INTRODUCTION TO
Machine Architecture

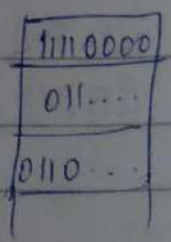
SIC →
- Simplified Instructional Computer
- This microprocessor is used in Systems
- 8 bit microprocessor.
- Hypothetical machine.

→ Architecture of SIC m/c
→ Architecture of SIC/XE m/c
→ Sample programs of SIC m/c
} contents of ch-1.

* Architecture of a m/c :-
- refers to Internal design of m/c
- memory bits supported by m/c
- registers supported by m/c
- data formats " " "
- Instruction format allowed by m/c
- Addressing modes supported by m/c
- Instruction set " " "
- Facilities to perform I/O

* ARCHITECTURE OF SIC MACHINE :-

- SIC supports 15 address bits.
- The no. of possible addresses are 2^{15}
- All locations are byte addressable i.e. every location stores 1 byte of data.



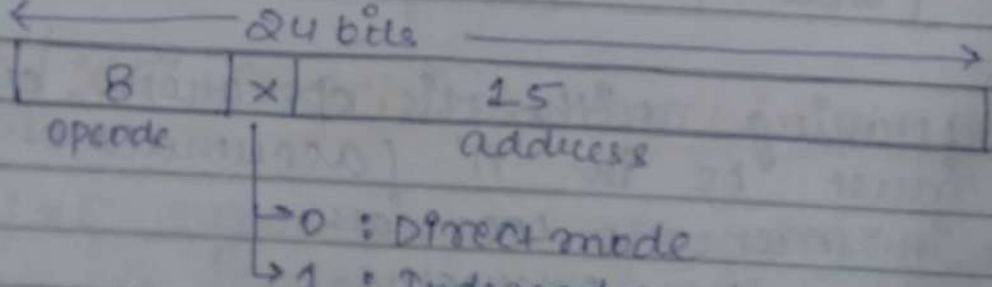
• In SIC machine, word size = 3 bytes

∴ To store a WORD, three consecutive locations are considered.

• It supports 5 registers :- A, X, L, PC, SW
All registers are ~~32~~ 24 bit or 1 WORD.
A - Accumulator
X - Index register, used to access array elements.
L - Linkage register, used to store ret add while sub-routine.
PC - Program counter, store add of next instruction.
SW - Status word, used to store flag operation.

• Data Formats : SIC only supports
Integer (24 bits)
Character (8 bits)

• Instruction Formats : < Direct / Indexed



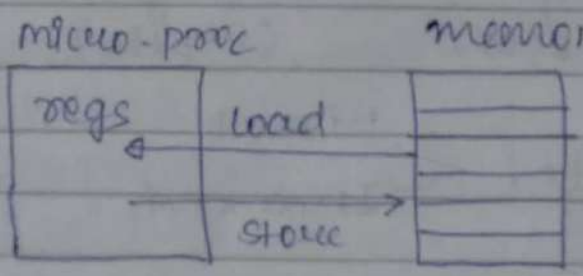
Address calculation depends on X :-
• If X=0, address is taken & then the control goes to that location & fetches data.
• If X=1, add ← [x] + add.

- If x=0, address is taken as effective add.
- If x=1, effective add is calculated as
effective add ← [x] + address.

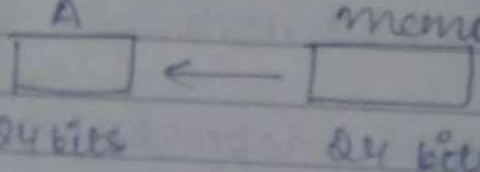
- Addressing modes:
  - Direct (address)
  - Indexed (x + address)

The purpose of addressing modes is how we access the data.

- Instructions:
  - LDA // loading
  - STA // storing
  - ADD, MUL, DIV, SUB // Arithmetic operations

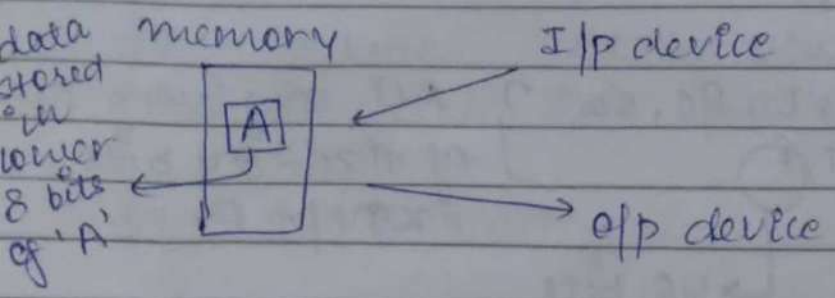


- For performing arithmetic operations, one of the data must be in 'A' (accumulator), other can be in memory location.



• I/O :

3 Instructions :-



- TD (Test device)
- RD (Read from i/p device)
- WD (Write to o/p device)

- TD → used to check whether i/p device or o/p device is ready or not.

- RD → used to perform reading from i/p device.

- WD → used to send data to o/p device.

Using these instructions, only 1 byte of data is transfered.
First, the data is transfered to accumulator 'A' but 'A' is of 24 bits, so data is stored in lower 8 bits of A.

* SIC/XE Machine Architecture :

- SIC/XE supports no. of add bits 20.
  ∴ Possible add = 2^{20} = 1 M byte.

- It is an improved version of SIC/XE.

- It supports several inst. formats & several addr modes as compared to SIC.

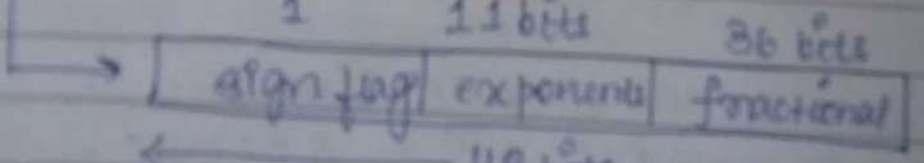
- Registers: { A, X, L, PC, SW, B, S, T, F } All registers are of size 24 bits except F.

F → 48 bits

A : accumulator
X : Index reg , addr[X]
L : Linkage reg
Pc : program counter , to point to next inst.
Sw : status word
B : Base register
S : } general purpose
T : } working registers
F : Floating point register

- Data Formats:

- Integer ⟶ 24 bits 1 WORD = 24 bits
- Floating Pt. ⟶ 48 bits
- Characters ⟶ 8 bits



$f \times 2^{(e-1024)}$

to convert into actual form

→ normalized form i.e. 0.0002 → 0.
x 10

$= \frac{0.2}{10} \times \frac{1}{1000}$

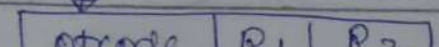
$= 0.0002$

• **Instruction format:-** 4 inst. formats are supported.

f1 - 1 byte // only opcode & no operands
f2 - 2 byte // the operation using 1 byte, operands 4-4 bits.
f3 - 3 byte
f4 - 4 byte

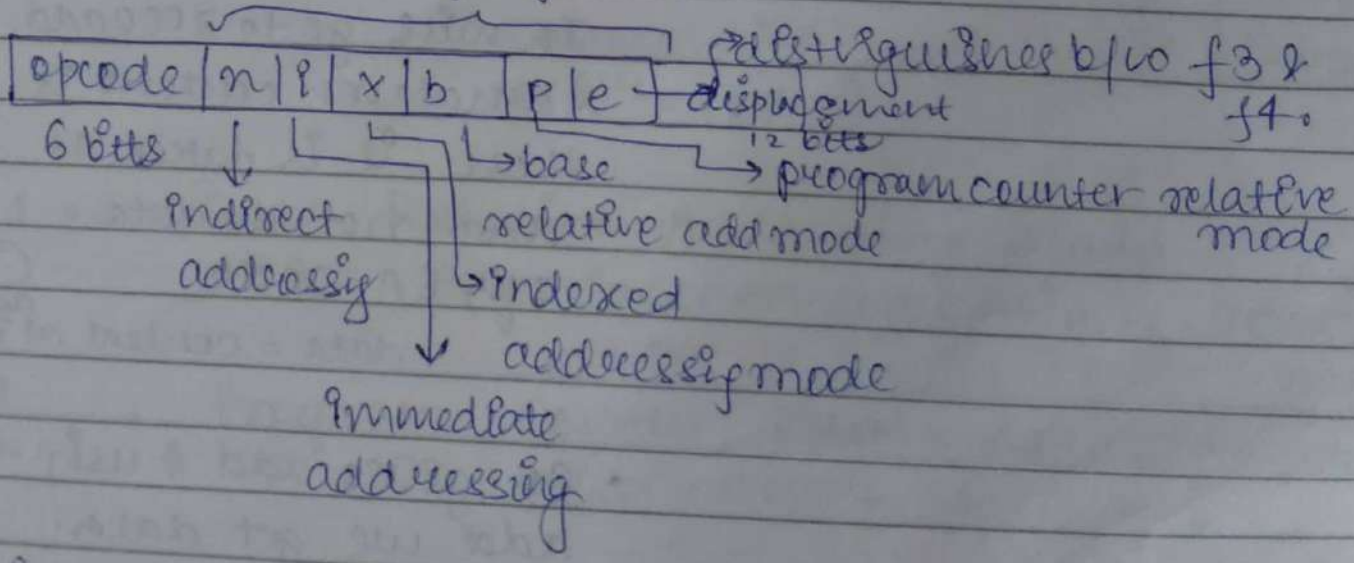
E.g f1: RSUB // Return from subroutine
Code for RSUB is 50. 0110 0000 — 1 byte.
// do not refer memory

f2: ADDR A,S // They do not refer memory.

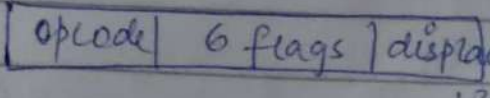


// only registers are used

f3: 3 bytes



e-bit 0: f3
1: f4



f4:

opcode	n	i	x	b	p	e	20 bits

6 bits — memory address

Addressing Modes: depends on various flags n, i, x, b, p, e

f3, f4

opcode 6 bits	n i x b p e	displacement : 12 → f3 mem. address : 20 → f4

n = 0 , i = 1 **Immediate Addressing Mode**

E.g $a_1 = a_2 + (45)$ → immediate

n = 1 , i = 0 **Indirect Addressing Mode**

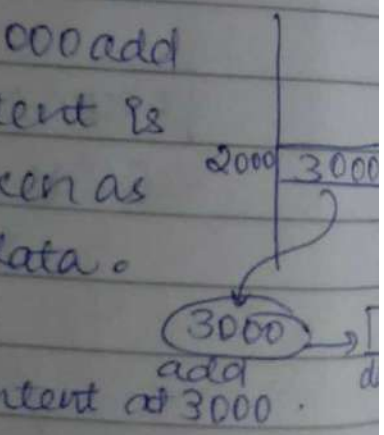
- we compute Effective Address (EA)

Let E.A = 2000

It will go to 2000 add & whatever content is there, it is taken as add. not the data.

If, EA = 2000

∴ data = content of 3000.



- EA gives add & using that add we get data.

- Similar to pointer concept in C.

e = 0 format 3

e = 1 format 4

In case of format 4, whatever address is there we take it as the effective add.

In case of format 3, Effective Add is calculated using displacement.

To Compute EA :-

If x = 1, EA = [X] + displacement
then we refer to the EA to get data.

We refer an address to get 1 WORD.
Let add = 1000
∴ data = 563412
(3 consecutive locat^{n}'s (bytes)

1000	12	1byte
	34	1byte
	56	1byte

If b = 1, Base Relative Addressing Mode
EA = B + displacement → {0 ... 4095}
$2^{12} = 4096$

P = 1, Program Counter Relative Addressing mode
EA = PC + displacement → $2^{12} = 4096$
-2048 +2047

~~SS lab~~

02/01/2021
~~29/12/2020~~

Synchronous Class [01]

* System Software: consists of a variety of programs that support the operation of a computer

E.g: OS, Assemblers, Linkers, Loaders, Compilers,

- Assembly to machine code conversion → Assemblers

★ SIC:

- 32K bit memory
- 05 registers of each 24 bits.
- Data format ~~is~~ are of 02 types:
  integer (24 bits) Character (8 bits).
- 02 addressing modes.

~~DIRECT~~ ADDRESSING MODE EXAMPLE

1). LDA TEN
2). STCH BUFFER, X

8	1	15
opcode	x	Address

Given LDA = 00h, STCH = 54H, TEN = 1000
Buffer = 1000

(i)

opcode	x	Address
00000000	0	0010 0000 0000 0000

↳ ∴ Direct AM

Direct
LDA is DAM

~~(ii)~~ Now, group them in bits of 4 from left.

∴ 0000,0000,0010,0000,0000,0000,
∴ Machine value is: 002000

What is the machine value that we get?

(ii) STCH Buffer, X

STCH = 54h

Buffer = 1000

→ Indexed Add mode (∵ STCH).

01010100	1	0010 0000 0000 0000

Machine value : 549000

{ Address line should be in range
0000 —— 7FFF }

★ SIC/XE :

- 2^{20} memory.
- 4 Inst format.
- 6 addressing modes.

Ques : Solve the following examples :-

<i> LDA #3

Given LDA = 00H

(Immediate addressing mode, ∵ #3)

format 3 or format 4.

↳ If Inst is appended with '+' at beginning.

∴ format 3 ✓

→ 12 bits

opcode(6)	n	i	x	b	p	e	displacement
0000 00	0	1	0	0	0	0	0000 0000 0011

- Immediate value will be written in displacement

∴ Machine Value : 010003 h

If it is format 4 → put n=1 i=1, x=0, b=0, p=0 neither addressed nor indexed

(ii) +JSUB RDREC JSUB = 4B

↳ format 4 instruct^{n}: RDREC = 1036

∴ Address = 20 bits

6 bits							20 bits
opcode	n	i	x	b	p	e	memory
0100 ~~000100~~ 10	1	1	0	0	0	1	0000 0001 0000 0011 0110

↓

first 6 bits of opcode i.e. 4B → 010010 11

1036 →

0000 append ∵ we need 20 bits.

0001 0000 0011 0110

∴ it becomes 01036

∴ Machine Code : 4B101036

<Y STX LENGTH

//Base Relative Addressing mode.

//format 3.

Given:- EA = 0033, [B] = 0033

EA = [B] + Disp

Also has STX = 10

Here, ∵ EA = [B]

∴ disp = 00h.

opcode	n	i	x	b	p	e	disp
000100	10	1	0	1	0	0	0000 0000 0000

∴ Machine code : 134000

extended
e=1

xppy STL RETADR Given: RETADR=0030, PC=0003
// PC relative add mode. We know
// format 3. EA = [PC] + Disp
STL = 14

Here, EA = RETADR
∴ Disp = 2D → 00101101

* SIC/XE Inst & addressing modes:

Hex → 032600

0000 0011 0010 0110 0000 0000

displacements = 600

opcode n i x b p e

∴ LDA

n: neither direct nor indexed

p: PC rel

format 3

∴ EA = [PC] + disp
∴ EA = [PC] + 600

Here, PC = 3000

∴ EA = 3000 + 600
= 3600 ← target address

∴ load into A register the contents of address 3600.
∴ A ← 103000 (given at addr 3600)

(PP) Hex → 03C300

0000 0011 1100 0011 0000 0000

opcode = LDA

n i x b p e

// format 3

// x=1

// b=1

displacement = 300

EA = [X] + [B] + displacement

Here, X = 000090

B = 006000

Content at 006390 is 00C303

∴ A ← 00C303

∴ EA = 006000 + 000090 + 300

= 006390

↑ target address

CHAPTER-02
ASSEMBLER

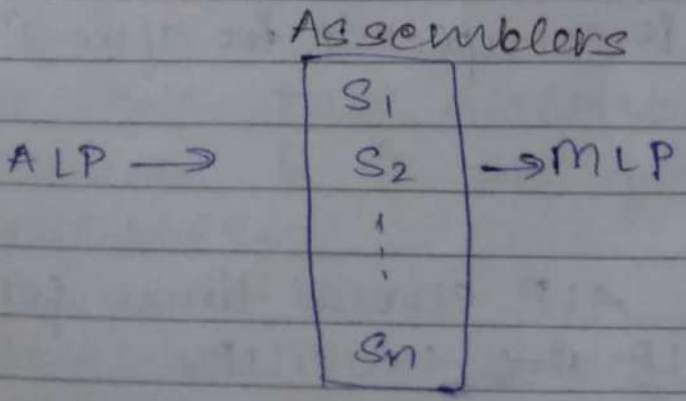
- Assembler is a system software.
- It takes an Assembly program as I/P & produces machine level codes.

ALP → [Assembler] → M.L.Code.

High level language → [Compilers] → Machine level language

Examples of System Software:
- Assemblers
- Loaders
- OS
- Compilers

★ Content:
- Machine dependent features
- Machine Independent features
- Types of Assembler



★ Machine dependent features:
Features that vary from machine to machine. E.g SIC 2 inst formats SIC/XE 4 inst formats

E.g Assembler for microprocessor 8085 may vary from that of Pentium.

* Machine Independent features: There are some features that do not depend on architecture of the machines, such are machine independent features.

* Types of Assembler:
  1). One Pass → transfer of ALP to MLP happens by scanning ALP only once.
  2). Two Pass
  3). Multi Pass

Two Pass
↳ scans ALP twice for conversion from ALP to MLP.
- During first scan, some operations are performed.
- Conversion is completed after 2nd scan.

Multi Pass
↳ goes through ALP several times for converting ALP to MLP.

Video 2: FUNCTIONS OF ASSEMBLERS

★ Need of two-pass Assembler :-

E.g
```
      S1 →
back: S2 →
      JMP next   (forward reference)
      S3 →
next: S4 →
      JMP back   (backward reference)
```

- In case of forward reference, such as JMP next first the address of 'next' is not known in the first pass.
- So, it becomes difficult or impossible for the assembler to Jump to that address in a single pass.
- So, two-pass assemblers are required. However, it is ~~ok~~ fine for the backward references.
- Thus, we need two-pass Assemblers due to forward references, as the add of the forward references are not known.
- So, most of the assembler uses 2 pass assembler for conversion of ALP to MLP.

★ Functionality of a 2 pass Assembler:

```
1000        JMP     next
1003        ADD     SUM
1006        SUB     SUM1
(1009)
next :      MUL     XYZ

1012        SUM     RESW  1
1015        SUM1    RESW  1
1018        XYZ     RESW  1
```

- 2 pass assemblers perform two passes.
- During 1st pass, addresses are assigned.
- Every instruction is replaced by the machine code.

Symbols
- Variables: {SUM, SUM1, ~~next~~ XYZ}
- Labels: {next}

- All these symbols are stored in SYMBOL TABLE during pass 1.

- SYMBOL TABLE

next	1009
SUM	1012
SUM1	1015

- It also processes assembler directive

E.g. 1012 SUM RESW 100
~~1015~~ SUM1 RESW1

1012 + (100X3)
1012 + 300
= 1312

∴ 3 major tasks in Pass 1 are :

(i) Assigning addresses
(ii) Storing symbols in Symbol table
(iii) Processing some of the assembler directives. i.e. specified amt of memory is reserved for particular variables.

- During pass 2, the statements are translated into machine code.
- Now, there is another table OPTAB. It includes every inst (opcode) & its code.

E.g.

ADD	- 70
SUB	- 99
MVI	- 77

∴ ADD SUM
↳ refers SYMBOL TABLE, gets 1012
↳ refers OPTAB, gets 70

∴ machine code 70 1012.

- Constants are also converted into binary forms in pass 2. E.g. EOF → 474849.
- It also checks whether it is valid instruction or not, by refering to OPTAB in Pass 1.

- If the opcode is present in SYMTA/OPTAB, it is valid statement, else not.

* Video 4: Object Program Generation

- In SIC machine, all instructions are represented using 24 bits hex representation.
- One hex digit represents 4 bits & we have 6 digits, 30, 24 bits.

ALP	SIC	MLP
S1	———	701234
S2	———	412567
S3	———	812354
S4	———	I24512

- 03 types of records are used:
  1). Header [H - name, starting add, length, ...]
  2). Text [T (several lines) - M/C level code]
  3). End. [E - starting add of the program]

ALP —(Assembler)→ MLP —(loader)→ Memory —(CPU)→ Execute

- M/C code is represented in the form of:

H
T
T
:
E

} Header, text, End.

06/01/2021 SS Class [02]

SIC PROGRAMMING EXAMPLES

Ques To store 5 in ALPHA & z in C1

```
LDA    FIVE     // Load constant 5 into Register A
STA    ALPHA    // STORE in ALPHA
LDCH   CHARZ    // Load character 'z' into register A
STCH   C1       // STORE in character variable C1
  :

ALPHA  RESW  1   // ONE-WORD VARIABLE
FIVE   WORD  5   // ONE-WORD CONSTANT
CHARz  BYTE  C'z' // ONE-WORD CONSTANT
C1     RESB  1   // ONE-WORD VARIABLE
```

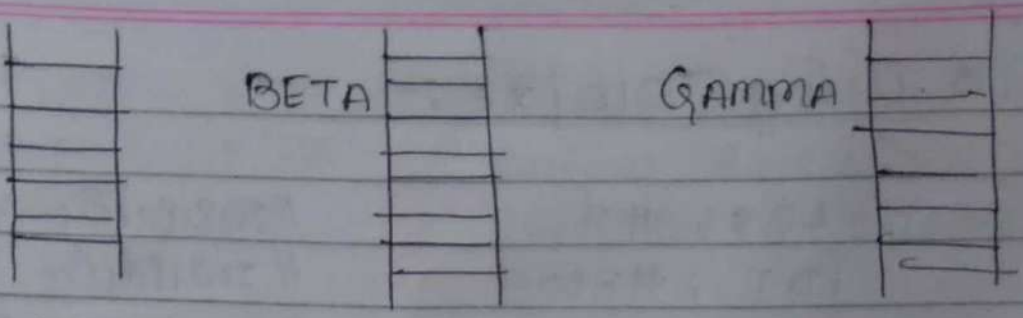
Ques Same program in SIC|XE

```
      LDA    #5      // Load value 5 into register A
      STA    ALPHA   // STORE in Alpha
      LDA    #90     // Load ASCII Code for 'z' into reg A
      STCH   C1      // STORE in char variable C1
        :
ALPHA RESW   1       // One-WORD VARIABLE
C1    RESB   1       // One-~~WORD~~ BYTE VARIABLE
```

- LDA → load into accumulator a no.
- LDCH → load into accumulator a character.
- STA → If storing in number form.
- STCH → If storing in character form.

* JLT (Jump less than) will check the given value with 'A'. If less, then jump.

Bues: ALPHA BETA GAMMA



Contents of ALPHA & BETA are summed up & results are stored in GAMMA.

Gamma[I] = Alpha[I] + Beta[I]
I = 0 to 100

* Using SIC :-

```
        LDA     ZERO
        STA     INDEX
ADDLP   LDX     INDEX       // load index value into reg. X
        LDA     ALPHA,X     // load word from Alpha into reg.
        ADD     BETA,X      // Add word from beta & store in
        STA     GAMMA,X     // Store the result in a word in Gam
        LDA     INDEX       // Add 3 to index value
        ADD     THREE
        STA     INDEX
        COMP    K300        //Compare new index value to 300
        JLT     ADDLP       //Loop if index is less than 300
        .
        .
        .

INDEX   RESW    1           ONE-WORD VARIABLES FOR INDEX VA
.                           ARRAY VARIABLES--100 WORDS E
ALPHA   RESW    100
BETA    RESW    100
GAMMA   RESW    100
.                           ONE WORD CONSTANTS
ZERO    WORD    0
K300    WORD    300
THREE   WORD    3
```

* Using SIC/XE :-

```
        LDS , #3          // Initialize reg S to 3
        LDT , #300        // Initialize reg T to 300
        LDX , #0          // Initialize index reg to 0
ADDLP   LDA , ALPHA,X     // Load word from Alpha into Reg
        ADD   BETA,X      // Add word from beta
        STA   GAMMA,X     // Store the res in a word in Gamma
        ADDR  S,X         // X ← X+5
        COMPR X,T         // Compare new index value to 300
        JLT   ADDLP       // Loop, if index values is less than
        :
        :                 ARRAY VARIABLES -- 100 words each
ALPHA   RESW  100
BETA    RESW  100
GAMMA   RESW  100
```

Ques: To read 1 byte of data from device F1 & copy it to device 05.

```
INLOOP  TD  INDEV     // Test I/p device
        JEQ INLOOP    // loop until device is ready
        RD  IDEV      // read one byte into reg A
        STCH DATA     // Store byte that was read
        :
OUTLP   TD  OUTDEV    // test o/p device
        JEQ OUTLP     // loop until device is ready
        LDCH DATA     // load data byte into register A
        WD  OUTDEV    // Write one byte to o/p device
        :
```

```
INDEV   BYTE   X'F1'   // Input device no.
OUTDEV  BYTE   X'05'   // Output device no.
DATA    RESB   @1      // ONE-BYTE variable.
```

Ques : To read 100 bytes of record from an i/p device into memory.

```
        JSUB   READ
        :
READ    LDX    ZERO
RLOOP:  TD     INDEV
        JEQ    RLOOP
        RD     INDEV
        STCH   RECORD,X
        TIX    K100
        JLT    RLOOP
        RSUB
        :

INDEV   BYTE   X'F1'
RECORD  RESB   100
.
ZERO    WORD   0
RECORD  WORD   100
```

SIC

```
        JSUB   READ
        :
READ    LDX    #0
        LDT    #100
RLOOP:  TD     INDEV
        JEQ    RLOOP
        RD     INDEV
        STHCH  RECORD,X
        TIXR   T
        JLT    RLOOP
        RSUB   // Exit Subroutine

INDEV   BYTE   x'F1'
RECORD  RESB   100
```

SIC/XE

Chapter-02

Assembler — Synchronous class & continued

* Func^{n} of Assembler:

1). Convert mnemonic op. codes to their m/c lang. equivalents
e.g: STL → 14 (Line 10)

2). Convert symbolic operands to their equivalent m/c address
e.g RETADR → 1033 (Line 10)

3). Build the machine inst in the proper format

4). Convert the data const to internal m/c representations.
e.g EOF → 454F46 (Line 80)

5). Write object program & the assembly listing.

E.g -

102A	EOF	BYTE	C'EOF'	
102D	THREE	WORD	3	
1030	ZERO	WORD	0	
1033	RETADR	RESW	1	∵ 1 ∴ increment by 3
1036	LENGTH	RESW	1	If 2, ∴ by 3x2=6
~~103A~~ 1039	BUFFER	RESB	4096	
2039	RDREC	LDX	ZERO	

→ 4096 bytes

★ SYMTAB :

→ Usage –

Pass 1 : labels are entered into SYMTAB with their add (from LOCCTR) as they are encountered.

Pass 2 : operands are looked up into SYMTAB to convert into m/c codes.

★ Pass 1 Algorithm :

```
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
                    begin
```

```
        search SYMTAB for LABEL
        if found then
            set error flg (duplicate symbol)
        else
            insert (LABEL, LOCCTR) into SYMTAB
        end {if symbol}
    search OPTAB for OPCODE
    if found then
        add 3 {instr. 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}.
```

* Object Program:

- Header Record-

Col 1	H
Col 2-7	Program name
Col 8-13	Starting address (hex)
Col 14-19	Total length of the particular program (End - starting add).

Class 3

11/01/2021

* Machine Dependent Features:

- Inst formats & add modes.
- program relocation.

* Machine Independent:-

Immediate add	op #c
Indirect add	op @m
PC or base relative	op m
extended format	+op m
Indexed addressing	op m,x

SIC | XE

+3 → format 3
+4 → format 4

E.g

Line	LOC	Source Pgm			OC
5	0000	COPY	START	0	
10	0000	FIRST	STL	RETADR	17202D
12	0003		LDB	#LENGTH	69202D
13	0006	CLOOP	+JSUB	RDREC	4B101036
15	000A		LDA	LENGTH	
1	000D		COMP	#0	
	0010		JEQ	ENDFIL	
	0013		+JSUB	WRREC	
	0017		J	CLOOP	
	001A	ENDFIL	LDA	EOF	
	001D		STA	BUFFER	
	0020		LDA	#3	
	0023		STA	LENGTH	
	0026		+JSUB	WRREC	
	002A		J	@RETADR	
80	002D	EOF	BYTE	C'EOF'	
95	0030	RETADR	RESW	1	
100	0033	LENGTH	RESW	1	
105	0036	BUFFER	RESB	4096	
125	1036	RDREC	CLEAR	X	
			CLEAR	A	
			CLEAR	S	

→ +1000 to the
∴ 4096 dec
↳ = 1000

If RESW
then 3x10
= 30

SIC | XE

* PC relative

E.g: FIRST STL RETADR (17202D) ← Ans

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

disp = RETADR − PC = 30 − 3 = 2D

∴ | 000101 | 110010 | 0000 | 0010 | 1101 |

1 7 2 0 2 D

- LDB #LENGTH given LENGTH = 0033
  // Immediate add mode LDB = 08

opcode (6)	n	i	x	b	p	e	disp (12)
000010	0	1	0	0	1	0	0000 0010 1101

6 9 2 0 2 D

// format (3) → disp (12)

∴ 0033 − 0006 = 2D. → 02D

↳ PC value for 12 bits

∴ OC → 692 02D

(48) (1036)

- ~~CLOOP~~ +JSUB RDREC // format 4

op(6)	n	i	x	b	p	e	memory (20 bits)
~~000100~~ 010010	1	1	0	0	0	1	0000 0001 0000 0011 0110

4 B 1 0 1 0 3 6

∴ OC = 4B101036

If we consider PC rel & we get -ve add then we can say it is base relative.

00 →0033 (PC relative) ∵ no base value given.

• LDA LENGTH (Ans→032026)

000000	n	i	x	b	p	e	
	1	1	0	0	1	0	0026

∴ 032026

0033 − PC

0033 − 0 = 26

→28

• COMP #0

op(6)	n	i	x	b	p	e	disp(12)
001010	0	1	0	0	0	0	0000 0000 0000
2		9			0		

~~0000 − 0010 = −10~~

CHAPTER-1
Self Study

* SIC Machine Architecture :-

- SIC is a hypothetical computer that includes the hardware features most often found on real machines.
- It stands for Simplified Instructional Computer.
- There are two versions of SIC :-
  (i) Standard model
  (ii) Extension model (XE)

* Memory :-

- SIC supports 15 address bits.
- The no. of possible addresses are $2^{15} = 32768$.
- Thus, 32768 bytes in the computer memory.
- All locations are byte addressable i.e. every location stores 1 byte of data.
- In SIC machine, WORD size = 3 bytes i.e. 3 consecutive bytes form a word (24 bits).

* Registers :-

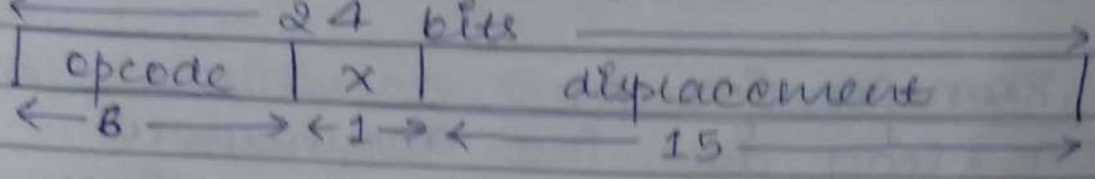
- There are 05 registers supported in SIC
- All of them are 24 bits in length.

A	0	Accumulator, used for arithmetic operations.
X	1	Index register, used for addressing
L	2	Linkage register, JSUB // stores return add
PC	8	Program Counter
SW	9	Status Word, including CC

* Data Formats :-

- SIC only supports Integer & Character.
- Integers are stored as 24 bit binary numbers.
- 8 bit ASCII codes are used for characters.
- SIC does not support floating-point hardware.

* Instruction Formats :-



- The instruction format for SIC is 24 bit long.
- First 8 bits are used for opcode.
- Followed by 01 bit for 'x'.
- Then, 15 bit displacement.
- Here, x-bit represents the addressing mode.

* Addressing Modes :-

- It refers to the way in which the operand of an instruction is specified.

- There are 02 addressing modes in SIC :-

<(i)> Direct
↓
x = 0
↓
target = address add

<(ii)> Indexed
↓
x = 1
↓
target = address + [x] add

* Instruction Set :

1). Data Transfer Group -

- LDA : load data into accumulator
- LDX : load data into index register
- LDL : load data into linkage register
- LDCH : load character into accumulator
- STA : store the contents of A into memory
- STX : store the contents of X into memory.
- STL : store the contents of L into memory
- STSW : store the contents of SW into memory

2). Arithmetic group of instructions -

- ADD
- SUB
- MUL
- DIV

- All arithmetic operations involve register A and a word in memory, with the result being left in the register.

3). Logical group of instructions -

- AND
- OR

> Both involve register A and a word in memory, with the result being left in the register. Condn flag isn't affected

- COMP → compares the value in ~~integer~~ register A (<, >, =) with a word in memory, this inst sets a condition code cc to indicate the result.

4). Branch Group of Instructions –

a). Conditional Jump Instructions :-

- JLT
- JEQ
- JGT
- these inst test the setting of CC (<, =, >) & jump accordingly.

b). Unconditional Jump Instructions :-

- J → this inst without testing the setting of CC, jumps directly to assigned memory.

c). Subroutine Linkage :-

- JSUB - jumps to subroutine, placing the return address in register L.
  - L ← PC, PC ← subroutine address.

- RSUB - RSUB returns by jumping to the address contained in register L.
  - PC ← L

* Input & Output:

I/p & o/p are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A.

- **TD** // test device, tests whether the addressed device is ready to send or receive a byte of data.

If CC = '<' the device is ready
CC = '=' the device is not ready.

- **RD** // read data.
// data from the input device specified by the memory is read into A lower order byte.

- **WD** // write data.
// Data is sent to output device specified by the memory.

- **TIX** // Increments the content of X and compares its content with memory.

Depending on the result the condn flags are updated

If (X) < (m) then CC = '<'
(X) = (m) then CC = '='
(X) > (m) then CC = '>'

1000 1000
300 300

* SIC/XE Machine Architecture:-

- It is an extended version of SIC machine.

* Memory:-

- SIC/XE supports 20 address bits.
- Thus, 2^{20} bytes in computer memory.
- The memory structure is same as that for SIC

* Registers:-

In addition to A, XL, PC, SW, it has B, S, T, F registers.

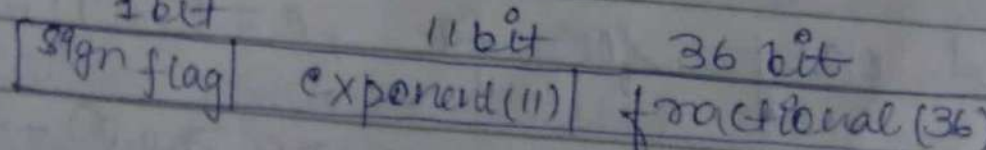
- All registers are 24 bits except F (48 bits).

	No.	
B	3	Base register, used for addressing
S	4	General working register
T	5	General working register
F	6	Floating-point accumulator (48 bits)

* Data formats:-

SIC/XE supports:

- Integer → 24 bits
- ~~Fraction~~ → 48 bits → Floatp
- Character → 8 bits

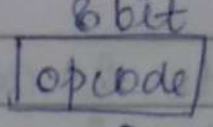


abs value = frac * $2^{(exp-1024)}$

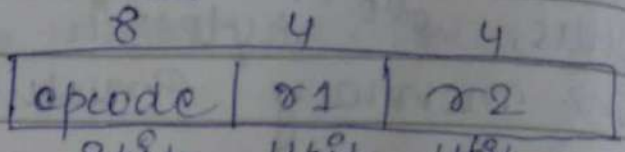
* Instruction Formats :

format 1 & 2 are instructions that do not refer memory at all.

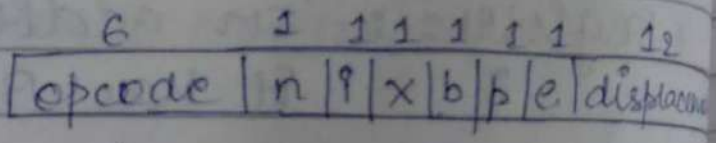
Format 1 (1 byte)



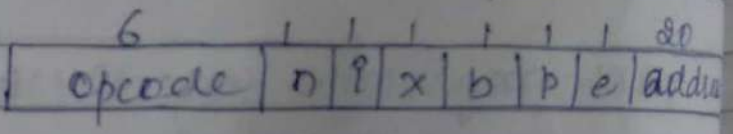
Format 2 (2 bytes)



Format 3 (3 bytes)



Format 4 (4 bytes)



n → Indirect addressing
i → Immediate addressing
x → Indexed addressing
b → base addressing
p → pc relative addressing
e → { 0 → f3
1 → f4 }

extended only for f4 , e=1 ; n=1, i=1

LDA [NUM]
↳ [NUM]-PC if PC > NUM → Base relative addressing mode

MUL SRC, dest

Exercise 1.3 Pg-40

1). ALPHA = BETA * GAMMA for SIC

Chapter 01 (LP)

Introduction to machine Architecture

1). Define System Software & give examples.

System software consists of a variety of programs that support the operation of a computer.

Examples : Operating system, System compiler assembler, macro processor, loader or linker, debugger, text editor, dbms & software engineering tools.

2). Difference between System software and application software.

- System software consits a variety of progsam that support the operation of a computer.
- Application software focuses on an application or problem to be solved.

3). List the addressing modes of SIC & SIC/XE with one example each.

SIC: There are two addressing modes in SIC.

<i> Direct (X=0)
Target add = add
E.g: LDA TEN

<ii> Indexed (X=1)
Target add = add + [X]
E.g: STCH Buffer, X

SIC/XE:

<i> Base Relative : n=1 i=1 b=1 p=0
<ii> PC relative : n=1 i=1 b=0 p=1
<iii> Direct : n=1 i=1 b=0 p=0
<iv> Immediate : n=0 i=1 x=0
<v> Indirect : n=1 i=0 x=0
<vi> Indexing : n=0, i=0/1, x=1
<vii> Extended : e=1

Rule 1:

e=0 : format 3
e=1 : format 4

- format 3 →
  - b=1, p=0 (base relative)
  - b=0, p=1 (PC relative)
  - b=0, p=0 (direct addressing)

- format 4 →
  - b=0, p=0 (direct addressing)
  - x=1 (Index)
  - i=1, n=0 (Immediate)
  - i=0, n=1 (Indirect)
  - i=0, n=0 (SIC)
  - i=1, n=1 (SIC/XE for SIC compatibility)

Rule 2:

i=0 n=0 (SIC)
b, p, e is part of the address

★ Base Relative → 0 ≤ disp ≤ 4095 (decimal) (FFF)

★ PC Relative → -2048 ≤ disp ≤ 2047 (7FF)

Symbol-Defining Statements

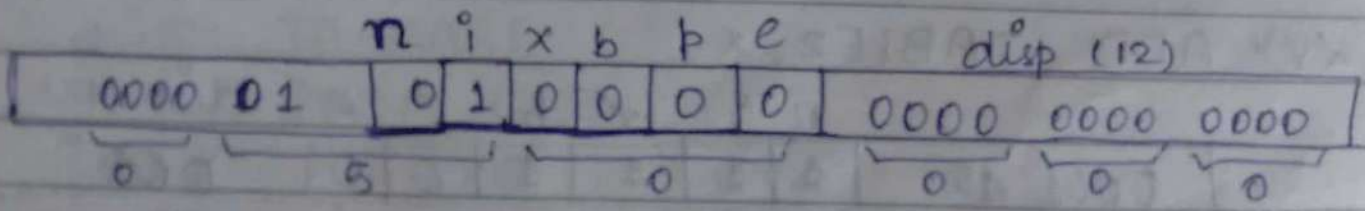
LDX = 04 LDA = 00 LDB = 68 STA = 0C

ADD = 18 TIX = 2C JLT = 38 RSUB = 4C

★ SIC|XE :-

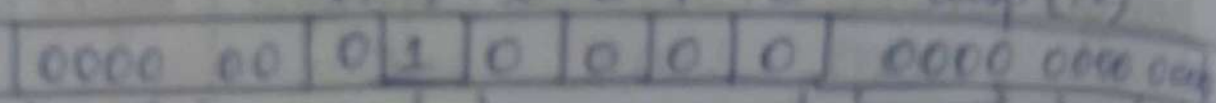
loc	length	label	Mnemonic	operand	object code
	3	SUM	START	0	
0000	3		LDX	#0	050000
0003	3		LDA	#0	010000
0006	4		+LDB	#TABLE2	69100000
			BASE	TABLE2	
000A	3	LOOP	ADD	TABLE,X	1BA013
000D	3		ADD	TABLE2,X	1BC000
0010	3		TIX	COUNT	2F200A
0013	3		JLT	LOOP	3B2FF4
0016	4		+STA	LOOP	
001A	3		RSUB		
001D	3	COUNT	RESW	1	
0020	2000 X 3 = 1770	TABLE	RESW	2000	
1790	2000 X 3 = 1770	TABLE2	RESW	2000	
2F00	3	TOTAL	RESW	1	
2F03			END	FIRST	

1) LDX #0



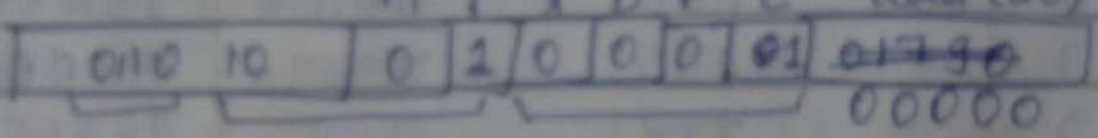
0C = 050000

<(ii)> LDA #0



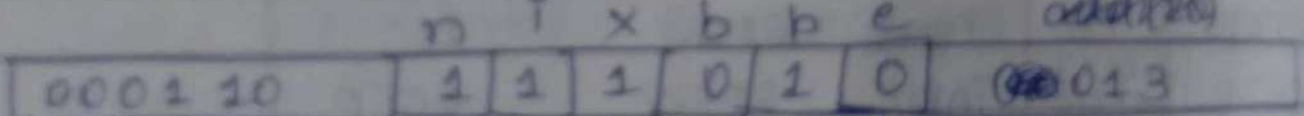
OC = 010000

<(iii)> +LDB #TABLE2



69100000

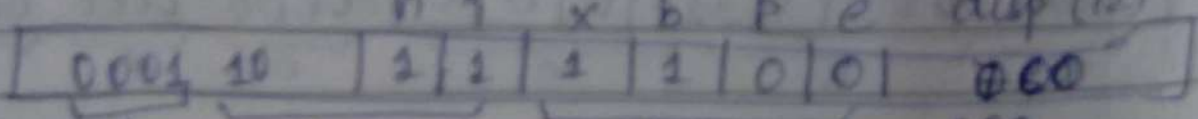
<(iv)> ADD, TABLE, X



add = TA - PC
= 0020 - 000D
= 13 → PC

OC = 1BA013

<(v)> ADD TABLE2, X



add = 1790 - 0010
= 1780 → dec: 6016 (>4095)
∴ Base relative ∴ disp = TA - BASE
= 1790 - 1790
= 00

OC = 1BC000

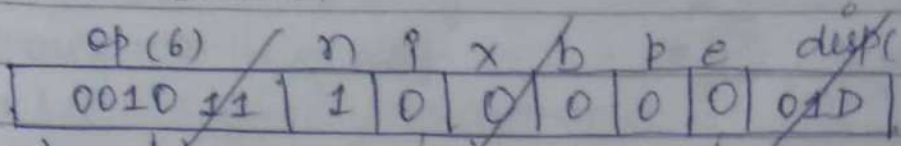
001D ⟷ · TIX COUNT

0013 n=1 e=0

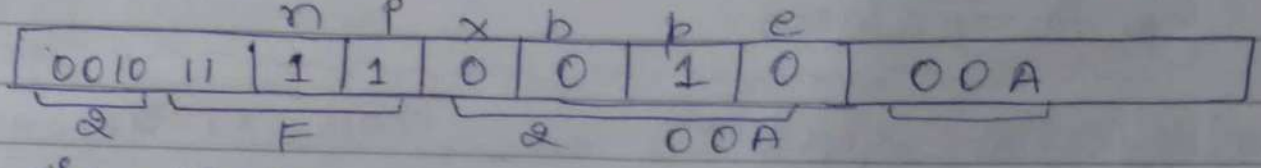
disp = 001D − 0013 i=1 x=0

= A → +ve b= p= FFF4

(vi) TIX COUNT



OC = 2E001D



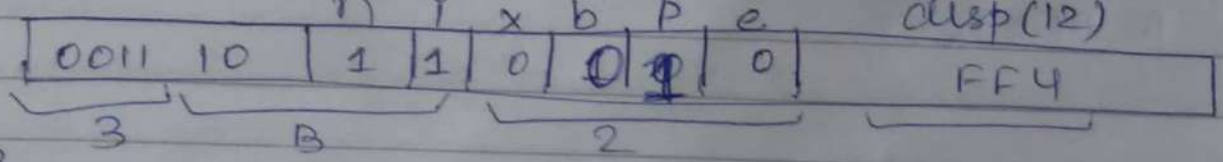
disp = TA − PC

= 001D − 0013

= A ⟶ +ve ∴ PC relative

OC = 2F200A

(vii) JLT LOOP



disp = ~~LOOP~~ TA − PC = 000A − 0016

= FFF4 ⟶ dec: −12

~~∴ Base relative~~

∴ ~~disp~~ = TA − BASE PC relattie

= 000A − 1790

= FE~~F~~7A ⟶ ~~dec: −6022~~

disp = TA − PC

= FFF4 → (−12) ✓ (PC)

H ^ <name of program> — — ^ starting add (6) ^ length (6)

T ^ starting add ^ 1E (30) ^ ---- first 10 records OC...

T ^ starting add of 11th record ^ 30 - 9 = 21 = 15 Hex ^ ---- next 10 records OC...

T ^ -

E ^ starting address