Subject Name: System Software

Subject Code: 10CS52

Prepared By: Ms. Sanchari Saha (Asst. Professor)

Department: Computer Science & Engineering

Date: 30.08.2014
UNIT-I

MACHINE ARCHITECTURE
Topic Details

1) Introduction
2) System Software and Machine Architecture
3) Simplified Instructional Computer (SIC)
4) SIC Machine Architecture
5) SIC/XE Machine Architecture
6) SIC Programming Examples
7) SIC/XE Programming Examples
8) Self check Exercise
9) Summary
Learning Outcome:

Students will understand:-

✓ The basic concept of system software
✓ Difference between System software & Application software
✓ SIC & SIC/XE machine architecture
✓ SIC & SIC/XE programming concept
1. Introduction
1.1 What is system software?

System software is a collection of variety of programs that supports the operations of a computer.

This makes it possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.

Examples:

text editor, compiler, assembler, OS, debugger, macro processor etc.
1.2 System Software VS Application Software

• The most important characteristic in which most system software differ from application software is machine dependency.

• System software--supports operation and use of computer. Application software - solution to a problem.

• An application program is primarily concerned with the solution to some problem, using computer as a tool.

• The focus is on the application, not on the application system. They are machine independent.

• System programs, on the other hand, are intended to support the operation and use of the computer itself, rather than any particular application.

• They are usually related to the architecture of the machine on which they are to run.
• **An application program** can not run by itself. It is dependent on System software to execute.

• **System software** have direct control & access to your computer hardware & memory location. They enforce various I/O operations on various memory locations & control the hardware to make the application software to do a task.

**Application software examples:** MySQL, Microsoft word, Opera (web browser) etc.

**System Software examples:** OS, Compiler, Assembler, Loader, Linker etc.
2. System Software & Machine Architecture
2.1 Machine Dependency

An **assembler** is a system software. It translates mnemonic instructions into machine code; the instruction formats, addressing modes, etc, are of direct concern in assembler design.

Similarly, **compilers** must generate machine language code, taking into account such hardware characteristics as the number and the types of registers and machine instructions available.

**Operating systems** are directly concerned with the management of nearly all of the resources of a computer system.

There are aspects of system software that **do not directly depend upon** the type of computing system, general design and logic of an assembler, general design and logic of a compiler and, code optimization techniques, which are independent of target machines.

Likewise, the process of linking together **independently** assembled subprograms does not usually depend on the computer being used.
When you took the first programming course

- **Text editor** - create and modify the program

- **Compiler** - translate programs into machine language

- **Loader or linker** - load machine language program into memory and prepared for execution

- **Debugger** - help detect errors in the program

➢ When you wrote programs in assembler language

- **Assembler** - translate assembly program into machine language

- **Macro processor** - translate macros instructions into its definition

you control all of these processes by interacting with **the OS**
3. SIC
(Simplified Instructional Computer)
3.1 SIC Definition:

SIC refers to Simplified Instruction Computer which is a hypothetical computer that has been designed to include the hardware features most often found on real machines, while avoiding unusual and irrelevant complexities.

This allows to clearly separate the central concepts of a system software from the implementation details associated with a particular machine.

**Important machine structures used in design of software:-**

- Memory structure
- Registers
- Data formats
- Instruction formats
- Addressing modes
- Instruction set
4. SIC Machine Architecture
4.1 Memory

There are 215 bytes in the computer memory, that is 32,768 bytes. Three consecutive bytes form a word. Each location in memory contains 8-bit bytes.

4.2 Registers

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Number</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>Accumulator; used for arithmetic operations</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>Index register; used for addressing</td>
</tr>
<tr>
<td>L</td>
<td>2</td>
<td>Linkage register; JSUB</td>
</tr>
<tr>
<td>PC</td>
<td>8</td>
<td>Program counter</td>
</tr>
<tr>
<td>SW</td>
<td>9</td>
<td>Status word, including CC</td>
</tr>
</tbody>
</table>
4.3 Data Formats

Integers are stored as 24-bit binary numbers, 2’s complement representation is used for negative values, characters are stored using their 8-bit ASCII codes, No floating-point hardware on the standard version of SIC.

4.4 Instruction Formats

<table>
<thead>
<tr>
<th>Opcode(8)</th>
<th>x</th>
<th>Address (15)</th>
</tr>
</thead>
</table>

4.5 Addressing Modes

<table>
<thead>
<tr>
<th>Mode</th>
<th>Indication</th>
<th>Target address calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>x = 0</td>
<td>TA = address</td>
</tr>
<tr>
<td>Indexed</td>
<td>x = 1</td>
<td>TA = address + (x)</td>
</tr>
</tbody>
</table>
4.6 Instruction Set

- **integer arithmetic operations**: ADD, SUB, MUL, DIV, etc.
  - All arithmetic operations involve register A and a word in memory, with the result being left in the register

- **comparison**: COMP
  - COMP compares the value in register A with a word in memory, this instruction sets a condition code CC to indicate the result

- **conditional jump instructions**: JLT, JEQ, JGT
  - these instructions test the setting of CC and jump accordingly

- **subroutine linkage**: JSUB, RSUB
  - JSUB jumps to the subroutine, placing the return address in register L
  - RSUB returns by jumping to the address contained in register L
4.7 Input and Output

- Input and output are performed by transferring 1 byte at a time to or from the rightmost 8 bits of register A.

- The Test Device (TD) instruction tests whether the addressed device is ready to send or receive a byte of data.

- Read Data (RD)

- Write Data (WD)
5. SIC/XE Machine Architecture
5.1 Memory:
Maximum memory available on a SIC/XE system is 1 Megabyte (220 bytes), i.e. $2^{20}$ bytes

5.2 Registers:
Additional B, S, T, and F registers are provided by SIC/XE, in addition to the registers of SIC

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Number</th>
<th>Special use</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>3</td>
<td>Base register</td>
</tr>
<tr>
<td>S</td>
<td>4</td>
<td>General working register</td>
</tr>
<tr>
<td>T</td>
<td>5</td>
<td>General working register</td>
</tr>
<tr>
<td>F</td>
<td>6</td>
<td>Floating-point accumulator (48 bits)</td>
</tr>
</tbody>
</table>
5.3 Floating-point data type:

There is a 48-bit floating-point data type, $F^*2(e^{-1024})$
5.4 Instruction Formats

The new set of instruction formats for SIC/XE machine architecture are as follows.

**Format 1 (1 byte):**
contains only operation code (straight from table).

**Format 2 (2 bytes):**
first eight bits for operation code, next four for register 1 and following four for register 2.

The numbers for the registers go according to the numbers indicated at the registers section (i.e, register T is replaced by hex 5, F is replaced by hex 6).
**Format 3 (3 bytes):**

First 6 bits contain operation code, next 6 bits contain flags, last 12 bits contain displacement for the address of the operand.

Operation code uses only 6 bits, thus the second hex digit will be affected by the values of the first two flags (n and i).

The flags, in order, are: n, i, x, b, p, and e. Its functionality is explained in the next section. The last flag e indicates the instruction format (0 for 3 and 1 for 4).

**Format 4 (4 bytes):**

same as format 3 with an extra 2 hex digits (8 bits) for addresses that require more than 12 bits to be represented.
Formats 1 and 2 are instructions do not reference memory at all.
Format 3 (3 bytes)

\[
\begin{array}{cccccccc}
6 & 1 & 1 & 1 & 1 & 1 & 1 & 12 \\
\text{op} & n & i & x & b & p & e & \text{disp}
\end{array}
\]

Format 4 (4 bytes)

\[
\begin{array}{cccccccc}
6 & 1 & 1 & 1 & 1 & 1 & 1 & 20 \\
\text{op} & n & i & x & b & p & e & \text{address}
\end{array}
\]
5.5 Addressing modes & Flag Bits

*Five possible addressing modes plus the combinations are as follows.*

**Direct** (x, b, and p all set to 0):

operand address goes as it is. n and i are both set to the same value, either 0 or 1.

While in general that value is 1, if set to 0 for format 3 we can assume that the rest of the flags (x, b, p, and e) are used as a part of the address of the operand, to make the format compatible to the SIC format

**Relative** (either b or p equal to 1 and the other one to 0):

the address of the operand should be added to the current value stored at the B register (if \( b = 1 \)) or to the value stored at the PC register (if \( p = 1 \))
**Immediate** \((i = 1, n = 0)\):

The operand value is already enclosed on the instruction (i.e. lies on the last 12/20 bits of the instruction.

**Indirect** \((i = 0, n = 1)\):

The operand value points to an address that holds the address for the operand value.

**Indexed** \((x = 1)\):

The value to be added to the value stored at the register \(x\) to obtain real address of the operand. This can be combined with any of the previous modes except immediate.
The various flag bits used have the following meanings

✓ e: e = 0 means format 3, e = 1 means format 4 Bits

✓ x,b,p: Used to calculate the target address using relative, direct, and indexed addressing Modes

Bits i and n: Says, how to use the target address

✓ b and p - both set to 0, displacement field from format 3 instruction is taken to be the target address.

Address filed from format 4 instruction is taken to be the target address

✓ If b=0, P=1, PC relative addressing, TA= PC+ Displacement
✓ If b=1, P=0, Base relative addressing, TA= B+ Displacement
✓ x - x is set to 1, X register value is added for target address calculation

✓ i=1, n=0 Immediate addressing,

TA: TA is used as the operand value, no memory reference

✓ i=0, n=1 Indirect addressing,

TA: The word at the TA is fetched. Value of TA is taken as the address of the operand value

✓ i=0, n=0 or i=1, n=1: Simple addressing,

TA: TA is taken as the address of the operand value.
5.6 Instruction Set:
SIC/XE provides all of the instructions that are available on the standard version.

In addition we have, Instructions to load and store the new registers: LDB, STB, etc.

Floating-point arithmetic operations: ADDF, SUBF, MULF, DIVF.

Register move instruction: RMO

Register-to-register arithmetic operations: ADDR, SUBR, MULR, DIVR

Supervisor call instruction: SVC.

5.7 Input and Output:
There are I/O channels that can be used to perform input and output while the CPU is executing other instructions. Allows overlap of computing and I/O, resulting in more efficient system operation. The instructions SIO, TIO, and HIO are used to start, test and halt the operation of I/O channels.
6. SIC Programming Examples
Example 1 (Simple data and character movement operation)

LDA FIVE
STA ALPHA
LDCH CHARZ
STCH C1

ALPHA RESW 1
FIVE WORD 5
CHARZ BYTE C’Z’
C1 RESB 1
Example 2 (Arithmetic operations)

LDA ALPHA
ADD INCR
SUB ONE
STA BEETA

.......
.......
.......

ONE     WORD  1
ALPHA   RESW  1
BEETA   RESW  1
INCR    RESW  1
Example 3 (Looping and Indexing operation)

LDX ZERO : X = 0
MOVECH LDCH STR1, X : LOAD A FROM STR1
STCH STR2, X : STORE A TO STR2
TIX ELEVEN : ADD 1 TO X, TEST
JLT MOVECH

STR1 BYTE C 'HELLO WORLD'
STR2 RESB 11
ZERO WORD 0
ELEVEN WORD 11
Example 4 (Input and Output operation)

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INLOOP</td>
<td>TD</td>
<td>INDEV</td>
<td>TEST INPUT DEVICE</td>
</tr>
<tr>
<td></td>
<td>JEQ</td>
<td>INLOOP</td>
<td>LOOP UNTIL DEVICE IS READY</td>
</tr>
<tr>
<td></td>
<td>RD</td>
<td>INDEV</td>
<td>READ ONE BYTE INTO A</td>
</tr>
<tr>
<td></td>
<td>STCH</td>
<td>DATA</td>
<td>STORE A TO DATA</td>
</tr>
<tr>
<td>OUTLP</td>
<td>TD</td>
<td>OUTDEV</td>
<td>TEST OUTPUT DEVICE</td>
</tr>
<tr>
<td></td>
<td>JEQ</td>
<td>OUTLP</td>
<td>LOOP UNTIL DEVICE IS READY</td>
</tr>
<tr>
<td></td>
<td>LDCH</td>
<td>DATA</td>
<td>LOAD DATA INTO A</td>
</tr>
<tr>
<td></td>
<td>WD</td>
<td>OUTDEV</td>
<td>WRITE A TO OUTPUT DEVICE</td>
</tr>
<tr>
<td>INDEV</td>
<td>BYTE</td>
<td>X 'F5'</td>
<td>INPUT DEVICE NUMBER</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>BYTE</td>
<td>X '08'</td>
<td>OUTPUT DEVICE NUMBER</td>
</tr>
<tr>
<td>DATA</td>
<td>RESB</td>
<td>1</td>
<td>ONE-BYTE VARIABLE</td>
</tr>
</tbody>
</table>
Example 5  (To transfer two hundred bytes of data from input device to memory)

LDX ZERO
CLOOP TD INDEV
JEQ CLOOP
RD INDEV
STCH RECORD, X
TIX B200
JLT CLOOP

INDEV BYTE X 'F5'
RECORD RESB 200
ZERO  WORD 0
B200  WORD 200
Example 6 (Subroutine to transfer two hundred bytes of data from input device to memory)

```
JSUB READ
.............
.............
READ       LDX ZERO
CLOOP      TD INDEV
           JEQ CLOOP
           RD INDEV
           STCH RECORD, X
           TIX B200   : add 1 to index compare 200 (B200)
           JLT CLOOP
           RSUB
.............
.............
INDEV     BYTE X 'F5'
RECORD    RESB 200
ZERO      WORD  0
B200      WORD 200
```
7. SIC/XE Programming Examples
Example 1 (Simple data and character movement operation)

LDA  #5
STA  ALPHA
LDA   #90
STCH    C1

ALPHA    RESW    1
C1        RESB    1
Example 2 (Arithmetic operations)

LDS  INCR
LDA  ALPHA
ADD  S,A
SUB  #1
STA  BEETA

...........

...........

ALPHA  RESW  1
BEETA  RESW  1
INCR   RESW  1
Example 3 (Looping and Indexing operation)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDT #11</td>
<td></td>
</tr>
<tr>
<td>LDX #0</td>
<td>$X = 0$</td>
</tr>
<tr>
<td>MOVECH</td>
<td></td>
</tr>
<tr>
<td>LDCH STR1, X</td>
<td>LOAD A FROM STR1</td>
</tr>
<tr>
<td>STCH STR2, X</td>
<td>STORE A TO STR2</td>
</tr>
<tr>
<td>TIXR T</td>
<td>ADD 1 TO X, TEST (T)</td>
</tr>
<tr>
<td>JLT MOVECH</td>
<td></td>
</tr>
</tbody>
</table>

```
STR1 BYTE C 'HELLO WORLD'
STR2 RESB 11
```
Example 4  (To transfer two hundred bytes of data from input device to memory)

LDT   #200
LDX   #0
CLOOP  TD   INDEV

JEQ   CLOOP
RD    INDEV
STCH  RECORD, X
TIXR  T
JLT   CLOOP
.
.
INDEV  BYTE  X ‘F5’
RECORD  RESB  200
Example 5  (Subroutine to transfer two hundred bytes of data from input device to memory)

```
JSUB  READ
       ........
       ........
READ   LDT  #200
       LDX  #0
CLOOP  TD   INDEV
       JEQ  CLOOP
       RD   INDEV
       STCH  RECORD, X
       TIXR  T  : add 1 to index compare T
       JLT   CLOOP
       RSUB
       ........
INDEV  BYTE  X 'F5'
RECORD RESB  200
```
8. Self Check Exercise
1) Write a sequence of instructions to set \( \text{ALPHA} = \text{BETA} \times \text{GAMMA} \)
2) Write a sequence of instructions to set \( \text{ALPHA} \) to the integer portion of \( \text{BETA} / \text{GAMMA} \)
3) Write a sequence of instructions to set \( \text{ALPHA} = \text{GAMMA} \times \text{BETA} - 9 \)
4) Write a sequence of instructions to set \( \text{ALPHA} = 4 \times \text{BETA} - 9 \)
5) Write a sequence of instructions to set \( \text{ALPHA} = \text{BETA} + \text{DECREMENT} + 1 \)
6) Write SIC instructions to swap the values of \( \text{ALPHA} \) \& \( \text{BETA} \)
7) Write a sequence of instructions in SIC/XE to add 2 arrays of 100 integers.
8) Write sequence of instructions for SIC/XE to divide \( \text{BETA} \) by \( \text{GAMMA} \) setting \( \text{ALPHA} \) to the integer portion of quotient \& \( \text{DELTA} \) to the remainder
9) Write a sequence of instructions for SIC to clear 20 bytes of string to all blanks
10) Write a sequence of instructions for SIC \& SIC/XE to copy the string “SYSTEM SOFTWARE” to another string.
11) Write a sequence of instructions for SIC to clear 20 bytes of string to all blanks using register to register instruction \& immediate addressing.
12) Suppose that \( \text{ALPHA} \) is an array of 100 words. Write a sequence of instructions for SIC to set all 100 elements of the array to 0.
13) Bring out the difference between system software \& application software
14) Explain the following w.r.to both SIC \& SIC/XE
   Instruction format, addressing mode, data formats, Register organization
15) Give the target address generated for the following machine instructions:
   032600H, 03C300H, 0310C303 If \( B = 006000 \), \( PC = 003000 \), \( X = 000090 \)
9. Summary
System software consists of a variety of programs that support the operation of a computer. This software makes it possible for the user to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.

Simplified Instructional Computer (SIC) is a hypothetical computer that includes the hardware features most often found on real machines. SIC is having two versions: SIC & SIC/XE. SIC & SIC/XE both the architecture is associated with its Memory, Registers, Data Formats, Instruction Formats, Addressing Modes, Instruction Set, Input and Output.
END of UNIT-1