4/8/17 Friday

MODULE - I

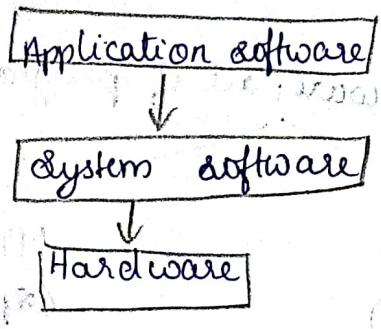
Software (set of programs)

System software
(General purpose s/w)
- Brain & heart
- controls & coordinates the operations of computer s/m
- manage s/m resources
- provides support functionality for applications.
- eg:- OS, DBMS, translator, text editor, macro-preprocessor, linker, loader

Application Software
(special purpose s/w)
eg:- Browser, VLC, iTune, ms office suite.
- executes user specific task

Application software	System software
1. Special purpose s/w (meet user needs)	1. General purpose s/w (meets s/m needs)
2. Not essential	2. Essential s/m
3. Executed as and when needed	3. Executed all the time.
4. Creates its own environment or work in the environment provided by system s/w	4. Provides environment for application software
5. Control hardware with the help of system software	5. Directly control hardware.
6. No of Application software is more	6. No: of system software is less

9/8/17 wednesday



System Software

System support software

↳ support the execution of application

↳ Eg:- translators, debuggers, text editor

↳ Translators → compiler, Interpreter, Assembler

System control software

↳ controls & coordinate system

↳ Eg:- OS, DBMS

Types of system software

(1) DBMS

(2) OS

(3) Text editor

(4) Macroprocessor (used to process macro), (macro substitution)

(5) Translators

(6) Debuggers

(7) Device driver

(8) Linker

(9) Loader

Text editor

* Software that help to create
  - program file
  - configuration file
  - Doc file

* Editing
  - cut
  - copy
  - paste
  - undo/Redo
  - Syntax highlighting

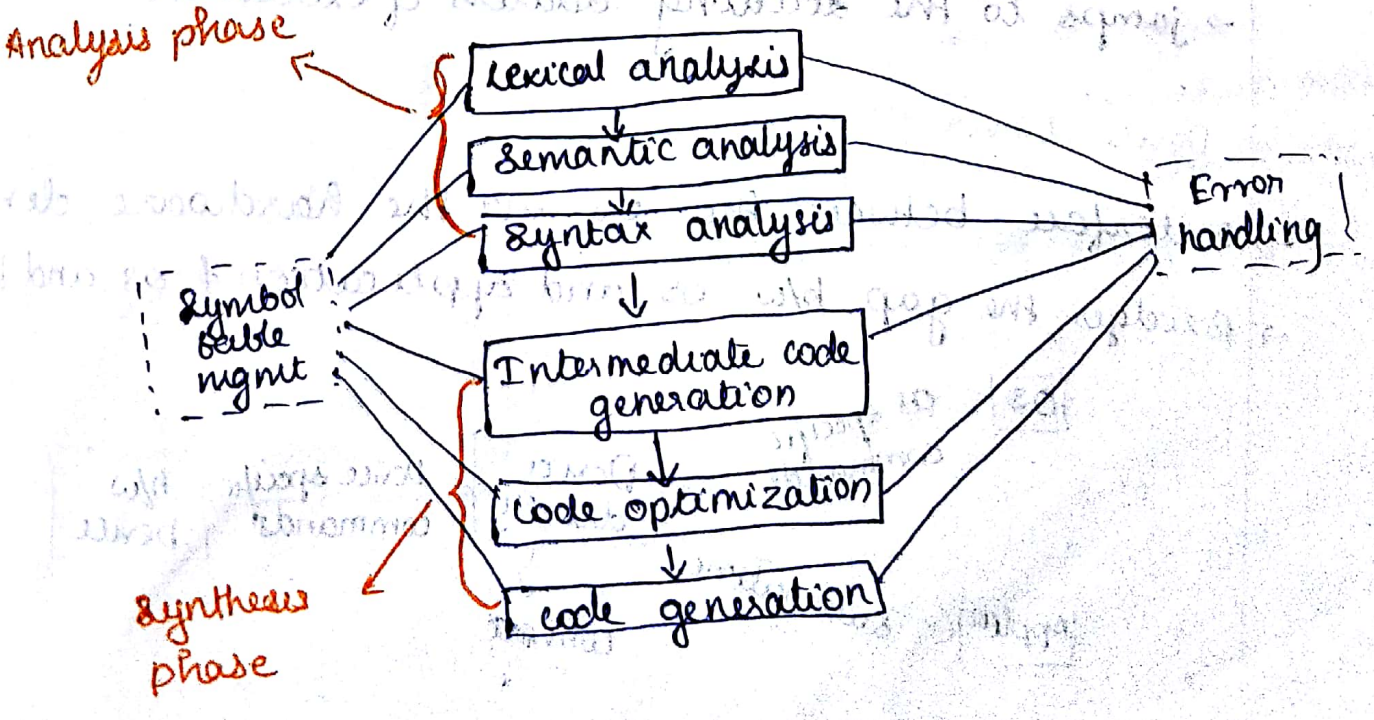
* Eg:- VI, Gedit, Textpad

Compiler

High level program → compiler → M/c code

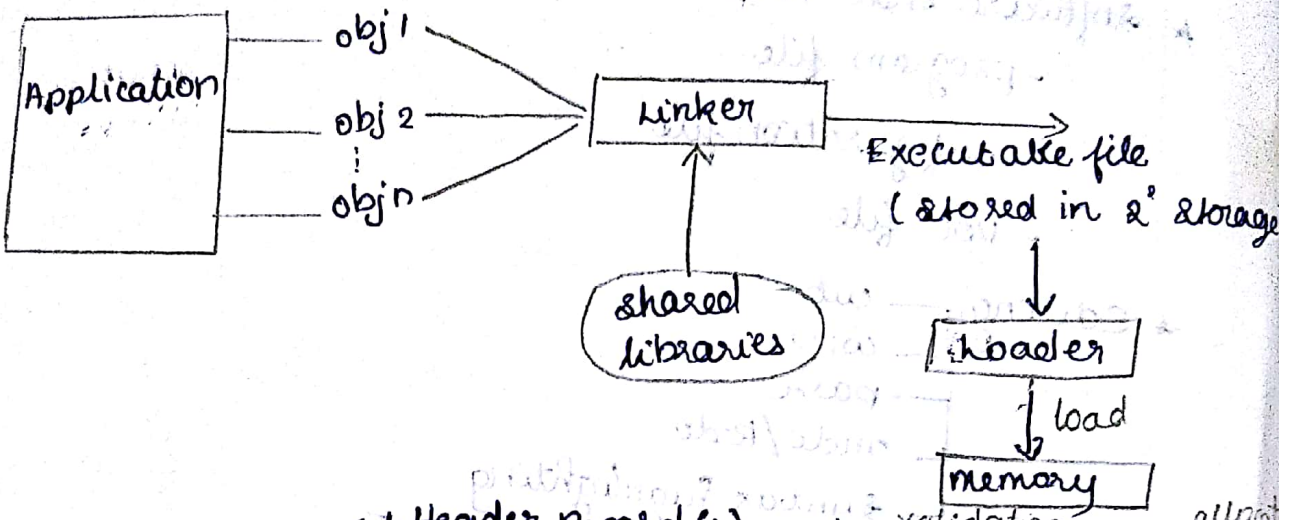
compilation

1. Analysis phase → Intermediate code (Quadruple three address code)
2. Synthesis phase → Generate object code



10/8/17 Thursday

Linker



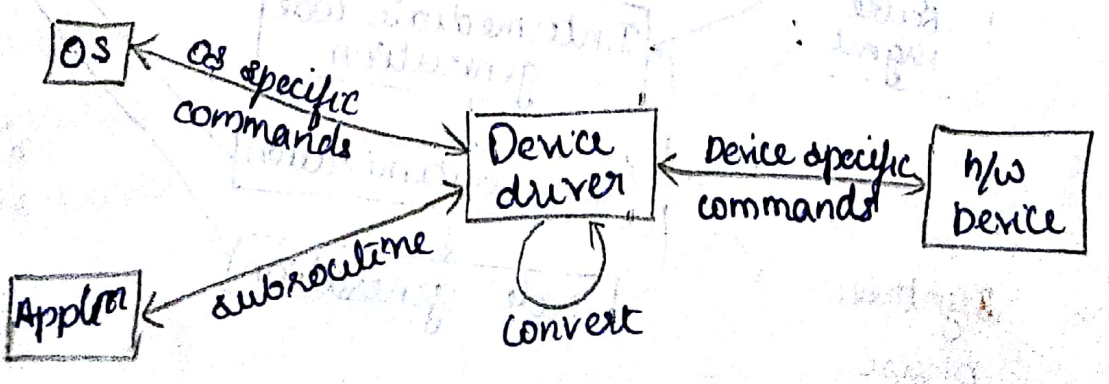
Executable file
1. Header Record (1) → (store validation pgm name + size) allocate
2. Text Record (1...n) → (stores object code)
3. End Record (1) → (starting address of execution)

Loader

→ validates the program name
→ allocates the size of the program
→ loads the object code
→ jumps to the starting address of execution

Device driver

→ Interface between the OS and the hardware device.
→ Bridges the gap b/w OS and application & OS and Hardware



11/8/17 Friday

SIC (Simplified Instruction Computer)

→ Hypothetical computer (model, simulator)
→ simplified hardware features
→ has 2 versions
  - standard (SIC)
  - Extended (SIC/XE)

Architecture of SIC

(1) Memory

→ 1 byte = 8 bit
→ 1 word = 3 bytes consecutively
→ memory address = 15 bits
→ memory address = 2^{15} = 32768 Bytes

(2) Registers

→ 5 registers
→ 24 bit length
→ registers have character or numeric representation

① A - (0) - Accumulator (Arithmetic operators)
② X - (1) - Index (address calc^n) → stores offset
③ L - (2) - Linkage (subroutine exec^n) → stores return address
④ P.C - (8) - Program Counter → stores address of nxt instruct^n to be exec^d
⑤ S.W - (9) - Status word (stores carry flag + overflow flag)

Character representat^n / Numeric representation

24/8/17 Thursday

Data Formats

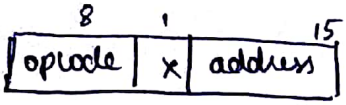
Integer : 24 bit binary no

: negative numbers - 2's complement

Character : 8 bit ASCII

no floating point representation

Instruction Format



Instruction length : 24 bit

Addressing modes in SIC

↳ The way in which operand is specified in the instruction

[TA - Target address]

Direct
- → x=0
- → TA is given in the instruction

Indexed
- → x=1
- → TA = Base + (X) — the value stored in index register

Types of Instructions in SIC

1. Load/store
   - LDA (A ← M)
   - LDX (X ← M)
   - STA (M ← A)
   - STX (M ← X)
   - LDCH [(A) ← (M)]
   - STCH [(M) ← (A)]

2. Integer Arithmetic
   - ADD [(A) ← (A) + (M)]
   - MUL [(A) ← (A) x (M)]
   - SUB [(A) ← (A) - (M)]
   - DIV [(A) ← (A) / (M)]

3. COMP
   - ↳ used to compare
   - ↳ compare the value stored in memory with the value in accumulator

[A - accumulator, M - memory, X - Index register]

↳ if (A) < (M) cc = '>'
elseif (A) > (M), cc = '<'
else (A) == (M)) cc = '='

LDCH → to load characters from memory to Accumulator

4. conditional Jump Instn

- JLT [Jump if <]
- JEQ [Jump if cc '=']
- JGT [Jump if >]

5. subroutine Linkage

- JSUB (Jump to subroutine)
- RSUB (Return from the subroutine)

→ Also to store return address to linkage register.

6. Input/Output

TD →
RD →
WD →

Load - value from memory is loaded to accumulator.

Store - to store the value in accumulator to memory

⇒ All Load/Store instruction only has a single operand i.e., memory

⇒ Integer Arithmetic only has a single operand [i.e, a Label]

⇒ comp is used before the jump instruction

Jump Instruction works on the basis of the value stored in cc code.

★ example for jump instruction

```
LOOPI  LDA  ALPHA
       STA  BETA
       ADD  BETA
       COMP GAMMA
       JLT  LOOPI
       LDA  GAMMA
```

* Example for Input/Output Instruction

```
LOOP1 : TD INDEV
        JEQ LOOP1
        RD INDEV
        WD OUTDEV
```

INDEV - Has input device identifier

OUTDEV - Has o/p device identifier

TD: Test whether a device is ready or busy

if the device is ready, ⇒ CC = '<'

if the device is busy ⇒ CC = '='

RD: Read from the device

WD Write to the device

25/8/17 Friday

ASSEMBLER DIRECTIVES

→ direction to assembler

→ Not translated to object code

1. START

eg: COPY START 1000

label - pgm name | starting address.

2. END

→ indicate end of the program

→ give the address of the first executable instruction

3. BYTE

eg:- ABC BYTE C'EOF'
hexadecimal

4. WORD

eg:- THREE WORD 3
Generate integer constant

5. RESB

X RESB 4096 // Reserve 4096 bytes

6. RESW

RESW 2 // 2 words will be reserved

SIC PROGRAMMING

Example

```
        LDX   ZERO
MOVECH  LDCH  STR1,X
        STCH  STR2,X
        TIX   ELEVEN
        JLT   MOVECH
STR1    BYTE  C 'TEST-STRING'
STR2    RESB  11
ZERO    WORD  0
ELEVEN  WORD  11
```

9) Write a sequence of instruction for SIC to set ALPHA equal product of BETA & GAMMA

Answer

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

Q) Write an SIC program to swap the values of a and b

Given a = 2
b = 5

Ans

```
LDA  A
STA  T
LDA  B
STA  A
LDA  T
STA  B
A  WORD 2
B  WORD 5
T  RESW 1
```

A - 1010 - 10
B - 1011 - 11
C - 1100 - 12

SIC/XE

Memory

Address length = 20 bit

memory capacity - 2^{20}

Data format

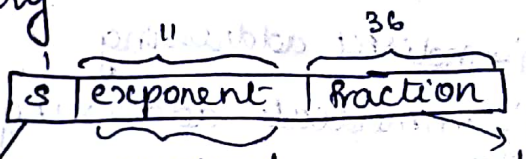
character - 8 bit ASCII

Integer - 24 bit

negative no - 2's complement

floating point - 48 bit binary

floating point representation:



s=0 positive
s=1 negative

unsigned integer value b/w 0 & 2047

value b/w 0 & 1

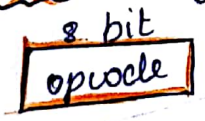
Instruction format

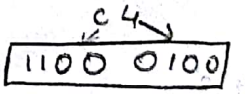
Appendix A SIC/XE (Pg 1417)

Instruction format in SIC/XE:

1) format 1 (1 byte)

- length of instruction in 1 byte





eg:- FIX (A) ← convert (F)

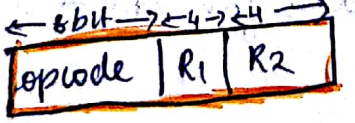
C4 is the opcode of FIX

Floating points converted to integer & stored in accumulator

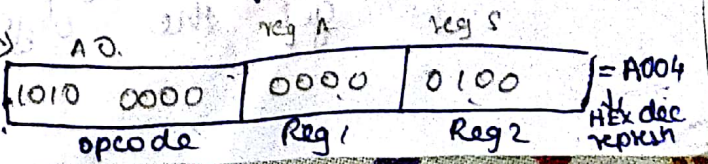
2) format 2 (2 bytes - instruction length)

- opcode followed by 2 reg.

Eg: COMPR A,S



★ Opcode of COMPR is A0

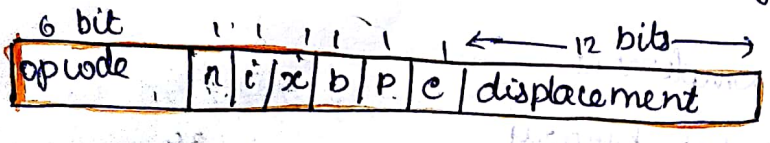


SIC/XE Registers 4 additional registers are present in SIC/XE

3 - B (Base register)
4 - S
5 - T } General purpose } 24 bit long
6 - F → Floating point - 48 bit long

numeric representation

3) Format 3 (3 byte) - Instruction length is 3 byte



'n' → indirect addressing
'i' - immediate addressing mode eg:- LDA #3
'x' - indexed addressing mode.
Relative addressing mode
└ Base relative (b=1) TA = (B) + displacement
└ program counter (p=1) TA = (P.C) + displacement

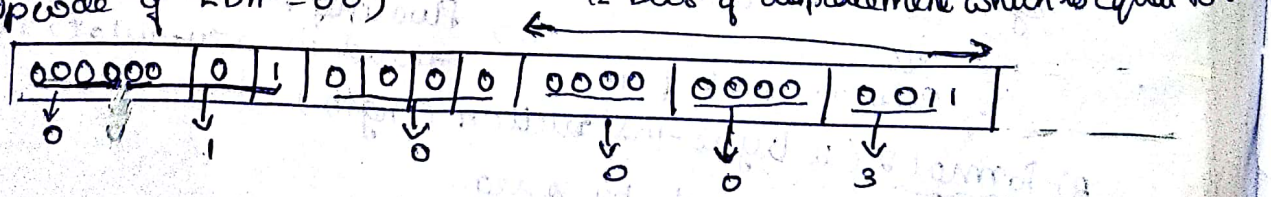
If displacement b/w
↳ 0 to 4095 Base relative
↳ -2048 to 2047 Program counter

30/8/17 Wednesday

eg:

★ LDA #3

(opcode of LDA = 00)



→ 'e' is 1 only for format 4
→ instruction bits are grouped to 4 bits entirely
∴ we get the object code for LDA #3 as 010003

eg 2

```
1000   LDA    ALPHA
1003   CLOOP  RESB 1
        .
        .
        .
1009   ALPHA  RESW 1
```

Note: Here we can see that, this question (LDA ALPHA) is program counter relative mode. (by default i, n=1 & p=1

b=0 p=1

To find target address,

$$TA = (P.C) + \text{displacement}$$

$$\therefore \text{displacement} = TA - (P.C)$$

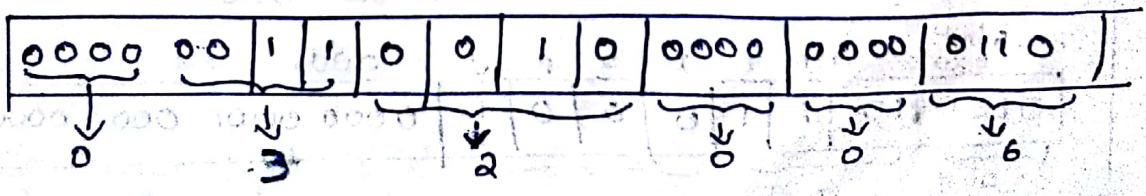
TA → (address corresponding to ALPHA)

T.A = 1009 (operand addr)

P.C = (next instruction after (1000) (LDA address))

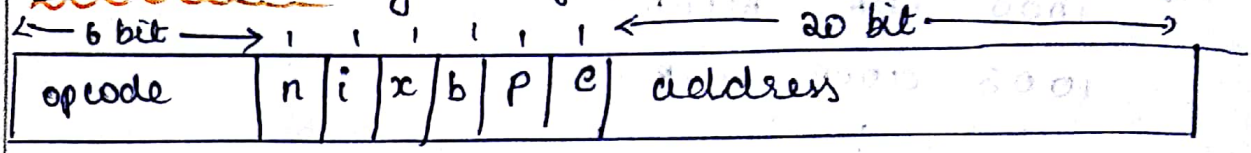
P.C = 1003

$$\therefore \text{displacement} = TA - (P.C)$$
$$= 1009 - 1003$$
$$= 6 \quad \text{(written in 12 bit)}$$
$$= 006$$



∴ object code : 032006

4) FORMAT 4 (4 byte length)



Eg

```
1000   +JSUB   ALPHA
1003   CLOOP   RESB 1
  .
  .
  .
1009   ALPHA   RESW 1
```

Ans

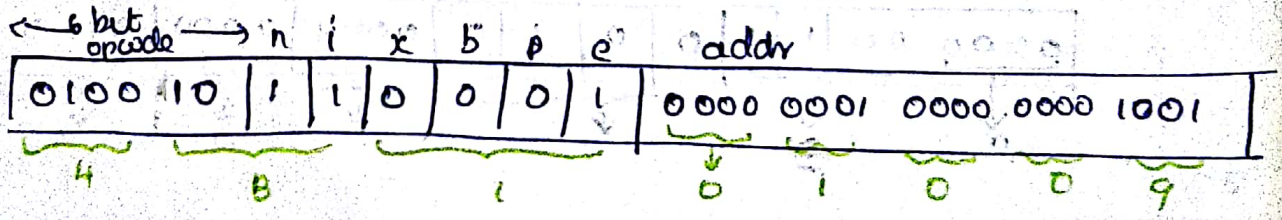
NOTE: + denotes format 4

- opcode of JSUB = 48
- to represent 48 in 8 bits → 0100 1000 (4, 8)
- first 6 bits are opcode 010010 ← 0100 1000
- by default n,i,e = 1 ; all others = 0
- address in 20 bits

here 1009 → 0000 0001 0000 0000 1001

0 1 0 0 9

20 bits



→ group as 4 bit entirely

∴ the object code is 4B101009

Textbook appendix A

SIC x/E Neumonic

Pg: 473 (470 → 473)

PROGRAMS

Q) Write an SIC x/E program to set ALPHA equal to the integer portion of BETA / GAMMA

A NOTE : round the floating point number to integer of beta & gamma after dividing & then store it in ALPHA

i.e., $\alpha = \beta / \gamma$

```
LDF   BETA        [load B to F register]
DIVF  GAMMA       [div B by γ & store to F regist]
FIX               [floating no: to integer & store in accumulator]
STA   ALPHA       [store accumulator value to α]

ALPHA  RESW  1
BETA   RESW  1
GAMMA  RESW  1
BETA   WORD  7    (reserve for integer using word)
GAMMA  WORD  3
```

Q) WAP to clear 20 byte strings with all blanks.

11/9/17 monday

MODULE II

ASSEMBLER

Functions

- Converts operation code to machine code

OPTAB { LDA ALPHA
⇓
00
m/c code

- Converts symbolic operand to machine address.
- Generation of object code for each assembly instruction.

* Two types of assembler

Single pass
→ Generates object code in 1 pass

Two pass
→ Generates object code in 2 pass

* Forward reference :- Reference to label defined later in program

eg:-
```
1003   CLOOP   LDA   RETADR
                .
                .
                .
1033   RETADR  RESB  1
```

This is the problem with single pass assembler resolved using two pass assembler.

⇒ Two pass assembler, SYMTAB. In first pass it stores all labels and its corresponding address in SYMTAB & in second pass generates op-code

Data structures

1) OPTAB (Operation code Table)

opcode	machine code	Format	length
LDA	00	3	3

→ While generating machine object code of opcode, its machine code is required, this is obtained through OPTAB.

→ Construction during pass I

2) SYMTAB (Symbol Table)

Relative - R
Absolute - A

SYMBOL	TYPE	ADDRESS
RETADR	R	1033
CLOOP	A	2033

→ Used to store symbols and its corresponding address.

→ Construction during pass I

3) LOCCTR (Location Counter)

→ Single data structure used to store the address of each instruction.