	2	3	H	5	6	7	8	9	A	B	C	D	E	F
0000																
0010																
⋮						.	.	.								
H000	0H	57	88	00	57	88	18	C0	15	2C	57	85	38	40	06	0C
H010	57	8B	HC	00	00											
H020																

object program

H^ SUM ^00^H000 ^001738

T^ 00H000^ 15^ 0H5788^ 005788^ 18C015^ 2C5785^ 38H006^ 0C5788^ H00000

T^ 005788^ 03 ^000000

E^ 00H000

LOADING INTO MAIN MEMORY

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000																
0010																
H000	0H	57	88	00	57	88	18	C0	15	2C	57	85	38	40	06	0C
H010	57	8B	HC	00	00											
H020																
						TABLE										
5780						COUNT			00	00	00	TOTAL				
5790																

3. Generate the object code for each statement in the following SIC/XE program and generate the object program for the same.

LOCCTR	LENGTH	LABEL	OPCODE	OPERAND

3. Generate the object code for each statement in the following SIC/XT program and generate the object program for the same.

LOCCTR	LENGTH	LABEL	OPCODE	OPERAND	OBJECT-CODE
		SUM	START	0	
0000	3	FIRST	LDX	#0	050000
0003	3		LDA	#0	010000
0006	4		+LDB	#TABLE2	69101790
			BASE	TABLE2	
000A	3	LOOP	ADD	TABLE, X	1BA013
000D	3		ADD	TABLE2, X	1BC000
0010	3		TIX	COUNT	2F200A
0013	3		JLT	LOOP	3B2FFA
0016	4		+STA	TOTAL	0F102F00
001A	3		RSUB		4F0000
001D	3	COUNT	RESW	1	
0020	1770	TABLE	RESW	2000 (1770)/3	
1790	1770	TABLE2	RESW	2000 (1770)/3	
2F00	3	TOTAL	RESW	1	
2F03		END	FIRST		

LDX = 04 LDB = 68 TIX = 2C STA = 0C
LDA = 00 ADD = 18 JLT = 38 RSUB = 4C

25

→ keep assigning the length for each instruction based on

(i) 1st operand is memory address — 3 byte
(ii) 1st operand is register — 2 byte
(iii) + (Extended format — 4 byte

→ Find the LOCCTR value; Program length = 2F03 − 0000 = 2F03

→ Create SYMTAB

Name	Value
FIRST	0000

→ keep assigning the length for each instruction based on

(i) 1st operand is memory address — 3 byte
(ii) 1st operand is register — 2 byte
(iv) + (Extended format — 4 byte

→ Find the LOCCTR value ; Program length = 2F03 − 0000 = 2F03

→ Create SYMTAB

Name	Value
FIRST	0000
LOOP	000A
COUNT	001D
TABLE	0020
TABLE2	1790
TOTAL	2F00

→ object code for each instruction

1) 0000 FIRST LDX #0 → immediate addressing
(LDX = 04)

disp = 000

n	i	x	b	p	e	
0	1	0	1	0	0	000

0 0 S 0 000 ⇒ 050000

2) 0003 LDA #0 → immediate addressing
(LDA = 00)

disp = 000

n	i	x	b	p	e	
0	1	0	0	0	0	000

0 0 1 0000 ⇒ 010000

3) 0006 +LDB #TABLE2 → Extended + immediate
(LDB = 68)

~~TA = PC + disp~~
~~disp = TA − PC = 1790 − 000A~~
disp = Target address = 01790 (20 bit)

n	i	x	b	p	e	
1	0	0	1	0	1	01790

6 9 101790 ⇒ 69101790

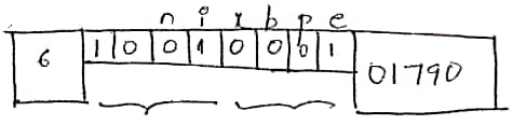
4) 000A LOOP ADD TABLE, X → indexed + PC relative
(ADD = 18)

3) 0006 +LDB (68) #TABLE2 → Extended + immediate

~~TA = PC + disp~~

~~disp = TA = PC = 1790 = 000A~~

disp = Target address = 01790 (20bit)

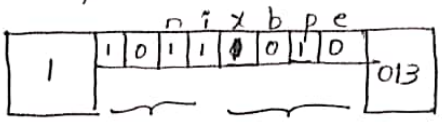


6 9 101790 ⇒ 69101790

4) 000A LOOP ADD (18) TABLE, X → indexed + PC relative

TA = PC + disp

disp = TA − PC = 0020 − 000D = Ø013



1 B A 013 ⇒ 1BA013

5) 000D ADD (18) TABLE2, X → indexed + base relative

(∵ TABLE2 is stored in base register)

Initially we can try for PC-relative & checkout whether displacement is within the range.

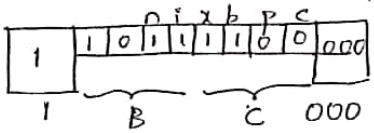
disp = TA − PC

= 1790 − 0010 = $(1790)_H \neq (6016)_d > (2047)_d$

∴ go for base relative

disp = TA − B (look for address of TABLE2 in SYMTAB)

= 1790 − 1790 = Ø000

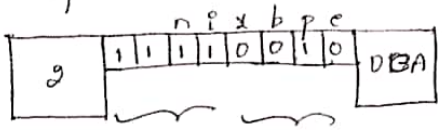


1 B C 000 ⇒ 1BC000

---

6) 0010 TIX (2C) COUNT → PC relative

disp = TA − PC = 001D − 0013 = Ø00A



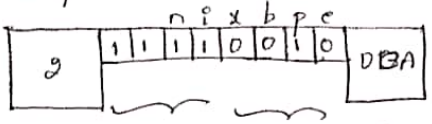
2 F 2 00A ⇒ 2F200A

7) 0013 JLT (38) LOOP ⇒ PC relative

disp = TA − PC = 000A − 0016 = FFF4

6) 0010 TIX COUNT → Pc relative
2C

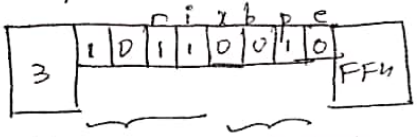
disp : TA − PC = 001D − 0013 = Ø00A



2 F 2 00A ⇒ 2F200A

7) 0013 JLT LOOP ⇒ Pc relative
3B

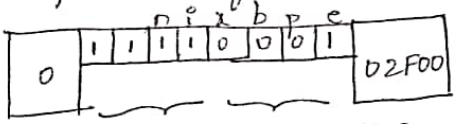
disp = TA − PC = 000A − 0016 = FFh



3 B 2FFh ⇒ 3B2FFh

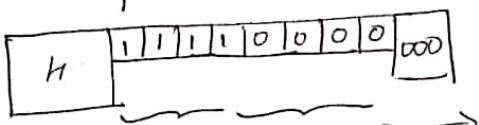
8) 0016 +STA TOTAL ⇒ Extended (Format 4)
0C

disp : address of TOTAL = 2F00



0 F 102F00

9) RSUB 001A RSUB
⇒ no operand ∴ no displacement



4 F 0000 ⇒ 4F0000

→ Object program

H^ SUM ^000000^002F03

T^000000^1D^050000^010000^69101790^1BA013^1BC000^2F200A^3B2FFh
^0F102F00^4F0000

E^000000

→ loader loads into memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	05	00	00	01	00	00	69	10	17	90	1B	A0	13	1B	C0	D0
0010	2F	20	0A	3B	2F	FF	0F	10	2F	00	4F	00	00	COUNT		

→ Object program

H^ SUM ^000000^002F03

T^000000^1D^050000^010000^69101790^1BA013^1BC000^2F200A^3B2FFh
^0F1021F00^hF0000

E^000000

→ loader loads into memory

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	05	00	00	01	00	00	69	10	17	90	1B	A0	13	1B	C0	00
0010	2F	20	0A	3B	2F	Fh	0F	10	2F	00	HF	00	00	COUNT		
002D																
⋮						TABLE										
1790																
						TABLE 2										
2F00	TOTAL															

Program Relocation

- Sometimes it is required to load and run several programs whenever there is place in the memory.

Absolute Program:

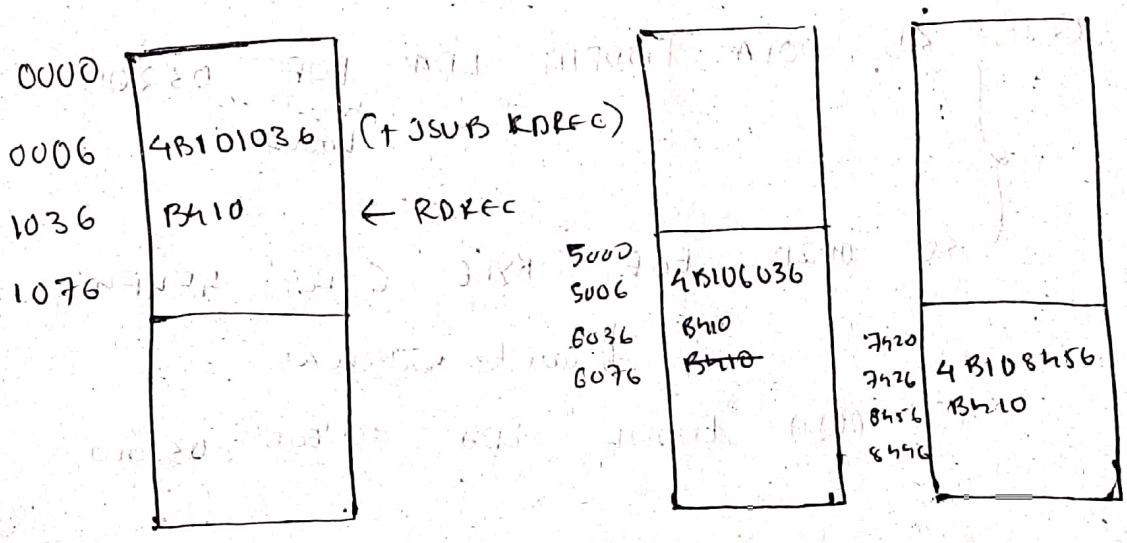
Consider the instruction:

55 101B LDA THREE 001020

-> This statement says that the register A is loaded with the value stored at location 1020.

-> Suppose, it is decided to load and execute the program at location 2000 instead of location 1000.

-> Then at address 1020 the required value which needs to be loaded in the register A is no more available.



Machine Independent Features

There are the features which do not depend on the architecture of the machine. There are:

- Literals
- Symbol-Defining Statements
- Expressions
- Program Blocks
- Control sections

① Literals

→ Constant operand can be specified as a part of the instruction that uses it instead of using a label which is defined as constant elsewhere. Such an operand is called a literal because the value is stated "literally" in the instruction.

```
60:-  45   001A   ENDFIL   LDA   EOF      032010
                                  LABEL
      80   002D   EOF      BYTE  C'EOF'    454F46
```

⇓ can be written as

```
      45   001A   ENDFIL   LDA   =C'EOF'   032010
```

② Symbol - Defining Statements

Most assemblers provide an assembler directive that allows the programmers to define symbols and specify their values.

The assembler directives are a) EQU b) ORG

a) EQU (Equate)

- It is used to equate the value to variables

Ex: +LDT #4096; load the value 4096 into reg T

↓ replace with

```
MAXLEN  EQU  4096
        +LDT  MAXLEN
```

→ When assembler encounters the EQU statement, it enters MAXLEN to SYMTAB with value 4096

- It is used to define mnemonic names for registers

Ex:
```
A  EQU 0
X  EQU 1
L  EQU 2
```

- To reflect the logical function of the registers

Ex: BASE EQU R1
COUNT EQU R2
:

5) ORG (Origin)

→ Assembler directive used to indirectly assign values to symbols

→ Syntax:

ORG value

→ When ORG is encountered, the assembler changes its LOCCTR to the specified value.

→ We can access the label entries in two ways (usage of EQU and ORG)

using EQU => Symbol EQU STAB
VALUE EQU STAB +6
FLAGS EQU STAB +9

offset from STAB

i) To fetch the value field,

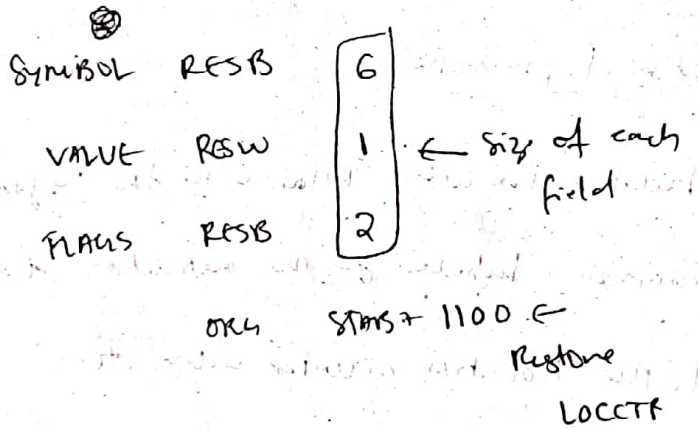
LDA VALUE, X ; where X=0,1,2,
↳ index register
for each entry

-> This method of definition simply defines the labels, it does not make the structure of the table as clear as it might be

-> Therefore, we make use of ORG.

Using ORG => STAB RESB 1100

ORG STAB ← Set LOCCTR to STAB



6) Expressions

-> Assemblers also allow use of expressions in place of operands in the instructions

-> Assemblers generally arithmetic expressions formed according to the normal rules using arithmetic operators : +, -, *, /.

-> The only spl symbol used is * which indicates the value of next unassigned memory location.

Expressions are classified as

(i) Absolute Expressions
(ii) Relative Expressions

} based on type of value produced

Absolute Expressions

→ Absolute means independent of program location and contains absolute terms like constants.

Relative Expressions

→ Relative means relative to the beginning of the program, such as labels on the instruction, data area, references to the location counter value etc.

eg:- MAXLEN EQU BUFEND - BUFFER

STAB RESB (6+3+1)*MAX

Here, the terms could be absolute or relative.

Program Blocks and Control Sections

→ Sometimes it is required to logically rearrange the statements of the source program so that the

1) Large buffer area can be moved to the end of object pgm

2) No need of using extended instruction format

3) The base register usage is not required

4) The problem of placing literals in program has to be more flexible.

→ All this can be achieved through some assembler features such as program blocks and control sections.

Assembler Directive : USE

Syntax : USE Blockname

eg:-

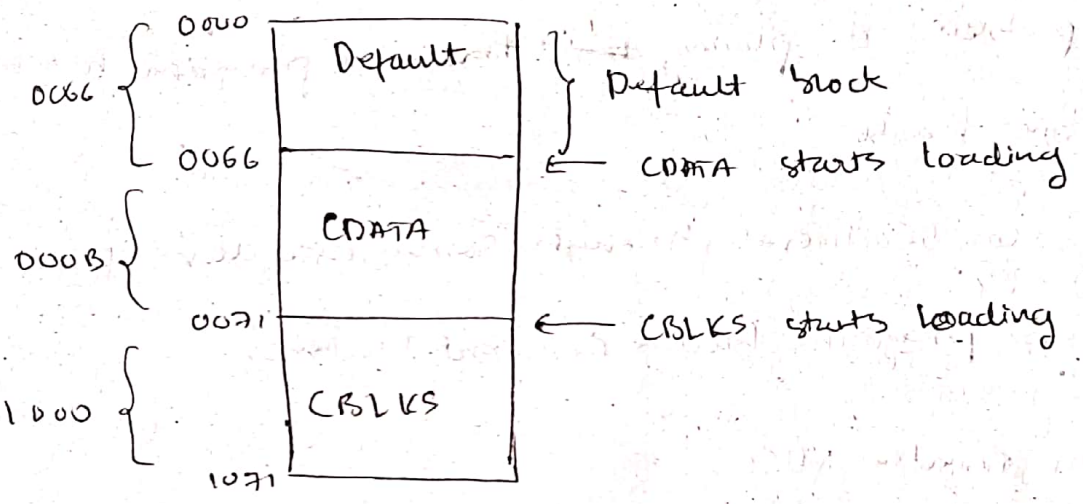
20	0006	0	LDA	LENGTH	
:					
92	0000	1	USE	CDATA	
100	0003	1	LENGTH	RESW	1

There are three blocks in the program

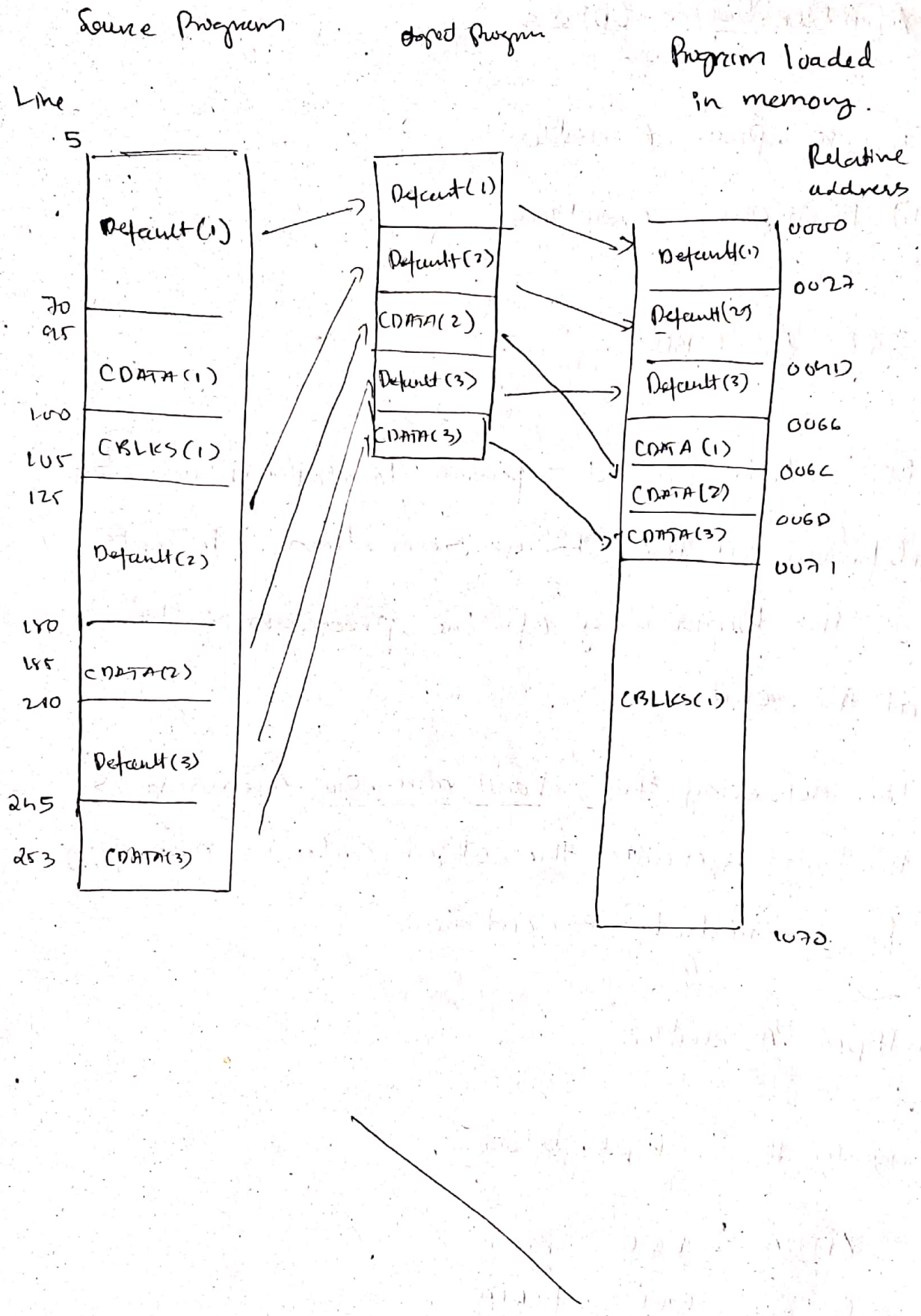
(1) Unnamed program block contains the executable instructions of the program

(ii) CDATA program block contains all data areas that consists of few blocks of memory i.e. few words or len in length

(iii) CBLKS program block contains all data areas that contains of larger blocks of memory.



- A control section is a part of the program that maintains its identity after assembly; each such control section can be loaded and relocated independently of the others.



Assembler Design Options

a) One pass Assemblers

b) Multipass Assemblers

ONE PASS ASSEMBLER:

-> Eliminating forward reference to data items, by defining all the storage reservation statements at the beginning of the program rather at the end.

-> For achieving this, load-and-Go Assembler is used, it generates the object code in memory for immediate execution.

Multipass Assembler

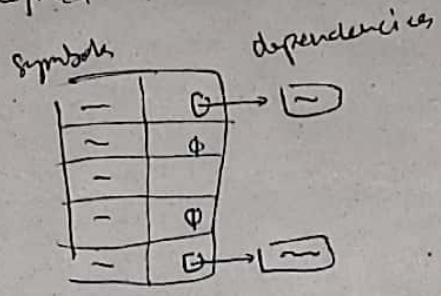
-> Consider the example below:

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

As we see above, we have multiple find references that cannot be resolved in two-pass

→ For this the assembler makes as many passes as needed for all undefined symbols.

→ It is not necessary to go through the entire program, but only through a part of it where symbols are undefined



Basic Loader Functions

- A loader is a system program that performs the loading function.

```
HCOPY  001000001007A
T001000 1E 14103348 20390010 36281030 30101548 20613C10 0300102A 0C103900 102D0C10
T001010 1E 150C1036 48206108 10334C00 00454F46 00000300 0000
T002039 1E 041030 001030 E0205D30 203FD8205D 28103030 20575490 392C205E 38203F
T002057 1C 101036 4C0000 F1001000 41030E02 079302064 509039 DC20792C1036
T002073 07 3820644C 000005
E001000
```

(a) Object program

Memory address	Contents				
0000	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
0010	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	
0FF0	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
1000	14103348	20390010	36281030	30101548	
1010	20613C10	0300102A	0C103900	102D0C10	
1020	36482061	0810334C	0000454F	46000003	
1030	000000xx	xxxxxxxx	xxxxxxxx	xxxxxxxx	← COPY
⋮	⋮	⋮	⋮	⋮	
2030	xxxxxxxx	xxxxxxxx	xx041030	001030E0	
2040	205D3020	3FD8205D	28103030	20575490	
2050	392C205E	38203F10	10364C00	00F10010	
2060	00041030	E0207930	20645090	39DC2079	
2070	2C103638	20644C00	0005xxxx	xxxxxxxx	
2080	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	
⋮	⋮	⋮	⋮	⋮	

(b) Program loaded in memory

Figure 3.1 Loading of an absolute program.

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

Figure 3.2 Algorithm for an absolute loader.

A Simple Bootstrap Loader

-> When a computer is first turned on or restarted, a special type of absolute loader, called a bootstrap loader, is executed.

-> The bootstrap itself begins at address 0

-> It loads the operating system starting at address 80

-> The object code from device F1 is always loaded into consecutive bytes of memory, starting at address 80.

-> After all of the object code from device F1 has been loaded, the bootstrap jumps to address 80, which begins execution of the program that was loaded.

-> Much of the work of the bootstrap loader is performed by the subroutine GETC.

-> It reads each character and converts it into ASCII value to hexadecimal digit represented by character.

-> For example, ASCII code for the character "0" (hexadecimal 30) is converted to 0 and same goes for other characters as well.

-> GETC is used to read and convert a pair of characters from device F1.

-> These two hexadecimal digit values are combined into a single byte => "1" and "4" are written as "14".

-> Along with these, STCH and TIXR are used.

```
BOOT     START    0        BOOTSTRAP LOADER FOR SIC/XE
.
. THIS BOOTSTRAP READS OBJECT CODE FROM DEVICE F1 AND ENTERS IT
. INTO MEMORY STARTING AT ADDRESS 80 (HEXADECIMAL). AFTER ALL OF
. THE CODE FROM DEVF1 HAS BEEN SEEN ENTERED INTO MEMORY, THE
. BOOTSTRAP EXECUTES A JUMP TO ADDRESS 80 TO BEGIN EXECUTION OF
. THE PROGRAM JUST LOADED.  REGISTER X CONTAINS THE NEXT ADDRESS
. TO BE LOADED.
.
         CLEAR    A        CLEAR REGISTER A TO ZERO
         LDX      #128     INITIALIZE REGISTER X TO HEX 80
LOOP     JSUB     GETC     READ HEX DIGIT FROM PROGRAM BEING LOADED
         RMO      A,S      SAVE IN REGISTER S
         SHIFTL   S,4      MOVE TO HIGH-ORDER 4 BITS OF BYTE
         JSUB     GETC     GET NEXT HEX DIGIT
         ADDR     S,A      COMBINE DIGITS TO FORM ONE BYTE
         STCH     0,X      STORE AT ADDRESS IN REGISTER X
         TIXR     X,X      ADD 1 TO MEMORY ADDRESS BEING LOADED
         J        LOOP     LOOP UNTIL END OF INPUT IS REACHED
.
. SUBROUTINE TO READ ONE CHARACTER FROM INPUT DEVICE AND
. CONVERT IT FROM ASCII CODE TO HEXADECIMAL DIGIT VALUE. THE
. CONVERTED DIGIT VALUE IS RETURNED IN REGISTER A. WHEN AN
. END-OF-FILE IS READ, CONTROL IS TRANSFERRED TO THE STARTING
. ADDRESS (HEX 80).
.
GETC     TD       INPUT    TEST INPUT DEVICE
         JEQ      GETC     LOOP UNTIL READY
         RD       INPUT    READ CHARACTER
         COMP     #4       IF CHARACTER IS HEX 04 (END OF FILE),
         JEQ      80           JUMP TO START OF PROGRAM JUST LOADED
         COMP     #48      COMPARE TO HEX 30 (CHARACTER '0')
         JLT      GETC     SKIP CHARACTERS LESS THAN '0'
         SUB      #48      SUBTRACT HEX 30 FROM ASCII CODE
         COMP     #10      IF RESULT IS LESS THAN 10, CONVERSION IS
         JLT      RETURN       COMPLETE. OTHERWISE, SUBTRACT 7 MORE
         SUB      #7           (FOR HEX DIGITS 'A' THROUGH 'F')
RETURN   RSUB             RETURN TO CALLER
INPUT    BYTE     X'F1'    CODE FOR INPUT DEVICE
         END      LOOP
```

Figure 3.3 Bootstrap loader for SIC/XE.