CC410: System Programming
Lecture 3: SIC/XE

By: Dr. Wael Hosny

SIC/XE machine architecture

- Memory
  - Maximum memory available on a SIC/XE system is 1 megabyte (2^{20} bytes)
  - Instruction format and addressing modes are changed
- Register (Additional registers)

Mnemonic	Number	Special use
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 (48bits)

  - Registers S and T are only for storing data. They can not use for accumulator
    - Ex: ADDR S, A A ← A+S
      COMPR X, T

Registers

Register	Purpose of Register (*each is 24 bits, except F)
(0) A	**Accumulator:** arithmetic, I/O
(1) X	**Index:** addressing
(2) L	**Linkage:** storing return addresses for subroutines (e.g. for JSUB)
(3) B	**Base:** addressing
(4) S	General working register
(5) T	General working register
(6) F	**Floating Point Accumulator** (48 bits)
(8) PC	**Program Counter:** address of the next instruction to be fetched for execution
(9) SW	**State Word:** information, including condition code (CC) used for tests (e.g. for I/O)

SIC/XE only

SIC/XE Machine Architecture

Data Formats

- Integer

  5 = 000000000000000000000101

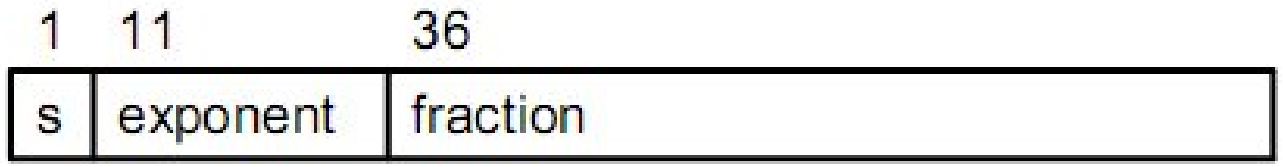
  -5 = 111111111111111111111011

- Character

  A 01000001

SIC/XE machine architecture

- Data formats
  - There is a 48-bit floating-point data type

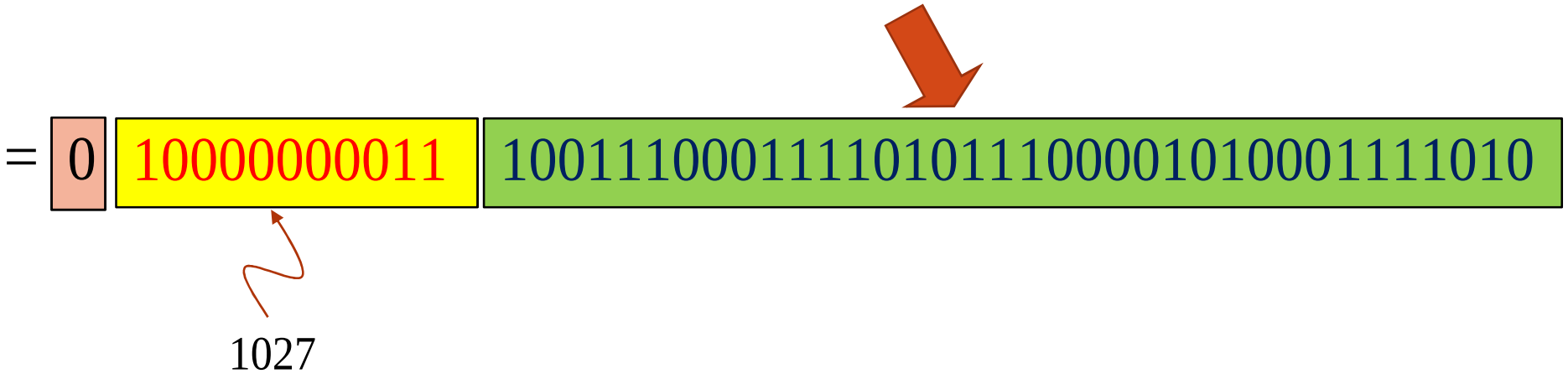


  - sign bit s (0: +, 1: -)
  - fraction f: a value between 0 and 1
  - exponent e: unsigned binary number between 0 and 2047
  - value: $s * f * 2^{(e-1024)}$
  - Ex: $5 = 2^2+2^0=(2^{-1}+2^{-3})*2^3= (2^{-1}+2^{-3})*2^{1027-1024}$

    0,10000000011,1010000….0

Data Formats Example

- Float
- 4.89 = 100.11100011110101110000101000111101011100001010 0
- 4.89 = .10011100011110101110000101000111101011100001010 0 * 2^3



Data Formats Example

- Float

  -.000489

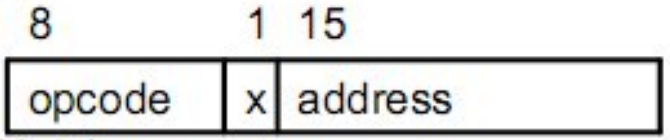
$= 10000000001100000011110000000011111110 * 2^{-10}$

$= 10000000001100000011110000000011111110 * 2^{1014-1024}$

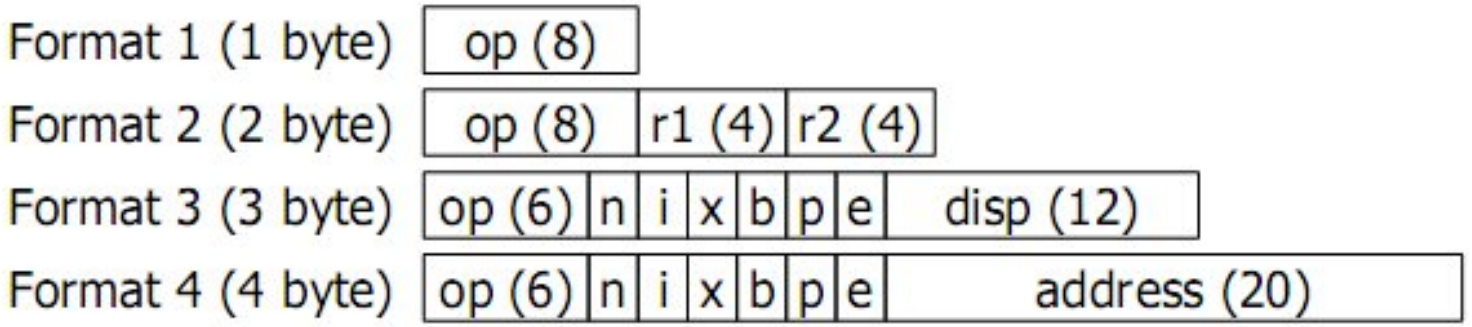
=1, 01111110110, 10000000001100000011110000000011111110

SIC/XE machine architecture

- Instruction formats
  - Since the memory used by SIC/XE may be 2^{20} bytes, the instruction format of SIC is not enough.
    - Solutions
      - Use relative addressing
      - Extend the address field to 20 bits



  - SIC/XE instruction formats



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

SIC/XE machine architecture

- Addressing modes
  - New relative addressing modes for format 3

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)

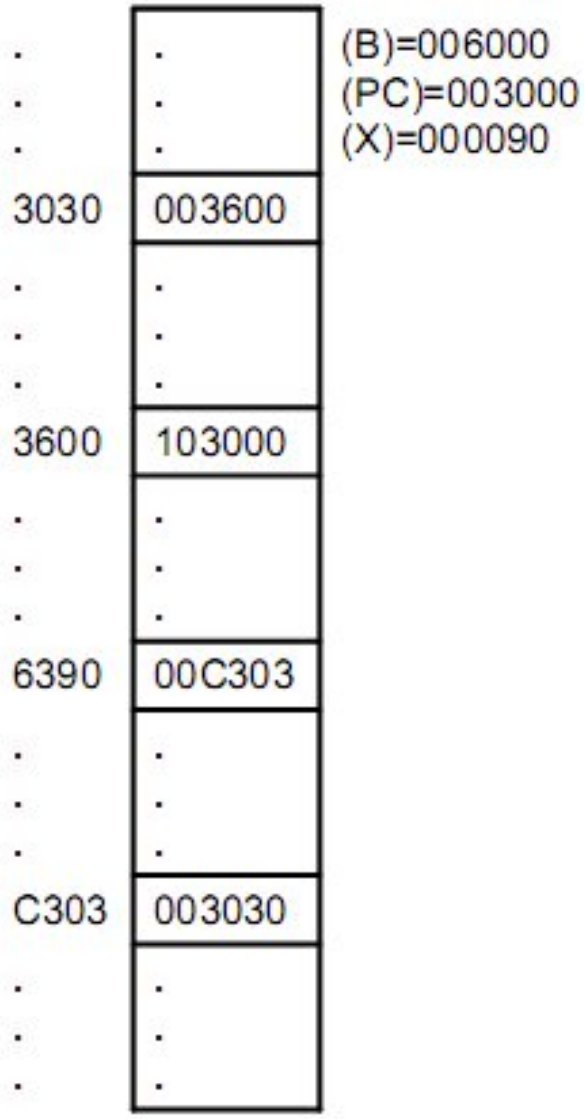
  - When base relative mode is used, disp is a 12-bits unsigned integer
  - When program-counter relative mode is used, disp is a 12-bits signed integer
    - 2's complement
  - Direct addressing for formats 3 and 4 if b=p=0
  - These two addressing mode can combine with index addressing if x=1

SIC/XE machine architecture

- Addressing modes
  - Bits x,b,p,e: how to calculate the target address
    - relative, direct, and indexed addressing modes
  - Bits i and n: how to use the target address (TA)

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

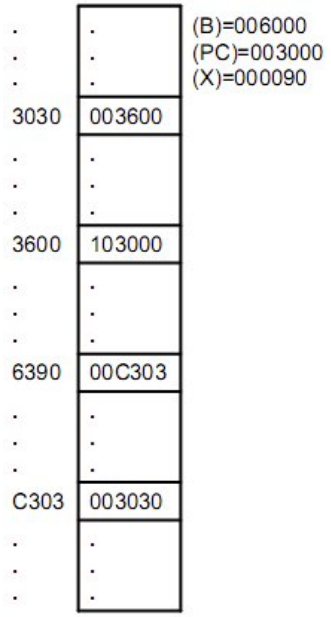
Addressing mode example



Addressing mode example

B=006000
PC=003000
X=000090

LDA's opcode is 00H



Hex	Machine instruction Binary								Target address	Value loaded into register A
	op	n	i	x	b	p	e	disp/address		
032600	000000	1	1	0	0	1	0	0110 0000 0000	3600	103000
03C300	000000	1	1	1	1	0	0	0011 0000 0000	6390	00C303
022030	000000	1	0	0	0	1	0	0000 0011 0000	3030	103000
010030	000000	0	1	0	0	0	0	0000 0011 0000	30	000030
003600	000000	0	0	0	0	1	1	0110 0000 0000	3600	103000
0310C303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

Case 1:

- n=1; i=1
  - Simple addressing: TA is taken as address of operand
- x=0
  - Not indexed (X has no influence on TA)
- b=0; p=1:
  - Program counter relative: therefore TA=PC +(disp)=3000 (PC)+600 (disp)=**3600** since it is indirect address, then the value stored in register A is **103000H**
- e=0:
  - Format 3

Case 2:

- n=1, i=1
  - Simple addressing: TA is taken as address of operand value
- x=1:
  - Indexed
- b=1, p=0:
  - Base relative addressing: therefore TA=B+(disp) = 6000 (B) +300 (disp)+90 (index)=**6390**, then the value stored in register A is **00C303H**
- e=0:
  - Format 3

Case 3:

- n=1, i=0
  - Indirect addressing: Word at TA is fetched. Value of TA is taken as address of operand value
- x=0:
  - Not indexed (X has no influence on TA)
- b=0; p=1:
  - Program counter relative: therefore TA=PC +(disp)=3000 (PC)+30 (disp)=3030 since it is indirect address, then the value stored in register A is **103000H** since 3030 has a value of 3600 which is address of the operand so look for **3600** will find data of **103000H**
- e=0:
  - Format 3

Case 4:

- n=0, i=1:
  - Immediate addressing: TA is used as operand. No memory reference.
- x=0:
  - Not indexed (X has no influence on TA)
- b=0; p=0 :
  - Direct addressing : therefore TA= disp = **30**, then the value stored in register A is **30H**
- e=0:
  - Format 3

Case 5:

- n=0, i=0
  - Standard SIC. Then bits b, p and e are considered to be part of the address field of the instruction (rather than flags indicating addressing mode)

- This makes format 3 identical to the format used on the standard version of SIC providing the desired compatibility

- Now aggregate bits x, b, p and e to disp/address Now it will be the X bit + 15 bits of address therefore TA is **3600**. then the value stored in register A is **103000H**

Case 6:

- n=1, i=1 → Simple addressing. TA is taken as address of operand value

- Since n=1, i=1, then it is not standard SIC, now
- Since e=1, then it is format 4

- x=0: not indexed (x has no influence on TA)
- b=0; p=0 : Direct addressing : therefore TA= address= **0C303**, then the value stored in register A is **3030H**

Special symbols (SIC & SIC/XE)

- # : immediate addressing
- @ : indirect addressing
- + : format 4
- * : the current value of PC
- C` ‘: character string
- op m, x : x denotes the index addressing

Note

- c: constant between 0 and 4095
- m: memory address or constant larger than 4095
- 4: format 4 instruction
- D: Direct Addressing instruction (p=0, b=0)
- A: Assembler selects either program counter relative (p=1, b=0) or base relative mode (p=0, b=1)
- S: Compatible with instruction format for standard SIC machine. Operand value can be between 0 and 32,767

Addressing mode summary

Note A: Assembler determines whether access is PC or B-relative

Addressing type	Flag bits n i x b p e	Assembler language notation	Calculation of target address TA	Operand	Notes
Simple	1 1 0 0 0 0	op c	disp	(TA)	D
n,i = 0,0 n,i = 1,1	1 1 0 0 0 1	+op m	addr	(TA)	4 D
	1 1 0 0 1 0	op m	(PC) + disp	(TA)	A
	1 1 0 1 0 0	op m	(B) + disp	(TA)	A
Indexed: x =1	1 1 1 0 0 0	op c,X	disp + (X)	(TA)	D
	1 1 1 0 0 1	+op m,X	addr + (X)	(TA)	4 D
	1 1 1 0 1 0	op m,X	(PC) + disp + (X)	(TA)	A
	1 1 1 1 0 0	op m,X	(B) + disp + (X)	(TA)	A
SIC	0 0 0 - - -	op m	b/p/e/disp	(TA)	D S
	0 0 1 - - -	op m,X	b/p/e/disp + (X)	(TA)	D S

Note A: Assembler determines whether access is PC or B-relative

Addressing type	Flag bits n i x b p e	Assembler language notation	Calculation of target address TA	Operand	Notes
Indirect	1 0 0 0 0 0	op @c	disp	((TA))	D
n,i = 1,0	1 0 0 0 0 1	+op @m	addr	((TA))	4 D
	1 0 0 0 1 0	op @m	(PC) + disp	((TA))	A
	1 0 0 1 0 0	op @m	(B) + disp	((TA))	A
Immediate	0 1 0 0 0 0	op #c	disp	TA	D
n,i = 0,1	0 1 0 0 0 1	+op #m	addr	TA	4 D
	0 1 0 0 1 0	op #m	(PC) + disp	TA	A
	0 1 0 1 0 0	op #m	(B) + disp	TA	A

Note: Cannot use indexing with Indirect, Immediate Modes

Note2: Cannot use b or p relative addressing with Format 4 (e bit): direct addressing only

Addressing mode summary

Addressing type	Flag bits n i x b p e	Assembler lenguage notation	Calculation of target address TA	Operand	Notes
Simple	1 1 0 0 0 0	op c	disp	(TA)	D
	1 1 0 0 0 1	+op m	addr	(TA)	4 D
	1 1 0 0 1 0	op m	(PC)+disp	(TA)	A
	1 1 0 1 0 0	op m	(B)+disp	(TA)	A
	1 1 1 0 0 0	op c,X	disp+(X)	(TA)	D
	1 1 1 0 0 1	+op m,X	addr+(X)	(TA)	4 D
	1 1 1 0 1 0	op m,X	(PC)+disp+(X)	(TA)	A
	1 1 1 1 0 0	op m,X	(B)+disp+(X)	(TA)	A
	0 0 0 - - -	op m	b/p/e/disp	(TA)	D S
	0 0 1 - - -	op m,X	b/p/e/disp+(X)	(TA)	D S
Indirect	1 0 0 0 0 0	op @c	disp	((TA))	D
	1 0 0 0 0 1	+op @m	addr	((TA))	4 D
	1 0 0 0 1 0	op @m	(PC)+disp	((TA))	A
	1 0 0 1 0 0	op @m	(B)+disp	((TA))	A
Immediate	0 1 0 0 0 0	op #c	disp	TA	D
	0 1 0 0 0 1	+op #m	addr	TA	4 D
	0 1 0 0 1 0	op #m	(PC)+disp	TA	A
	0 1 0 1 0 0	op #m	(B)+disp	TA	A

SIC/XE machine architecture

- Instruction set
  - Standard SIC's instruction
  - Load and store registers (B, S, T, F)
    - `LDB, STB, …`
  - Floating-point arithmetic operations
    - `ADDF, SUBF, MULF, DIVF`
  - Register-register arithmetic operations
    - `ADDR, SUBR, MULR, DIVR`
  - Register move operations
    - `RMO`
  - Supervisor call (SVC)
    - `generates an interrupt for OS (Chap 6)`
- Input/Output
  - SIO, TIO, HIO: start, test, halt the operation of I/O device

SIC/XE machine architecture

- Instruction set
  - Refer to Appendix A for all instructions (Page 496)
  - Notations for appendix
    - A ← (m..m+2): move word begin at m to A
    - P: privileged instruction
    - X: instruction available only in SIC/XE
    - C: condition code CC

Programming examples (SIC/XE)
- Data movement

- Page 13, Figure 1.2 (b)

```
        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 CHARACTER VARIABLE C1
        .
        .
ALPHA   RESW    1       ONE-WORD VARIABLE
C1      RESB    1       ONE-BYTE VARIABLE
```

Programming examples (SIC/XE)
- Arithmetic

- Page 15, Figure 1.3 (b)

```
        LDS     INCR    LOAD VALUE OF INCR INTO REGISTER S
        LDA     ALPHA   LOAD ALPHA INTO REGISTER A
        ADDR    S,A     ADD THE 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 VARIABLES
ALPHA   RESW    1
BETA    RESW    1
GAMMA   RESW    1
DELTA   RESW    1
INCR    RESW    1
```

Programming examples (SIC/XE)
-Looping and indexing

- Page 16, Figure 1.4 (b)

```
MOVECH   LDT     #11              INITIALIZE REGISTER T TO 11
         LDX     #0               INITIALIZE INDEX REGISTER TO 0
         LDCH    STR1,X           LOAD CHARACTER FROM STR1 INTO REG A
         STCH    STR2,X           SOTRE 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
```

Programming examples (SIC/XE) - Indexing and looping

- Page 17, Figure 1.5 (b)

```
        LDS     #3          INITIALIZE REGISTER S TO 3
        LDT     #300        INITIALIZE REGISTER T TO 300
        LDX     #0          INITIALIZE 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
```

Programming examples (SIC/XE)
- Subroutine call and record input

- Page 20, Figure 1.7 (a)

```
            JSUB    READ        CALL READ SUBROUTINE
            .
            .
.                               SUBROUTINE TO READ 100-BYTE RECORD
READ        LDX     #0          INITIALIZE INDEX REGISTER TO 0
RLOOP       LDT     #100        INITIALIZE REGISTER T TO 100
            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 SUBROUTINE
            .
            .
INDEV       BYTE    X'F1'       INPUT DEVICE NUMBER
RECORD      RESB    100         100-BYTE BUFFER FOR INPUT RECORD
```