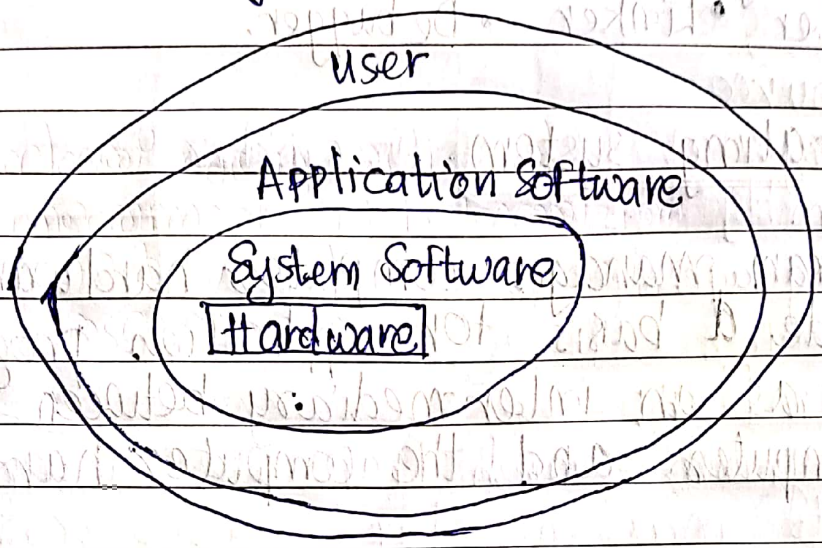
KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module - 1

Software → Application software
→ System software.



- Program: collection of executable instruction.
- Software: collection of programs, procedures associated with operation of computer system.

Application software: Software perform specific task for the end user as per the requirement.
eg: photoshop, Ms-office, Media players etc.

System software:

- programs that support the operation of a computer.
- Makes it possible for the user to focus on an application or others problem to be solved.
- User donot required to know the internal details of how machine works internally

eg: compilers, Assemblers etc.

Different types of System Software:

- OS
- Device Drivers
- Text editors
- Macro Processors
- Language Translators :- Assembler, Compiler, Interpreter
- Loader,
- Linker
- Debugger.

Operating System.

- Program manages computer hardware
- provide a basis for application programs and acts as an intermediary between a user of a computer and the computer hardware

Eg: Mac Os, Microsoft Windows, GNU/Linux, Google's Android Os, iOs

Functions of Operating System:

- Memory Management.
- Processor Management
- Device management
- File management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users.

Device Drivers:

- Glue between Os. and Its I/O devices.
- Program that controls a particular device (keyboard, Printer, Scanner) that is connected to your computer.
- Acts as translators- converts generic requests received from the Os into commands that specific

peripheral controllers can understand.

Text Editors

- Allows users to create and device documents
- Earlier - documents are plain text
- Now - Improved input-output mechanisms and file formats varies within the document.
- Text editors with advanced features.
  eg: Gi-edit.

-type of generated outputs, editors classified as:
- Line editors - Edlin in early MS-DOS
- Stream editors - sed in UNIX (1973)
- Screen editors - Vi editor
- Word processors
- structure editors - visual studio

to reduce repeated set of codes

```
#define pi 3.14;
int area = pi*r**r;
```

use of macros.

Macro Processors

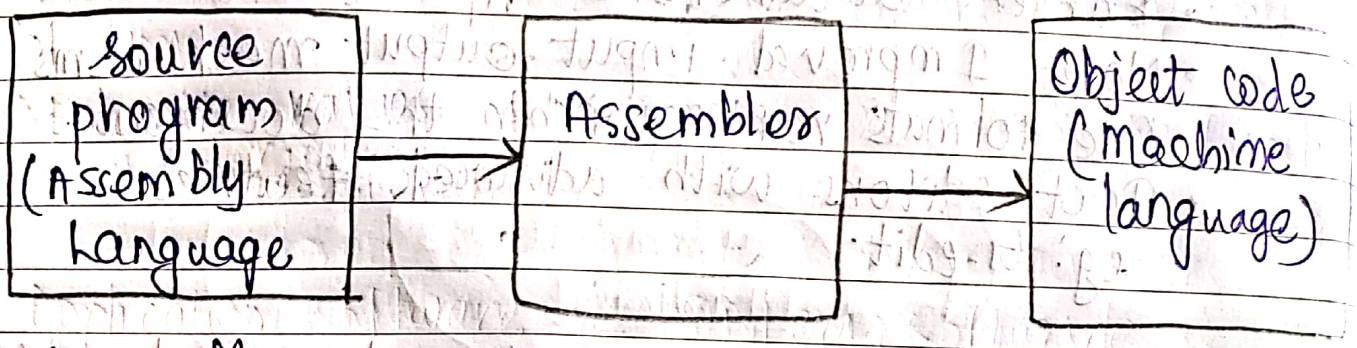
- program that copies a stream of text from one place to another, making a systematic set of replacement.
- often embedded in another programs- assembler, compiler.
- Sometimes as standalone programs
- macro definition - invocation - expansion.

Language Translators

- Computer are basically machines that follow very specific and primitive instructions.
- Programs written in another (high/low level) language have to be translated into machine

language.

1. Assembler
2. Compiler
3. Interpreter.



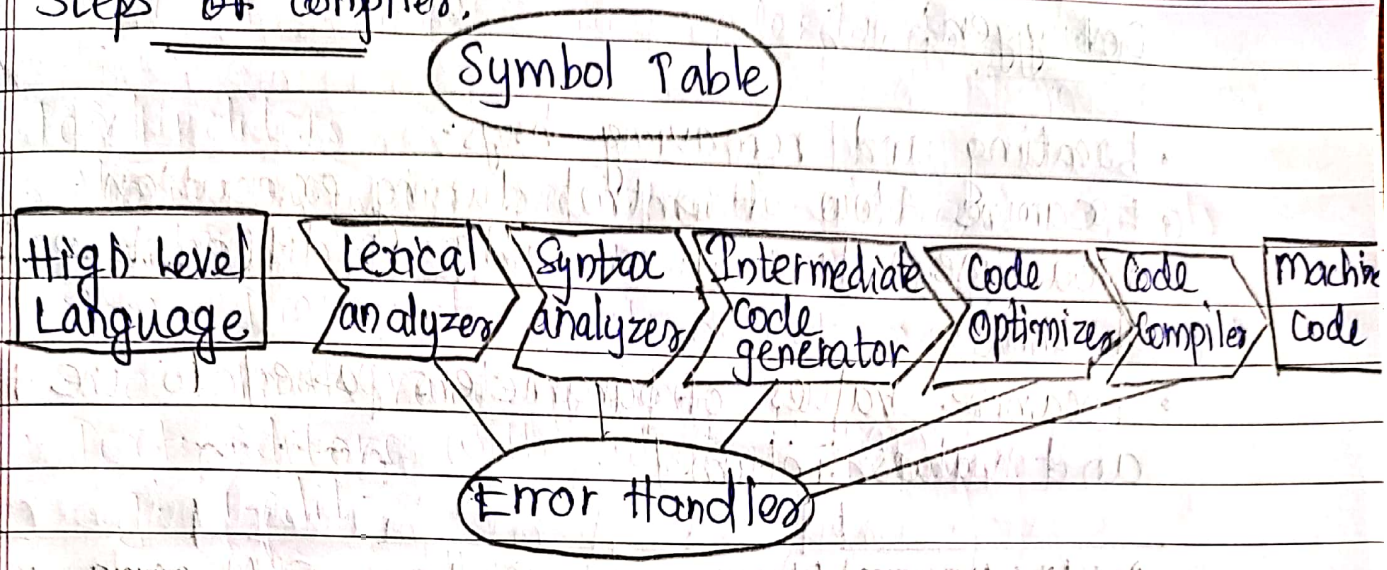
Assembler:

- program which translates assembly language into machine language is called assembler.
- self assembler: runs on a computer and produce machine code for the same computer
- Cross Assembler: runs on a computer and produce machine code for another computer

Compiler	Interpreter
• Intermediate code generated	• No Intermediate code generated
• Runtime is more	• Less Run time
• High memory use	• Low memory use.
• C, C++	• Python, Ruby
• program as a whole translated and executed	• Line by line translation and execution.
• Slow Speed	• Faster

gcc - GNU compiler Collection.

Steps of Compiler:

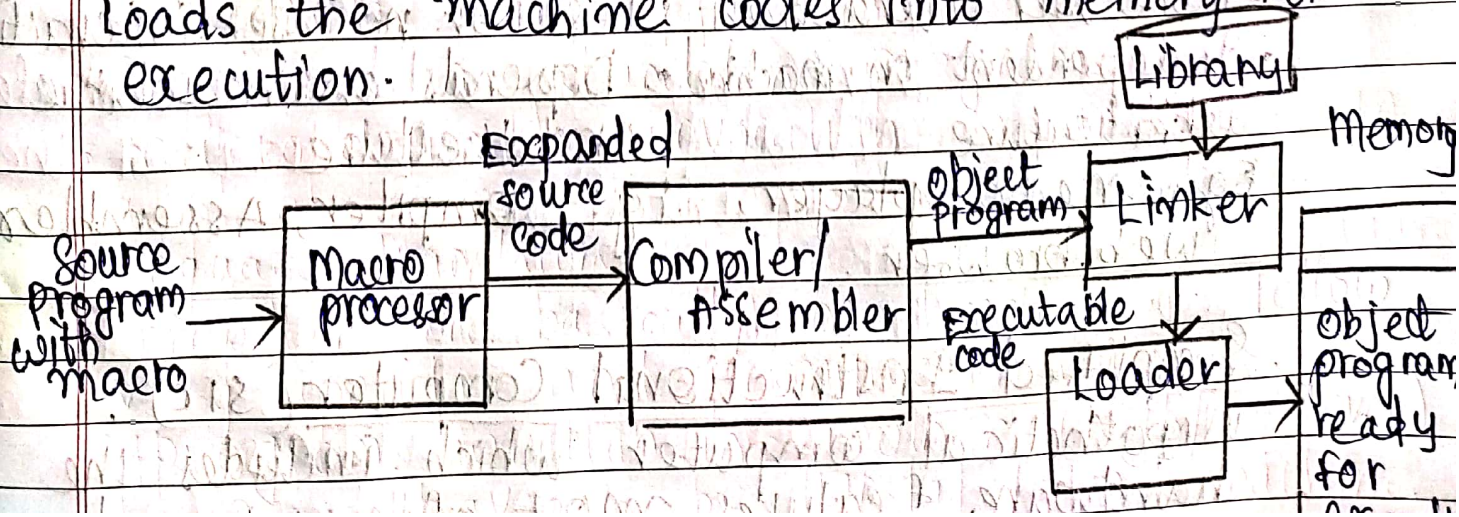


Linker:

- High level Language - header files or libraries (built in / user defined)
- Links the program with functions defined in the Libraries.
- If not found library - linker informs the compiler then it reports an error.
- Compiler automatically invokes the linker
- Longer programs - modules - Linker links the module.

Loader

Loads the machine codes into memory for execution.



Identifying and removing errors-

Debugger:

- Locating and removing bugs.
- Examine flow of control during execution.
- Examine values of variables at different points of time.
- Examine values of parameters passed to the function and values returned.

Application software	vs	System software
• Application software is the type of software which runs as per user request - runs on the platform provided by the system software		• It is the type of software which is the interface between application software and the system.
• Developed in high level languages		• Are developed in low level language which is more compatible with the system hardware in order to interact with.
• used by user to perform a specific task		• Installed on the computer when the OS is installed
• More user interaction		• Less or no user interaction.
• Independent on machine architecture		• Depends on machine architecture
Eg:- media player, web browser		Eg:- Compiler, Assembler.

Simplified Instructional Computer (SIC):

A hypothetical computer which includes the hardware features most often found on

TA - Target address.

JSUB - Jump to Subroutine.

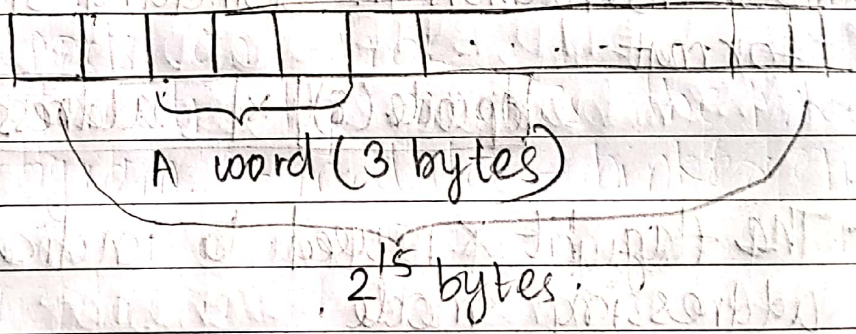
real machines.
- SIC (standard model)
- SIC/XE

XE - Extra Equipment
EOctra, Expensive.

- upward compatible.
  - programs for SIC can run on SIC/XE

SIC machine Architecture

- memory
  - 8 bit bytes
  - Any 3 consecutive bytes form a word (24 bits)
  - All addresses on SIC are byte addresses
  - words are addressed by the location of their lowest numbered byte.
  - 2^{15} (32768) bytes in the computer memory



- Registers
  - 5 registers and each register is 24-bit in length

Mnemonic	Number	Special Use
A	0	Accumulator, for arithmetic operations
X	1	Index register, for addressing
L	2	Linkage register
PC	8	program counter
SW	9	status word.

Linkage register: Jump to Subroutine (JSUB) instruction stores the return address in this register.
Status Word: contains a variety of information including a condition code (CC)
PC - contains address of the next instruction to be executed.

) Data format:
- Integer:
  - stored as 24-bit binary numbers.
  - negative numbers are stored as 2's complement representation.
- characters
  - Stored using their 8 bit ASCII codes.
  - There is no floating point hardware on the standard version

) Instruction format:
- all instruction in standard SIC are 24-bit format

opcode (8)	x	address (15)

- The flag bit x is used to indicate indexed addressing mode.

) Addressing mode:
There are two addressing modes available, indicated by the setting of the x bit in the instruction.

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

LDA - load into Accumulator

- Target Address is calculated from the address given in the instruction.
- parathesis are used to indicate the contents of a register or a memory location
(X) - represents contents of the register X.

- Instruction Set:
  - Instruction for load and store registers
    - LDA, STA, LDX, STX, LDL, STL
  - Instructions for integer aritmetic operation
    - ADD, SUB, MUL, DIV
    - All arithmetic operations involve register A and a word in memory with result being left in the register.
- Instruction Comp
  - compares the value in register A with a word in memory.
  - this sets a condition code, CC to indicate the result (<, =, >)
- Conditional Jump Instructions
  - JLT, JEQ, JGT
  - Test the setting of CC and jump accordingl
- Instructions for subroutine Linkages.
  - JSUB - jumps to the subroutine, placing the return address in register L.
  - RSUB - returns by jumping to the address contained in the register L.
- Input/ Output Instructions
  - Input and output are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A.
  - Each device is assigned a unique 8 bit code.

RESW - Reserve Word.
RESB - Reserve Byte.

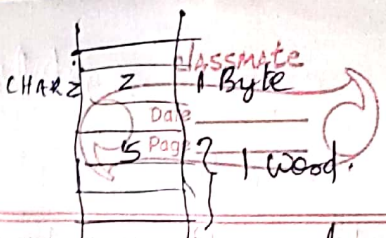
- Test Device (TD): This instruction tests whether the addressed device is ready to send or receive a byte of data. Condition code is set to indicate the result of this test.
  - If <, means the device is ready to send or receive data
  - If =, means the device is not ready.
- A program needing to transfer data must wait until the device is ready
  - RD (Read Data)
  - WD (Write Data)
- This sequence must be repeated for each byte of data to be read or written.

Assembler Directives:

- Pseudo Instruction:
- START, END, BYTE, WORD, RESW, RESB are - assembler directives.

4 ways of defining storage of data items in the SIC Assembler Language.

1) BYTE: Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant
   - Eg: CHARZ BYTE C'Z'
2) WORD: Generate one-word integer constant
   - Eg: FIVE WORD 5.



RESB : Reserve the indicated bytes for a data area

RESW : Reserve the indicated ~~number~~ words for a data area.

Data Movement : Program 1

prgrm:1

RESW 1 : Reserve one word in the memory.

ALPHA	RESW	1	One Word variable
FIVE	WORD	5	One word constant
CHARZ	BYTE	C'Z'	One Byte constant
C1	RESB	1	One (one byte will be reserved somewhere in memory) Byte variable.

Load accumulator

LDA	FIVE		
STA	ALPHA		
LDCH	CHARZ	load the accumulator with a character.	
STCH	C1		

A		5

Alpha	5	1 word
FIVE	5	1 word
CHARZ	Z	1 Byte
C1	Z	1 Byte

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

- There is no memory-memory move instruction
- So all data movement must be done using registers.

BETA
$= (ALPHA + INCR - 1)$
DELTA $= (GAMMA + INCR - 1)$

Program 2: Arithmetic Operation: - SIC program

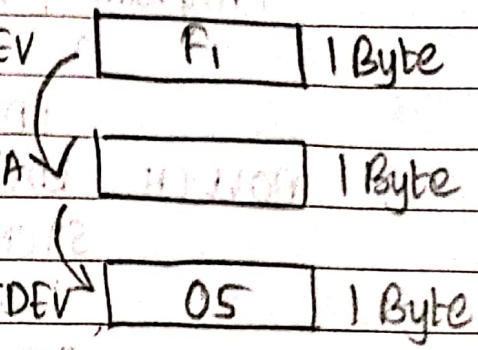
One	1	1 word
Alpha		1 word
Beta		1 word
Gamma		"
Delta		"
INCR		"

```
LDA   ALPHA   Load Alpha into register A.
ADD   INCR    Add the value of INCR, A=A+1
SUB   ONE     Subtract 1    A = A-ONE
STA   BETA    Store in Beta
LDA   GAMMA   Load GAMMA into Register A
ADD   INCR    Add the value of INCR A = A+INCR
SUB   ONE     Subtract 1
STA   DELTA   Store in Delta

ONE   WORD  1   ONE WORD CONSTANT
                ONE-WORD VARIABLES
ALPHA RESW  1
BETA  RESW
GAMMA RESW
DELTA RESW
INCR  RESW
```

- Write a sequence of instructions for SIC to set ALPHA = BETA * 9 + GAMMA

TD: Test Device

```
LDA   BETA
MUL   NINE
ADD   GAMMA
STA   ALPHA

NINE  WORD  9
BETA  RESW  1
GAMMA RESW  1
ALPHA RESW  1
```



Program 3: Input-Output Operation

cc := denotes device busy.
C: < denotes device available

```
INLOOP  TD   INDEV  TEST INPUT DEVICE
        JEQ  INLOOP LOOP UNTIL DEVICE IS READY
        RD   INDEV  READ ONE BYTE INTO REGISTER A
        STCH DATA   STORE BYTE THAT WAS READ
        .
        .

OUTLP   TD   OUTDEV TEST OUTPUT DEVICE
        JEQ  OUTLP  LOOP UNTIL DEVICE IS READY
        LDCH DATA   LOAD DATA BYTE INTO REGISTER A
        WD   OUTDEV WRITE ONE BYTE TO OUTPUT DEVICE
        .
        .

INDEV   BYTE X'F1'  INPUT DEVICE NUMBER
OUTDEV  BYTE X'05'  OUTPUT DEVICE NUMBER
DATA    RESB 1      ONE-BYTE VARIABLE
```

looping

LDCH - Load Accumulator with character into the location.

Program 4 : Looping and Indexing

Index addressing mode.

```
        LDX   ZERO     INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH  STR1,X   LOAD CHARACTER FROM STR1 INTO REG A
        STCH  STR2,X   STORE CHARACTER INTO STR2
        TIX   ELEVEN   ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT   MOVECH   LOOP IF INDEX IS LESS THAN 11
        .
        .
        .
STR1    BYTE  C'TEST STRING'   11-BYTE STRING CONSTANT
STR2    RESB  11               11-BYTE VARIABLE
                               ONE WORD CONSTANTS
ZERO    WORD  0
ELEVEN  WORD  11
```

STR1 1000	T	STR2 2000	
1001	E	2001	
1002	S	2002	X 0
1003	T	2003	
1004		2004	Str1 + X
1005	S	2005	= 1000 + 0
1006	T	2006	= 1006
1007	R	2007	next stage
1008	I	2008	1000 + 1
1009	N	2009	= 1004
100A	G	200A	
ZERO	0		
ELEVEN	11		

After incrementation

X 1

Str1 + X
1000 + 1 = 1001

Str2 + X
2000 + 1 = 2001

checking X = 11

TD
Busy Available
CC= CC<

TIX
X++ Compare

classmate
Date
Page
X & particular variable

Program 5: Subroutine Call and record Input Operations

```
        JSUB    READ        CALL READ SUBROUTINE
        .
        .                   SUBROUTINE TO READ 100BYTE-RECORD
READ    LDX     ZERO        INITIALIZE INDEX REGISTER TO 0
RLOOP   TD      INDEV       TEST INPUT DEVICE
        JEQ     RLOOP       LOOP IF DEVICE IS BUSY
        RD      INDEV       READ ONE BYTE INTO REGISTER A
        STCH    RECORD,X    STORE DATA BYTE INTO RECORD
        TIX     K100        ADD 1 TO INDEX AND COMPARE TO 100
        JLT     RLOOP       LOOP IF INEX IS LESS THAN 100
        RSUB                EXIT FROM SUBROUTINE
        .
INDEV   BYTE    X'F1'       INPUT DEVICE NUMBER
RECORD  RESB    100         100-BYTE BUFFER FOR INPUT
        .                   RECORD ONE-WORD CONSTANTS
ZERO    WORD    0
K100    WORD    100
```

INDEV	F1	
RECORD	✓	2000
		2001
		2002
		.
		20991
Zero	0	
K100	100	

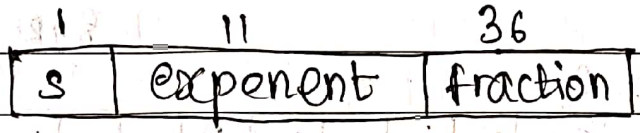
SIC/XE Machine Architecture

- Memory:
  - 8 bit bytes.
  - Any 3 consecutive bytes form a word (24 bits)
  - All addresses on SIC/XE are byte addresses.
  - Words are addressed by the location of their lowest numbered byte.
  - Maximum memory available on SIC/XE system is 1 megabyte (2^{20} bytes)
  - Results change in Instruction format and addressing modes.
- Registers:

Mnemonic	Number	Special use
A	0	Accumulator, used for arithmetic operations
X	1	Index register, used for addressing
L	2	Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register
B	3	Base register; used for addressing
S	4	General working register - no special use
T	5	General working register - no special use
F	6	Floating-point accumulator (48 bits)
PC	8	Program counter
SW	9	Status word, contains a variety of information including a condition code (CC).

- Data format ← Integer, character, floating point.
  - Integers:
    * stored as 24-bit binary numbers.
    * Negative numbers are stored as 2's complement representation.
  - characters:
    * Stored using their 8-bit ASCII codes.

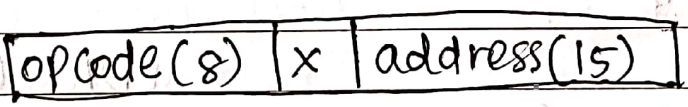
Additionally, there is a 48-bit floating-point data type



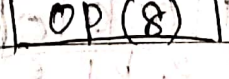
s → sign of floating point number
- If s=0, then it is a +ve number
- If s=1 then -ve number.

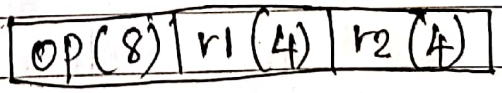
- A value of zero represented by setting all bits (including sign, exponent and fraction) as 0.

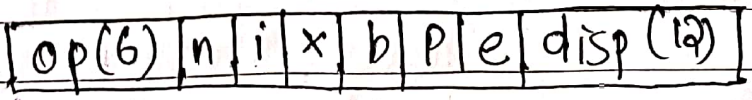
- Instruction format
  - In SIC → only 1 format (24 bit)



There are 4 formats of instructions available in SIC/XE

format 1 (1 byte) 

format 2 (2 byte) 

format 3 (3 byte) 

Format 4 (4 byte) 

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

op - 8 bits

• format 1 : RSUB → 4C

8 4 2 1 representation

0100	1100
8 bits	4 bits, 4 bits

• format 2 : COMPR A, S

compare registers A & S.

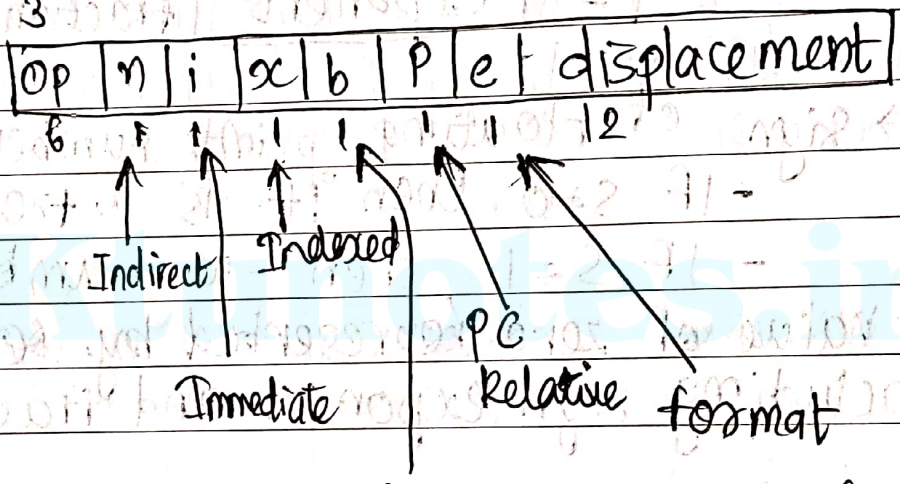
COMP ⇒ A0 (hexadecimal no)

A ⇒ 0

S ⇒ 4

1010	0000	0000	0100
8421	8421	842	8421

• Format 3



LDA #3

LDA = 00

8421 84

0000 00

000000	0	1	0	0	0	0000 0000 0011

n i x b p e

8421 8421 8421

0000 0000 0011

Format 4

8 bits additional (than format 3).

Addressing Modes

- b- and p-bit
- e-bit

Relative Addressing:

- Means we are specifying some address of the operand relating to some registers
- If specifying the address in relative to base register (B), then it is called base relative addressing mode.
- If specifying the address in relative to program counter (PC), then it is program counter relative addressing mode.

Mode	Indication	Target address Calculation	
Base relative	b=1, p=0	TA=(B)+disp	(0≤disp≤4095)
program-counter relative	b=0, p=1	TA=(PC)+disp	(-2048≤disp≤2047)

IF b=0, p=0, TA=disp

In format 4 instruction the bits p & b are normally set to 0 → then target address is taken from the address field of the instruction. This is known as Direct Addressing.

ADD ALPHA

A = A + ALPHA

- i and n bits: specify how target address is used.

Mode	Indication	Operand value
Immediate addressing	i=1, n=0	TA: TA is used as the operand value, no memory reference.
indirect addressing	i=0, n=1	((TA)): The word at the TA is fetched. Value of TA is taken as the address of the operand value
Simple addressing	i=0, n=0 i=1, n=1	Standard SIC (TA): TA is taken as the address of the operand value.

- Indexing cannot be used with immediate or indirect addressing modes.

- Special characters used.
  - # means immediate addressing
  - @ means Indirect addressing.

- Instruction set
  - Instructions for load & store registers.
    - LDA, STA, LDX, STX, LDL, STL
    - LDB, STB.

ADD
SUB
MUL
DIV

Instructions: Floating point arithmetic operations
: ADDF, SUBF, MULF, DIVF

RMO

Instructions that take operands from register
: RMO
→ Register Move

SVC → Supervisor call.
LPS → Load processor status

Instruction for register-to-register arithmetic operations

- ADDR, SUBR, MULR, DIVR

ADDR R_1, R_2 ; $R_2 = R_1 + R_2$

- Instruction COMP, COMP R

* Compares the value in register A with a word in memory and also between registers.
* This sets a condition code, CC to indicate the result (<, =, >)

- Conditional Jump Instructions

* JLT, JEQ, JGT

PROGRAM-1 : Data movement

```
LDA   #5     LOAD VALUE 5 INTO REGISTER A
STA   ALPHA  STORE IN ALPHA
LDA   #90    LOAD ASCII CODE FOR 'Z' INTO REGA
STCH  C1     STORE IN CHARACTER VARIABLE C1
.
.
ALPHA RESW 1  ONE WORD VARIABLE
C1    RESB 1  ONE BYTE VARIABLE
```

PROGRAM-2 : Arithmetic Operation

```
LDS   INCR   LOAD VALUE OF INCR INTO REGISTERS
LDA   ALPHA  LOAD ALPHA INTO REGISTER A
ADDR  S A    ADD VALUE OF INCR
SUB   #1     SUBTRACT 1
STA   BETA   STORE IN BETA
LDA   GAMMA  LOAD GAMMA INTO REGISTER A
```

```
        ADDR    S,A     ADD THE VALUE OF INCR
        SUB     #1      SUBTRACT 1
        STA     DELTA   STORE IN DELTA
        .
        .
        .
```

ONE WORD VARIABLE

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

BETA = ALPHA + INCR - 1
DELTA = GAMMA + INCR - 1

```
BS TF   A   Alpha       ADDR    S,A
                        A = S + A
        S   INCR        A = A - 1
```

Program 3 : Looping & Indexing

```
        LDT     #11     INITIALIZE REGISTER T TO 11
        LDX     #0      INITIALIZE INDEX REGISTER TO 0
MOVECH  LDCH    STR1,X  LOAD CHARACTER FROM STR1 INTO REGA
        STCH    STR2,X  STORE CHARACTER INTO STR2
        TIXR    T       ADD 1 TO INDEX, COMPARE RESULT TO 11
        JLT     MOVECH  LOOP IF INDEX IS LESS THAN 11
        .
        .
STR1    BYTE    C'TEST STRING'  11-BYTE STRING CONSTANT
STR2    RESB    11              11-BYTE VARIABLE
```

Program 4: Indexing and looping

```
        LDS    #3      INITIALISE REGISTER S TO 3
        LDT    #300    INITIALISE REGISTER T TO 300
        LDX    #0      INITIALISE INDEX REGISTER TO 0
ADDLP   LDA    ALPHA,X LOAD WORD FROM ALPHA INTO REGISTER A
        ADD    BETA,X  ADD WORD FROM BETA
        STA    GAMMA,X STORE THE RESULT IN A WORD IN GAMMA
        ADDR   S,X     ADD 3 TO INDEX VALUE
        COMPR  X,T     COMPARE NEW INDEX VALUE TO 300
        JLT    ADDLP   LOOP IF INDEX VALUE IS LESS THAN 300
        .
        .
        .              ARRAY VARIABLES-100 WORDS
                       EACH
ALPHA   RESW   100
BETA    RESW   100
GAMMA   RESW   100
```

S: 3
T: 300
X: 0

Program 5: Subroutine Call & record input operation.

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