SIC

- SIC & SIC/XE Architecture
- Addressing modes
- SIC & SIC/XE Instruction set
- Assembler Directives & Programming

$\frac{n}{205} \times 100 = 20$

SIC

- It a hypothetical computer (Model, Abstraction / simulator)
- Introduced in Beck textbook.
- Include h/w features found on real m/c.
- Avoid unusual / irrelevant Complexities.
- Abstract complex behaviors in real s/m.

} SIC provides a simplified view of s/m hardware from perspective of s/m pgmmer

SIC has 2 versions

↳ Standard version

↳ Extended version. (SIX/XE)

(Extra equipment, Extra expensive)

SIC M/c Architecture

I) Memory

- Consist of 8 bit bytes
- 15 bit address
- 3 consequtive bytes (24bits) form word
- Total 32768 (2^{15}) bytes in mly

II) Registers

- Five Registers, each 24 bit in length
- Both Numeric & Character representation 4 representing register

Mnemonic	Number	Special use
A	0	Accumulator (4 calculation)
X	1	Index register (4 addr calc)
L	2	Linkage register (Subroutine Linkage)
PC	8	Program Counter (store addr of next instn)
SW	9	status word. (store carry or Overflow flag).

Accumulator(0) $\longrightarrow$ Arithmetic operations

X (1) $\longrightarrow$ Store and calculate address (Offset)
index Register

L(2) $\longrightarrow$ used to jump to specific mly address & Store return address
Linkage register

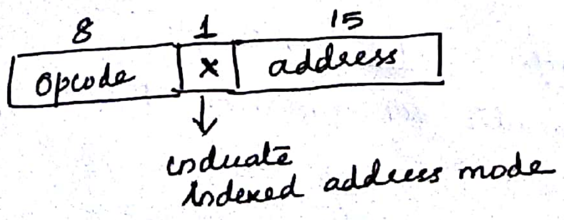
PC(8) $\longrightarrow$ store addr of next instn to execute.
Program Counter

SW (9) $\rightarrow$ Carry or over flow flag are stored.
Status word

DATA FORMATS

- Integers stored as 24-bit binary number.
- Negative number — 2's complement represent
- Characters — 8 bit ASCII codes.
- No floating point supported in standard version

Instn format



Addressing modes $\downarrow$ The way in which Operands are Specified in an Instn.

Addressing Modes

- 2 addressing modes
- Indicated by x bit in the instn.

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

(x) → Content of register X

Instn set →

- Input & output: RD, WD } TD Ready / Test Device (If TD (cc='<') = Device Ready; cc ⇒ '=' means device not ready)
- Subroutine linkage } JSUB / RSUB } Return address in Register L; Returns by jumping to address contained in L
- Load/store registers
- Integer arithmetic
- COMP
- Conditional jump instn

Load/store registers

- LDA
- LDX
- STA
- STX
- LDCH
- STCH

Integer arithmetic

- ADD
- MUL
- SUB
- DIV

} Involves register A & word in m/y

COMP

- COMP;
  ↳ compare value in register A with word in m/y
  ↳ Condition code (saved in SW)
  cc (<, =, >)

Conditional jump instn

- JLT
- JEQ
- JGT

} test cc & jump

LDA → Load value of memory to Accumulator

Eg LDA ALPHA ; Load value of ALPHA to accumulator

→ Eg COMP ALPHA
⇕
cc ← (>, =, <) of A ? ALPHA

} Based on the value of cc (<, =, >) jump operation is done

LDX BETA → Load value of BETA to X-register

STA GAMMA → Store value of A to GAMMA
(A = accumulator)

STX DELTA → Store value of X to DELTA

Input & Output → Perform by transferring 1 byte at a time to or from rightmost 8 bits of register A
Each device is assigned a unique 8 bit code as operand of I/O instructions

Assembler Directives

→ Instruct Assembler to perform certain action during assembly of pgm

→ Not translated into m/c instn instead provide instn to assembler itself.

→ Examples

- START

  Specify name and starting address of pgm

  E.g COPY START 1000

- END

  Indicate the end of source pgm and optionally specify the 1st executable instn in the pgm.

  E.g END FIRST

- BYTE

  Generate character or hexadecimal constant occupying as many bytes needed

  E.g EOF BYTE C'EOF' // character

  INPUT BYTE X'F1' // hexadecimal constant

- WORD

  Generate one word integer constant

  E.g THREE WORD 3

- RESB

  Reserve number of bytes for data area

  E.g BUFFER RESB 4096 // Reserve 4096 bytes

- RESW

  Reserve indicated number of words for a data area

  Eg RETA RESW 1 // Reserve 1 word.

SIC PROGRAMMING

DATA MOVEMENT OPERATION

```
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-BYTE CONSTANT
C1     RESB  1        ONE-BYTE VARIABLE
```

SAMPLE ARITHMETIC OPERATION FOR SIC

```
LDA    ALPHA     // LOAD ALPHA INTO REGISTER A
ADD    INCR      // ADD THE VALUE OF INCR
SUB    ONE       // SUBTRACT 1
STA    BETA      // STORE IN BETA
LDA    GAMMA     // LOAD GAMMA INTO REGISTER A
ADD    INCR      // ADD VALUE OF INCR
SUB    ONE       // SUBTRACT 1
STA    DELTA     // STORE IN DELTA
        .
        .
        .
ONE    WORD  1        // ONE WORD -CONSTANT
                         ONE WORD VARIABLE
ALPHA  RESW  1
BETA   RESW  1
GAMMA  RESW  1
DELTA  RESW  1        INCR  RESW  1
```

SAMPLE LOOPING & INDEXIN OPERATION IN SIC

```
        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
```

Sample Indexing & looping Instn

```
ADDLP   LDX    INDEX
        LDA    ALPHA
        ADD    BETA
        STA    GAMMA
          :
        COMP   K300
        JLT    ADDLP
INDEX   RESW   1
ALPHA   RESW   1
BETA    "      "
GAMMA   "      "
K300    WORD   300
```

Sample Input & output opern

```
INLOOP  TD     INDEV      // TEST INPUT DEVICE
        JEQ    INLOOP     // Loop untill device ready
        RD     INDEV      // Read 1 byte into register A
        STCH   DATA
          :
OUTLP   TD     OUTDEV     // Test output device
        JEQ    OUTLP      // Loop untill device Read
        LDCH   DATA       // Load data into reg A
        WD     OUTDEV     // Write one byte to output device
INDEV   BYTE   X'F1'      // input device number
OUTDEV  BYTE   X'05       // Output device no
DATA    RESB   1          // one byte variable
```