Module - 1

SANJU.D.J
ASST Prof
Computer Science
S.V.I.T.

* System Software

System Software consists of a variety of programs that support the operation of a computer. This software enables the programmer to concentrate on his program (task) without needing to know as to how the machine works internally.

There are various system softwares such as.

(1) Text editor: used to create and modify the program.

(2) Compiler: used to translate the high level language programs into machine level language.

(3) Loader & Linker: used to load the program from hard-disk to main memory and prepare it for execution.

(4) Debugger: used to detect errors in the program

* Differences between System Software & application S/w

System Software	Application Software
• System softwares are intended to support the operation and use of the computer	• Application softwares are primarily concerned with the solution of some problem using the computer as a tool
• The focus is on the architecture of the computing system.	• The focus is on application and not on the computing system

• System softwares are such softwares which remain in the background and enables the programmer to achieve his task	• Application softwares are such softwares which directly enables the programmers to achieve the required task.
• These are System oriented	• These are user oriented
• They are not dependent on application softwares	• They are dependent on System softwares.
• Eg: Assembler, compilers, linkers, loaders etc	• eg: MS-paint, MS-word, vedio gaming S/w etc

* Differentiate b/w RISC and CISC

RISC	CISC
• Stands for reduced instruction set computer	• Stands for complex instruction set computer
• RISC has a simple architecture	• CISC has a complicated architecture
• provides very few instructions	• provides numerous instruction.
• provides very few registers	• provides relatively more no/. of registers.
• RISC is less expensive	• CISC is relatively more expensive
• Normally supports character and integer data formats	• Normally supports character integer and floating point data formats.
• programing is difficult	• programming is simple

• program execution is slow	• program execution is relatively fast.
• Supports very few addressing modes	• Supports relatively more addressing modes
• Supports very few instruction formats.	• Supports relatively more instruction formats
• programs normally non-relocatable	• programs normally relocatable

* The Simplified instructional computer (SIC)

SIC stands for "Simplified Instructional computer". It is a representative of the RISC family of computers. The SIC machine architecture consists of

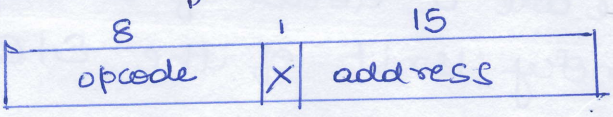
- **Memory:** Consists of bytes made up of 8 bits. Three consecutive bytes form a word. There are a total of 2^{15} bytes in the memory unit of the SIC.
- **Registers:** refers to small storage space available on the CPU. The SIC has 5 registers all of which have special used as tabulated below.

Mnemonic	Number	Use
A	0	Accumulator; used for arithmetic operations
X	1	Index register; used for addressing
L	2	Linkage register; JSUB
PC	8	Program counter
SW	9	Status word, including CC

- **Data Formats:** SIC supports only character and integer data formats. It does not support floating point data formats

  Characters are stored using their 8-bit ASCII codes. positive integers are stored in their 24-bit binary format. Where as negative numbers are stored using the 2's complement 24-bit binary format.

- **Instruction Formats:** SIC supports only one instruction format. All instructions have to be assembled using this format.



- **Addressing Modes:** SIC supports 2 addressing modes namely the direct addressing mode & the indexed addressing mode as shown below.

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

- **Instruction Set:** SIC supports very few instructions. These include load and store instructions (LDA, LDX, STA, STX). Arithmetic instructions (ADD, SUB, MUL, DIV). Conditional jump instructions (such as JLT, JEQ, JGT). instructions related to subroutines (such as JSUB, RSUB) etc.
- **Input and output:** SIC provides 3 instructions for input and output related activities. An instruction RD stands for read data and is used to input data whereas an instruction WD stands for write data and is used to output data. However before performing RD or WD operation. the TD (test device) operation must be performed, to verify in the device is ready or not to perform I/O operation

* SIC/XE Machine Architecture

SIC/XE stands for "Simplified Instructional computer extra equipment" version. It is a representative of the CISC family of computers. The SIC/XE machine architecture consists of:

- **Memory:** Consists of bytes made up of 8 bits. Three consecutive bytes form a word. There are a total of 2^{20} bytes in the memory unit of the SIC/XE computer.

- **Registers :** refers to small storage space available on CPU. The registers supported by SIC/XE are

Mnemonic	Number
A	0
X	1
L	2
PC	8
SW	9

Mnemonic	Number	Special use
B	3	Base register
S	4	General working register
T	5	General working register
F	6	Floating-point accumulator (48 bits)

- **Data Formats :** SIC/XE supports character, integer and floating point data formats.

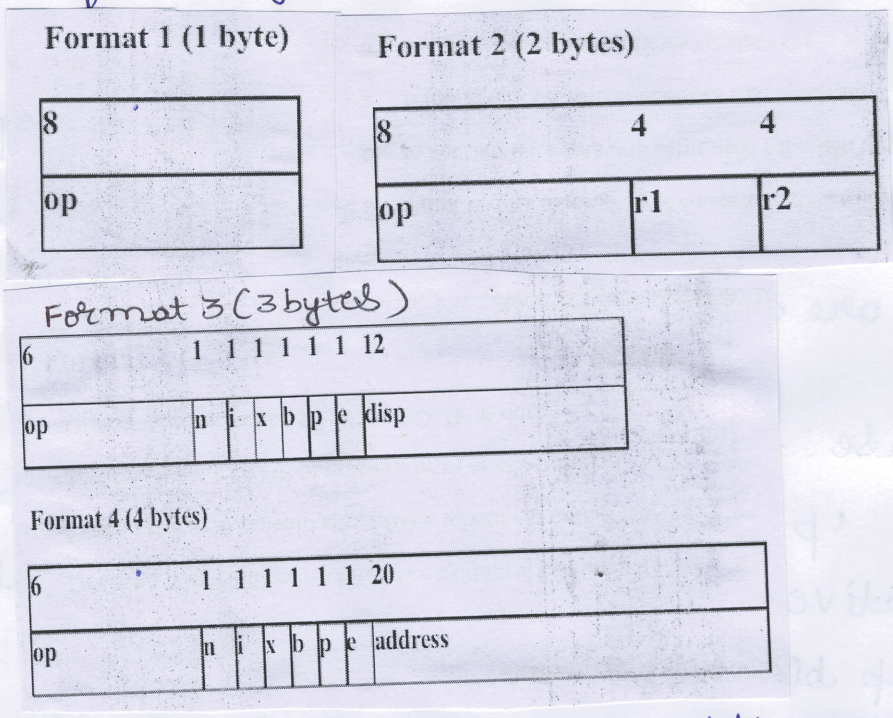
characters are stored using their 8-bit ASCII code. positive integers are stored in their 24-bit binary format, whereas negative integers are stored using the 2's complement 24 bit binary format.

SIC/XE supports a 48-bit floating point data format as shown below.

1	11	36
s	exponent	fraction

In the above format, the S bit is 1 for negative numbers and 0 for positive numbers. Exponent is treated as a number between 0 and 2047. Fraction is interpreted as a value between 0 & 1.

- **Instruction Formats:** SIC/XE supports 4 instruction formats namely one byte format, two, three and four byte formats as shown below.



Format 1 is used for assembling implied addressing mode instructions whereas format 2 is used for assembling register addressing mode instructions. Format 3 & 4 are distinguished using e-bit. For Format 3, e bit would be 0 whereas for format 4, e bit would be 1.

- **Addressing modes:** unlike SIC, the SIC/XE Computer supports numerous addressing modes such as the implied addressing mode, register addressing mode, base relative, program counter relative, direct, indirect addressing mode, immediate, indexed, Simple addressing mode etc.

Instructions which do not have an explicit operand are said to be in the implied

addressing mode and are assembled using Format 1.

eg FIX
HIO etc.

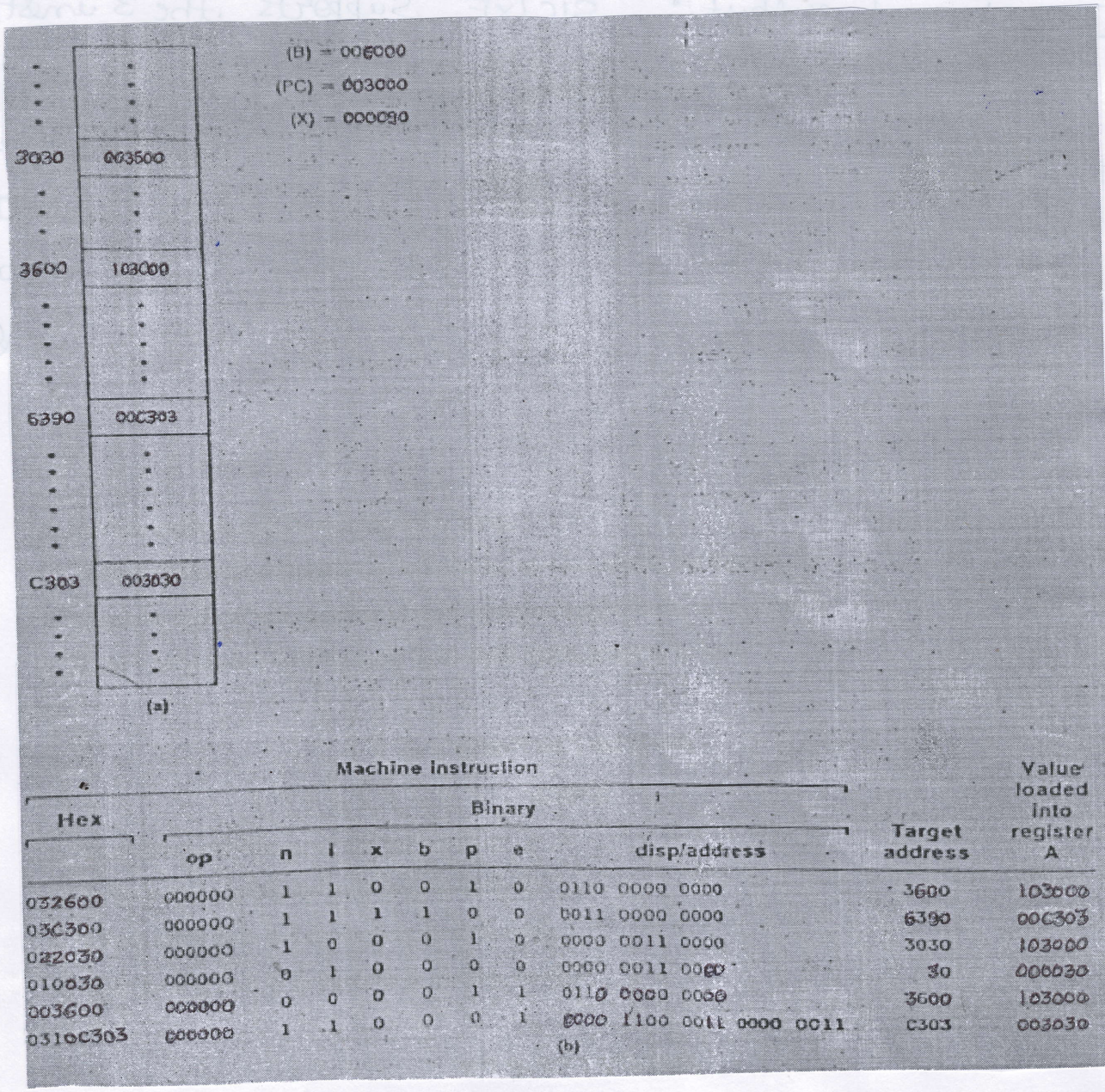
Instructions which have only register operands are said to be in the register addressing mode and are assembled using format 2.

For the base relative addressing mode, 'b' bit is 1 and 'p' bit is 0 whereas for the program counter relative addressing mode, the b bit is 0 and p bit is 1 as shown below.

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)

for direct addressing mode, both the b and p bits are set to 0 and the e bit is set to 1. with respect to n and i bits, the addressing modes can be calculated as shown below.

n	i	addressing mode
0	1	immediate addr mode
1	0	indirect addr mode
1	1	simple addr mode
0	0	Simple addr mode



Machine instruction										
Hex	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	1011 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
031000303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

(b)

- **Instruction Set:** SIC/XE supports numerous instructions. These include load and store instructions (LDA, LDX, LDB, LDS, STA, STX, STB, STS), arithmetic instructions (ADD, ADDF, MUL, MULF, DIV, DIVF) and register to register instructions such as (ADDR, SUBR, MULR, DIVR) etc.

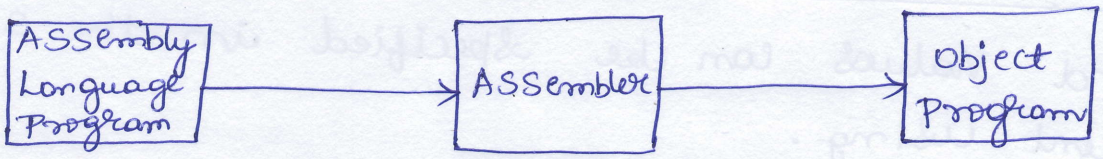
- **Input and output:** SIC/XE supports the 3 instructions RD, WD and TD (as supported by SIC) to read, write and test the device. Along with these instructions, it also provides SIO, TIO & HIO instructions which are used to start, test and halt the operation of I/O channels.

* SIC programs
* SIC/XE programs

} Read from class notes.

* ASSEMBLERS

- Translating mnemonic language code to its equivalent object code.
- Assigning machine address to symbolic labels.



* Assembler Directive

Assembler directive as the name suggests represents a set of directions given by the programmer to the assembler regarding the manner in which program must be assembled. Assembler directives are not instructions and hence would not be converted by the assembler to the machine level language. Rather Assembler Directives are guidelines using which the Assembler would perform its assembly process.

Ex: START : Specify name & Starting address for the program

END : Indicate End of the program & specify the first execution instruction in program

BYTE : Generate character or hexadecimal constant, occupying as many bytes as needed to represent the constant

WORD : Generate one word integer constant.

RESB : Reserve the indicated no. of bytes for data area

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

* What are the different ways of Specifying Operand Values in the source statement? write advantages and disadvantages.

Operand Values can be specified in the source statement using.

1. Implied mode
2. Register mode
3. Immediate mode
4. Relative mode
5. Absolute mode

(1) Implied mode: In this mode, the operand value is not explicitly specified in the instruction. It is implied.

eg: FIX
RSUB

adv:
- It occupies one byte in memory
- Relocatable
- Fast in execution.

disadv: Not very powerful.

(2) Register mode: in this mode, the operand values are specified through registers such as A, S, T, B etc.

eg: ADDR S, X
COMPR X, T

Adv : • It occupies 2 bytes in the memory unit
• Relocatable
• Fast in execution

Disadv : • There are limited number of registers on the CPU and hence it is not a powerful addressing mode

③ Immediate mode : In this mode the data is directly specified in the instruction itself.

eg : LDA #0
LDX #100

Adv : • Relocatable.
• Fast in execution.

Disadv : • Range of data that can be specified through immediate addressing mode is limited.

④ Relative mode : In this mode, the operand value is not expressed in absolute terms. Rather it is expressed in relative terms by using the Base register (B) or the program counter (P)

eg : LDA COUNT
LDX TOTAL

Adv : • Relocatable

Disadv : • 12 bit disp field is available to specify the relative address.

⑤ Absolute mode : In this mode, the operand value is expressed in absolute terms. The Base register (B) or the program counter (P)

is not used in the specification of the absolute address.

eg: +LDT TOTAL
+LDA TABLE

adv: • A large range of operand value can be specified since 20 bit address field would be available in the format.

Disadv: • Not relocatable
• occupies 4 bytes in the memory unit.

* What are literal operands? how does the assembler handle literals

Such operands whose value are literally stated in the instruction is called as a literal operand. A literal operand is normally identified by a prefix = which is followed by the specification of literal value.

eg: LDA =C'EOF'
RD =X'05'

character literals would be identified with the prefix =C & hexadecimal literals with the prefix =X.

The basic data structure needed by the assembler to handle litterals is called as the LITTAB. This table would contain the name of the literal, value of the literal, length of the literal & the address assigned to the literal.
LITTAB would often be created as a hash-table.

During the I pass of the assembler, when the assembler encounters a literal operand it searches the LITTAB for the operand. If the literal is already present in the table, then no action is taken. If it is not present, then the literal is added to the LITTAB. Finally when the END or LTORG assembler directive is encountered, then the assembler would assign addresses to the accumulated literal operands in the LITTAB.

During the II pass of the assembler, the data values specified by the literals are inserted at appropriate places in the object program.

* LTORG Assembler directive

The need for an assembler directive such as LTORG usually arises when it is desirable to keep the literal operand close to the instruction that used it.

Normally, all the literal operands used in the program are gathered together in a table and are placed at the end of the program. However, in some cases it would be desirable to place the literal operands not at the end of the program but at some other previous location. This is possible by using the LTORG assembler directive

If the LTORG assembler directive is not used, some times there is a possibility for the literal operand to get placed very far away from the instruction that uses it and hence we may not be able to assemble the instruction using a 3 byte format. we may have to use a 4 byte format as shown below.

```
BACK    +TD   INDEV
        JEQ   BACK
        +RD   INDEV
        .
        .
BUFFER  RESW  2000
INDEV   BYTE  =X'05'
```

This problem can be overcome using the LTORG assembler directive as shown below.

```
BACK    TD    INDEV
        JEQ   BACK
        RD    INDEV
        LTORG
BUFFER  RESW  2000
INDEV   BYTE  =X'05'
```

* Data structures used in the design of assemblers.
* Write the Algorithm for pass one of a 2 pass assembler.
* Write the Algorithm for pass two of a 2 pass assembler.
* Program Relocation & Example
* Symbol definition statements & Example
* program Blocks & example
* control sections & Example
* Two pass assembler & Example
* Multi pass assembler & Example
* Different Record formats used in order to obtain a object code & give their formats
* Problems on two pass assembler(object code generation)
* problems on multi pass assembler.
* Macro processor & Macro definition.
* Macro processor algorithm.
* Macro processor data structures & Example

Above topics Read from Class notes