# Chapter 1 – Background

## 1.1 Introduction

- *System software* consists of a variety of programs that support the operation of a computer – *Text editor*, *Compiler*, *Loader* or *Linker*, *Debugger*, *Assembler*, *Marco processor*, *Operating system*, *etc*.
- The major topics of this course – assemblers, loaders and linkers, macro processors, compilers, and operating systems. The other topics including *database management systems*, *text editors*, and *interactive debugging systems* are mentioned in Chapter 7.

## 1.2 System Software and Machine Architecture

- One characteristic in which most system software differs from applications software is *machine dependency*.
- System programs are intended to support the operation and use of the computer itself, rather than any particular application. For this reason, they are usually related to the architecture of the machine on which they are to run.
- Because most system software is machine-dependent, we must include real machines and real piece of software in our study. We will present the fundamental functions of each piece of software based on a Simplified Instructional Computer (SIC) – a hypothetical computer.

## 1.3 The Simplified Instructional Computer (SIC)

- In this section, we describe the architecture of SIC.
- SIC comes in two versions: the *standard model* and an

*XE version* (XE stands for “extra equipment” or “extra expensive”).

### 1.3.1 SIC Machine Architecture

#### *Memory*

- Memory consists of 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.
- There are a total of 32,768 ($2^{15}$) bytes in SIC memory.

#### *Register*

- There are five registers, all of which have special uses.
- Each register is 24 bits in length.
- See table at the bottom of Page 5.

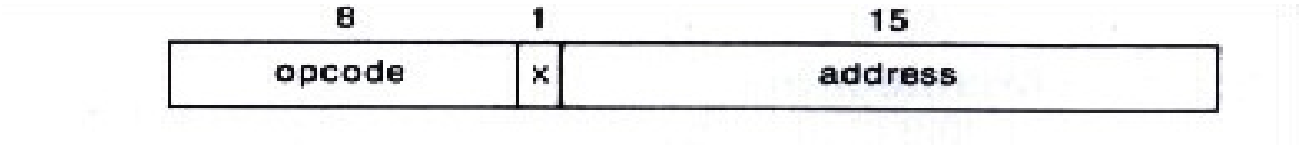
| Mnemonic | Number | Special use |
|---|---|---|
| A | 0 | Accumulator; used for arithmetic operations |
| X | 1 | Index register; used for addressing |
| L | 2 | Linkage register; the Jump to Subroutine (JSUB) instruction stores the return address in this register |
| PC | 8 | Program counter; contains the address of the next instruction to be fetched for execution |
| SW | 9 | Status word; contains a variety of information, including a Condition Code (CC) |

#### *Data Formats*

- Integers are stored as 24-bit binary numbers; 2’s complement representation is used for negative values.
- No floating-point hardware on the standard version of SIC.

#### *Instruction Formats*

- See Page 6.




## *Addressing Modes*

- Two addressing modes available; see Page 6.

| Mode | Indication | Target address calculation |
|---|---|---|
| Direct | x = 0 | TA = address |
| Indexed | x = 1 | TA = address + (X) |

## *Instruction Set*

- SIC provides a basic set of instructions that are sufficient for most simple tasks. (See Appendix A!)
- Instructions that load and store registers (LDA, LDX, STA, STX, etc.).
- Instructions for integer arithmetic operations (ADD, SUB, MUL, DIV). All arithmetic operations involve register A and a word in memory.
- An instruction (COMP) that compares the value in register A with a word in memory; this instruction sets a condition code CC to indicate the result (<, =, or >).
- Conditional jump instructions (JLT, JEQ, JGT) can test the setting of CC, and jump accordingly.
- JSUB and RSUB are provided to subroutine linkage.

## *Input and Output*

- On the standard version of SIC, 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.

- Three I/O instructions: TD (Test Device), RD (Read Data), WD (Write Data).

## 1.3.2 SIC/XE Machine Architecture

### *Memory*

- The memory structure for SIC/XE is the same as SIC.
- Maximum memory on a SIC/XE is 1 Mbyte.

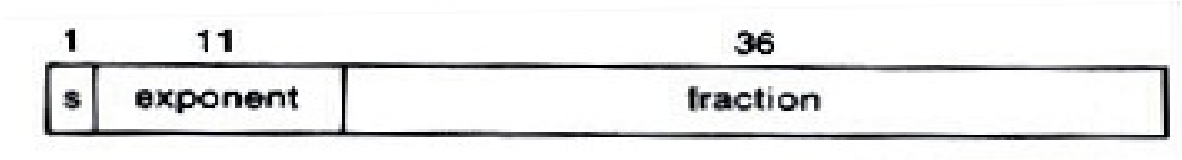
### *Registers*

- Additional registers are provided by SIC/XE and shown at the bottom of Page 7.

| 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 (48 bits) |

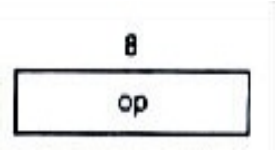
### *Data Formats*

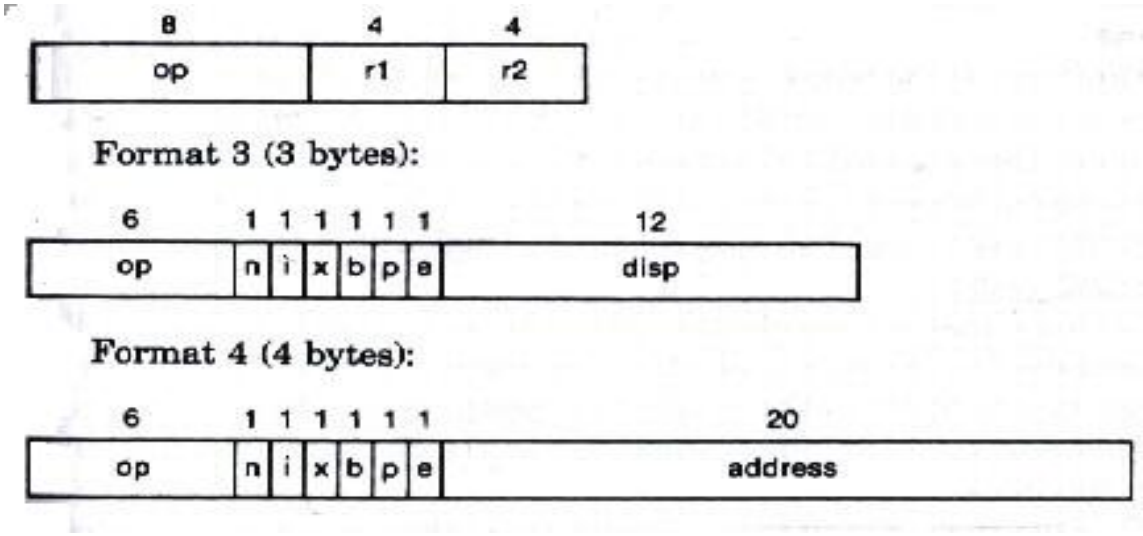
- SIC/XE provides the same data formats as SIC.
- In addition, a 48-bit floating-point data type is also provided. See the top of Page 8.




### *Instruction Formats*

- In addition to the instruction format of SIC, the new set of instruction formats for SIC/XE is shown in Page 8~9.






- The settings of the flag bits:

  1) Bit *e*: to distinguish between Formats 3 and 4. (*e*=0 → Format 3, *e*=1 → Format 4)

## *Addressing Modes*

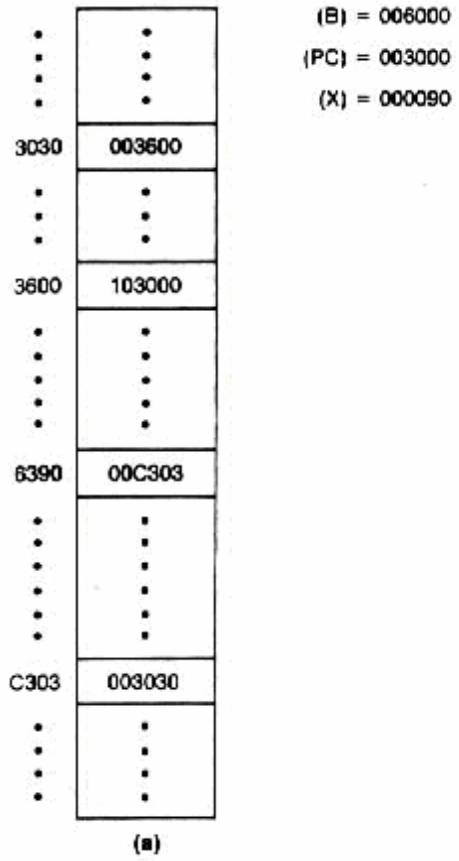
- Two new *relative addressing modes* (Base relative, Program- counter relative) are available for use with instructions assembled using Format 3. See Page 9.

| 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) |

- *Direct addressing mode*: bits *b* and *p* are both set to 0 to Formats 3 and 4.
- *Indexing addressing mode*: if bit *x* is set to 1, the term (X) is added in the target address calculation (Formats 3 and 4).
- *Immediate addressing mode*: if bits *i*=1 and *n*=0, the target address itself is used as the operand value; no memory reference is performed.
- *Indirect addressing mode*: if bits *i*=0 and *n*=1, the word at

the location given by the target address is fetched; the *value* contained in this word is taken as the *address* of the operand value.

- *Simple addressing mode*: if bits *i* and *n* are both 0 or both 1, the target address is taken as the location of the operand.
- Fig 1.1 gives examples of the different addressing modes available on SIC/XC.




(a)

| | Machine instruction | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Hex | Binary | | | | | | | | Value loaded into register A |
| | op | n | i | x | b | p | e | disp/address | Target 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 |

(b)

**Figure 1.1** Examples of SIC/XE instructions and addressing modes.

## *Instruction Set*

- SIC/XE provides all of the instructions available on SIC.
- In addition, there are instructions to load and store the new registers (LDB, STB, etc) and to perform floating-point arithmetic operations (ADDF, SUBF, MULF, DIVF).
- Register-to-register arithmetic operations: ADDR, SUBR, MULR, DIVR.
- Supervisor call instruction: SVC.

## *Input and Output*

- The I/O instructions for SIC are also available on SIC/XE.

## 1.3.3 SIC Programming Examples

- Fig 1.2 contains examples of data movement operations for SIC and SIC/XE. See Page 13.

```
        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
```

(a)

```
        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
```

(b)

**Figure 1.2** Sample data movement operations for (a) SIC and (b) SIC/XE.

- Fig 1.3: Sample arithmetic operations for (a) SIC and (b) SIC/XE. See Page 15.

```
            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 THE VALUE OF INCR
            SUB     ONE         SUBTRACT 1
            STA     DELTA       STORE IN DELTA
              .
              .
              .
ONE         WORD    1           ONE-WORD CONSTANT
.                               ONE-WORD VARIABLES
ALPHA       RESW    1
BETA        RESW    1
GAMMA       RESW    1
DELTA       RESW    1
INCR        RESW    1
```

(a)

```
            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
```

(b)

**Figure 1.3** Sample arithmetic operations for (a) SIC and (b) SIC/XE.

- Fig 1.4: Sample looping and indexing operations for (a) SIC and (b) SIC/XE. See Page 16.

```
        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
```

(a)

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

(b)

**Figure 1.4** Sample looping and indexing operations for (a) SIC and (b) SIC/XE.

- Fig 1.5: Sample indexing and looping operations for (a) SIC and (b) SIC/XE. See Page 17.

```
            LDA     ZERO        INITIALIZE INDEX VALUE TO 0
            STA     INDEX
ADDLP       LDX     INDEX       LOAD INDEX VALUE INTO REGISTER X
            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
            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 VARIABLE FOR INDEX VALUE
.                               ARRAY VARIABLES--100 WORDS EACH
ALPHA       RESW    100
BETA        RESW    100
GAMMA       RESW    100
.                               ONE-WORD CONSTANTS
ZERO        WORD    0
K300        WORD    300
THREE       WORD    3
```

(a)

```
            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
```

(b)

**Figure 1.5** Sample indexing and looping operations for (a) SIC and (b) SIC/XE.

- Fig 1.6: Sample input and output operations for SIC. See Page 19.

```
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
```

**Figure 1.6** Sample input and output operations for SIC.

- Fig 1.7: Sample subroutine call and record input operations for (a) SIC and (b) SIC/XE. See Page 20.

```
        JSUB    READ        CALL READ SUBROUTINE
        .
        .
        .
.                           SUBROUTINE TO READ 100-BYTE 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 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
.                           ONE-WORD CONSTANTS
ZERO    WORD    0
K100    WORD    100
```

(a)

```
        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 SUBROUTINE
        .
        .
        .
INDEV   BYTE    X'F1'       INPUT DEVICE NUMBER
RECORD  RESB    100         100-BYTE BUFFER FOR INPUT RECORD
```

(b)

**Figure 1.7** Sample subroutine call and record input operations for (a) SIC and (b) SIC/XE.

## 1.4 Traditional (CISC) Machines

- The machines described in this section are classified as Complex Instruction Set Computers (CISC).
- CISC machines generally have a relatively *large and complicated instruction set*, several different *instruction formats* and *lengths*, and many different *addressing modes*.
- The implementation of such architecture in hardware tends to be complex.

## 1.5 RISC Machines

- The RISC (Reduced Instruction Set Computers) concept, developed in the early 1980s, was intended to simplify the design of processors.
- This simplified design can result in faster and less expensive processor development, greater reliability, and faster instruction execution times.
- In general, a RISC system is characterized by a standard, fixed instruction length (usually equal to one machine word), and single-cycle execution of most instructions.