MODULE I

SYLLABUS

System Software vs Application Software, Different System Software– Assembler, Linker, Loader, Macro Processor, Text Editor, Debugger, Device Driver, Compiler, Interpreter, Operating System (Basic Concepts only). SIC & SIC/XE Architecture, Addressing modes, SIC & SIC/XE Instruction set, Assembler Directives.

INTRODUCTION

Computer Software

A computer cannot do anything on its own. It must be instructed to do a job. So it's necessary to specify a sequence of instructions that a computer must perform to solve a problem. Such a sequence of instructions written in a language understood by a computer is called a computer program. Computer software is the collection of computer programs and related data that provide instructions telling a computer what to do. Software is divided into 2 types:

1. System Software
2. Application Software

1. System Software

System Software is a set of programs that manages and supports the operation of a computer. It controls the computer system and enhances its performance. It enables the application software to interact with the system hardware. That is system software act as a bridge between application software and computer hardware. Eg: Operating System, Compiler etc. System software can be broadly classified into 3 types:

    i. System Control Programs
    ii. System Support Programs
    iii. System Development Programs

i.System Control Programs: controls the execution of programs, manage the storage and processing resources of the computer and perform other management and monitoring functions. Eg: Operating System, Database Management Systems (DBMS).

ii. System Support Programs: provide routine service functions to the other computer programs and computer users. Eg: utilities, libraries

iii.System Development Programs: assists in the creation of application programs. Eg: Language translators like compiler, assembler.
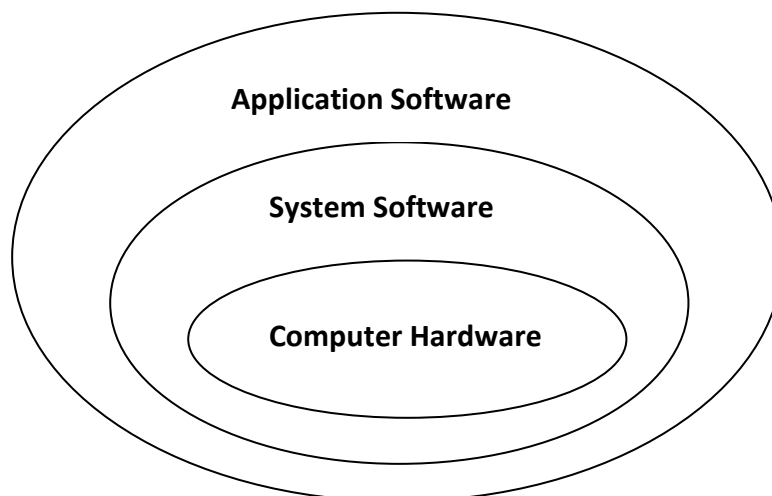
I. Application Software:

Application software consists of programs designed to perform specific tasks for users. .Eg: Media player, Ms. Word, Notepad etc. There are 2 types of application software:

i. General Purpose Application Software: These provide general user needs, not for a specific purpose. Eg: Spreadsheet program like Microsoft Excel, which can be used to perform different applications.

ii. Special Purpose (Custom) Application Software: Typically used for specific applications. Eg: Turbo Tax, which is a special purpose application used to perform tax returns.

Relationship between system software and application software:

System software control and manages hardware thereby providing a platform for application software to operate. Application software helps user to accomplish one or more tasks using a computer through system software. For an application to run on a computer it needs to be allocate space from memory, allocated resources (CPU time, hardware like keyboard, display, printer etc), given access to system libraries, all of which is done by operating system which is a system software.

**Differences between system software and application software (System Software Vs Application Software):**

Category	Application Software	System Software
Definition	Application software is computer software designed to help the user to perform specific tasks	System software is computer software designed to operate the computer hardware and to provide a platform for running application software.
Purpose	It is specific purpose software.	It is general purpose software.
Environment	Application Software performs in a environment which created by Operating System	System Software creates his own environment to run itself and run other application.
Execution Time	It executes as and when required	Some system software must execute all the time in computer.
Essentiality	Application software is not essential for a computer.	System software is essential for a computer
Number	The number of application software is much more than system software.	The number of system software is less than application software.
Machine dependent/ independent	Application software uses the computer for solving problems. That is, its focus is on application. So its machine independent.	System software is intended to support the use and operation of a computer. So it's usually machine dependent.

DIFFERENT SYSTEM SOFTWARE

1. Language Translators
   a. Compiler
   b. Interpreter
   c. Assembler
   d. Macro Preprocessor
2. Linker
3. Loader
4. Text Editor
5. Device Driver
6. Debugger
7. Operating Systems
8. Database Management Systems(DBMS)

1. **<u>Language Translators</u>**

It is the program that takes an input program in one language and produces an output in another language.
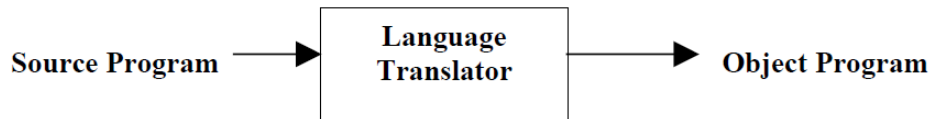
Different language translators are:

a. Compiler
b. Interpreter
c. Assembler
d. Macropreprocessor

a. Compiler

A compiler is a program that translates programs written in any high level language (source program) into its equivalent machine language program (target language). An important role of the compiler is to report any errors in the source program that it detects during the translation process.
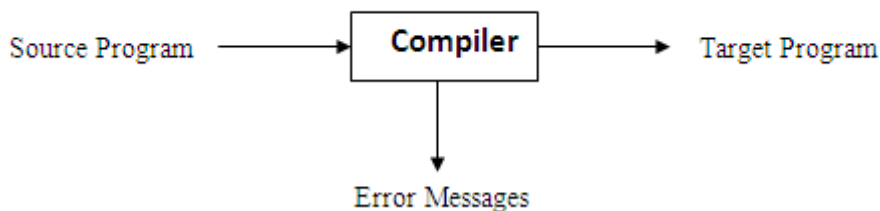
b. Interpreter

An interpreter is also a language translator which converts source program in high level language into machine language, just like compiler did. It reads the source code one line at a time converts this line into machine code and executes it. The machine code is then discarded and next line is processed. It stops translating after the first error.

Compilers, on the other hand, translates the entire program in one go and then executes it. Compiler analyse the entire program, displays where errors have occurred. If errors are present, then program cannot run.

Interpreter works as follows. The interpreter reads the source program and stores it in memory. Program counter (PC) indicates which statement of the source program is to be interpreted next. During interpretation, it takes a source statement, determines its meaning and

performs the actions and PC is incremented. The interpretation cycle consists of the following steps:

- Fetch the statement.
- Analyze the statement and determine its meaning.
- Execute the meaning of the statement.

Advantages of interpreter:

Easier to use particularly for beginners, since errors are immediately displayed.

.**Disadvantage of interpreter:**

Every line has to be translated every time it is executed, even if it is executed many times as the program runs. Because of this interpreters tend to be slow.

Example for interpreters: Basic on older home computers, script interpreters such as JavaScript.

c. Assembler

Assembler converts assembly language program into its equivalent machine language.

Assembly Language ⟶ | **Assembler** | ⟶ Machine Language

Example: MASM (8086 assembler)

d.Macro Preprocessor

A macro processor is a program that reads a file and scans them for certain keywords. When a keyword is found, it is replaced by some text. The keyword/text combination is called a macro. A simple example is the C language pre-processor:

```
#define max 100;
int i;
for (i=0; i<max; i++)
{
.........
}
```

The C prerocessor reads the first line and stores it as macro definition. When it comes across the later reference of max in the for loop, it replaces with the macro definition 100. The output of the C processor is then fed to the C compiler.

2. Linker

A linker is a program that combines object modules to form an executable program. Many programming languages allow us to write different pieces of code, called *modules*, separately. This simplifies the programming task because we can break a large program into small, more manageable pieces. Eventually, though, we need to put all the modules together. This is the job of the linker. In addition to combining modules, a linker also replaces symbolic addresses with real addresses. Therefore, we may need to link a program even if it contains only one module.
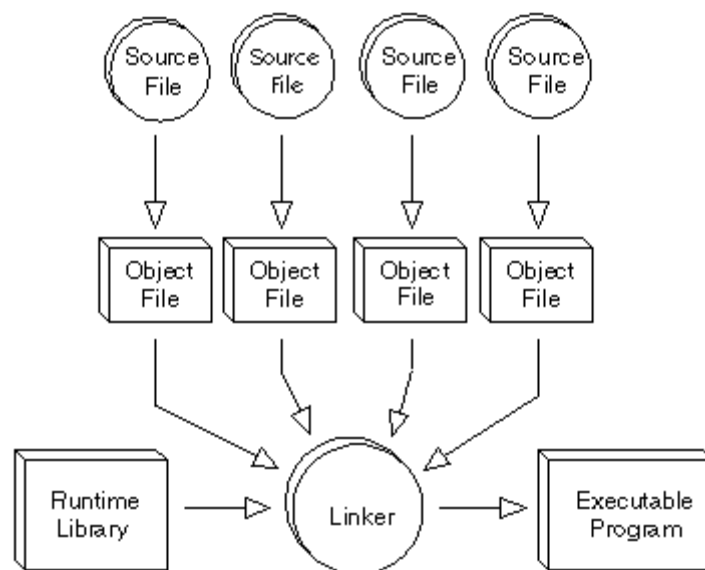
3. Loader

A **loader** is the part of an operating system that is responsible for loading programs. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution. Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code.

4. Text Editors

A **text editor** is a type of program used for editing plain text files. Text editors are often provided with operating systems or software development packages. Example for text editors are Microsoft word, gedit in Linux etc.

5. Debuggers

A **debugger** or **debugging tool** is a computer program that is used to test and debug other programs.

6. Device Drivers

In computing, a **device driver** is a computer program that operates or controls a particular type of device that is attached to a computer. A Device Driver is glue between an OS and its I/O devices. They act as translators converting requests received from the operating system into commands that the devices can understand.

7. Operating System

It is the most important system program that act as an interface between the users and the system. It makes the computer easier to use. It provides an interface that is more user-friendly than the underlying hardware.

The functions of OS are:

1. Process management
2. Memory management
3. Resource management
4. I/O operations
5. Data management
6. Providing security to user's job.

8. Database Management System

A **database** is a collection of related data. Data can be considered as a piece of information which can be recorded.

Data Base Management System (DBMS) is a collection of programs that enables users to create and maintain a database. It is general purpose software that facilitates the processes of defining, constructing, manipulating and sharing of database among various users and applications.

- **Defining** a database means specifying the data types, structures and constraints for the data to be stored in the database.
- **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.

- **Manipulating** the database means processing the database. It includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the mini world and generating reports from the data.

- **Sharing** a database allows multiple users and programs to access the database concurrently.

A **Database System** is a computerized record keeping system whose overall purpose is to store information and to allow users to retrieve and update that information on demand. Database system consists of a Database, Database Management System and an Application program. An application program accesses the database by sending requests or queries for data to the DBMS.
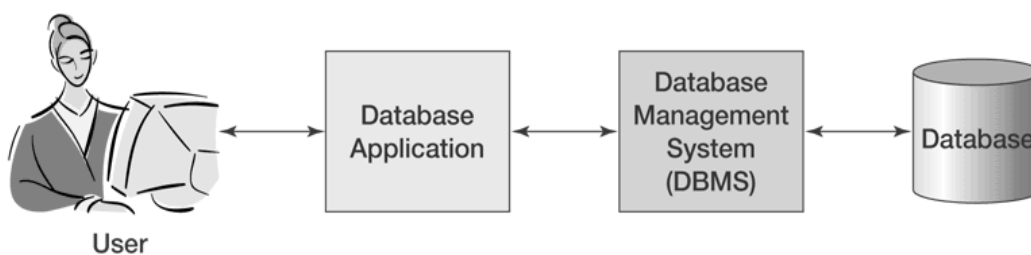

Fig: A Database System

THE SIMPLIFIED INSTRUCTIONAL COMPUTER (SIC)

It is a hypothetical computer that has been carefully designed to include the hardware features most often found in real machine. It comes in two versions:

- The standard model

- XE version

SIC MACHINE STRUCTURE

Memory

- It consists of bytes(8 bits)
- Any 3 consecutive bytes form a word (24 bits) which are addressed by the location of their lowest numbered byte.
- There are totally 2^{15}=32,768 bytes in memory.

Registers

SIC machines have 5 registers, each 24 bits long and having both a numeric and character representation. The registers are:

- **A (0):** Used for basic arithmetic operations; known as the accumulator register.
- **X (1)**: Used for addressing; known as the index register.
- **L (2)**: Used for storing the return address of the jump of subroutine instructions (JSUB); known as the linkage register.
- **PC (8)**: Contains the address of the next instruction to be executed; known as the program counter register.
- **SW (9)**: Contains a variety of information, (such as carry or overflow flags)including a condition code (CC); known as the status word register.

(In the representation A (0) – A is the character representation for the accumulator register and 0 is the numeric representation. Similarly for other registers.)

Data formats

- Integers are stored as 24-bit binary numbers
- For storing negative numbers 2's complement representation is used
- Characters are stored using their 8 bit ASCII codes.
- SIC do not support floating point data items.

Instruction Formats

- All machine instructions are of 24-bits length. It has following format:

Opcode (8)	X (1)	Address (15)

- x is a flag bit used to indicate indexed-addressing mode.

  If x=0, then direct addressing mode

  If x=1, then indexed addressing mode/indirect addressing mode

Addressing modes

The term addressing modes refers to the way in which the operand of an instruction is specified. Two types of addressing modes are available in SIC architecture, indicated by the setting of x-bit in the instruction. They are

   1. Direct addressing mode

   2. Indexed addressing mode or indirect addressing mode

Addressing Mode	Indication	Target Address Calculation
Direct	X=0	TA=address
Indexed/Indirect	X=1	TA=address +(X)

- where (X) represents the contents of index register.

Instruction Set

It includes instructions like:

1. Load and Store Registers (Data movement instruction)

- Ex: LDA, STA, LDX, STX.
- LDA – Load Accumulator, STA – Store the content of Accumulator to specified location
- LDX – Load X Register, STX – Store the content of X Register to specified location

2. Arithmetic operating instructions

- Ex: ADD, SUB, MUL, DIV.
- This involves register A and a word in memory, with the result being left in the A register.

3. Compare Instruction

- Ex: COMP

- Instruction COMP compares a value in A with a word in memory, and sets the condition code CC to indicate the result of <, > and =.

4. Branching instructions (Conditional Jump Instruction)

- Ex: JLT, JEQ, JGT.

- These instructions test the settings of CC and jump accordingly.

- JLT – Jump on Less than, JEQ- Jump on Equal, JGT – Jump on Greater than

5. Subroutine linkage instructions

- Ex: 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.

Input and Output Operations

- Input/Output is 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
- There are 3 I/O instructions, each of which specifies device code as an operand.

1. **TD (Test device)**
   - Tests whether the addressed device is ready to send or receive a byte of data
   - The condition code is set to indicate the result of this test.
   - If CC setting is < the device is ready, if setting is = device is not ready
2. **RD (Read Data)**
   - for reading data

3. **WD (Write Data)**
   - for writing data

- A program needing to transfer data must wait until the device is ready, and then execute RD or WD.

- This sequence must be repeated for each byte of data to be read or written.

SIC/XE MACHINE STRUCTURE

Memory

- It consists of bytes(8 bits)
- Any 3 consecutive bytes form a word (24 bits) which are addressed by the location of their lowest numbered byte.
- There are totally 2^{20}=**1,048,576bytes (1 Mbyte)** in memory.

Registers

- Totally 9 registers are available in SIC/XE
- All registers except Floating point accumulator are of 24 bits. Floating point accumulator is of 48 bits.

MNEMONIC	Register	Purpose
A	0	Accumulator
X	1	Index register
L	2	Linkage register (JSUB/RSUB)
B	3	Base register
S	4	General register
T	5	General register
F	6	Floating Point Accumulator (48 bits)
PC	8	Program Counter (PC)
SW	9	Status Word (includes Condition Code, CC)

Data formats

- Integers are stored as 24-bit binary numbers
- For storing negative numbers 2's complement representation is used
- Characters are stored using their 8 bit ASCII codes.
- SIC/XE supports floating point data items. Floating point is stored in 48 bit signed-exponent-fraction format

| s	exponent {11}	fraction {36}
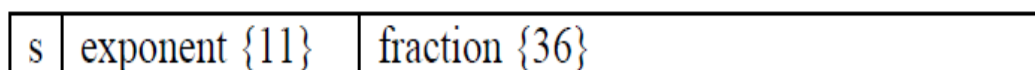
- The fraction is represented as a 36 bit number and has value between 0 and 1

- The exponent is represented as an 11 bit unsigned binary number between 0 and 2047.
- The sign of the floating point number is indicated by s: 0=positive, 1=negative.
- Therefore, the absolute floating point number value is: $f*2^{(e-1024)}$

Instruction Format

There are 4 different instruction formats available in SIC/XE:
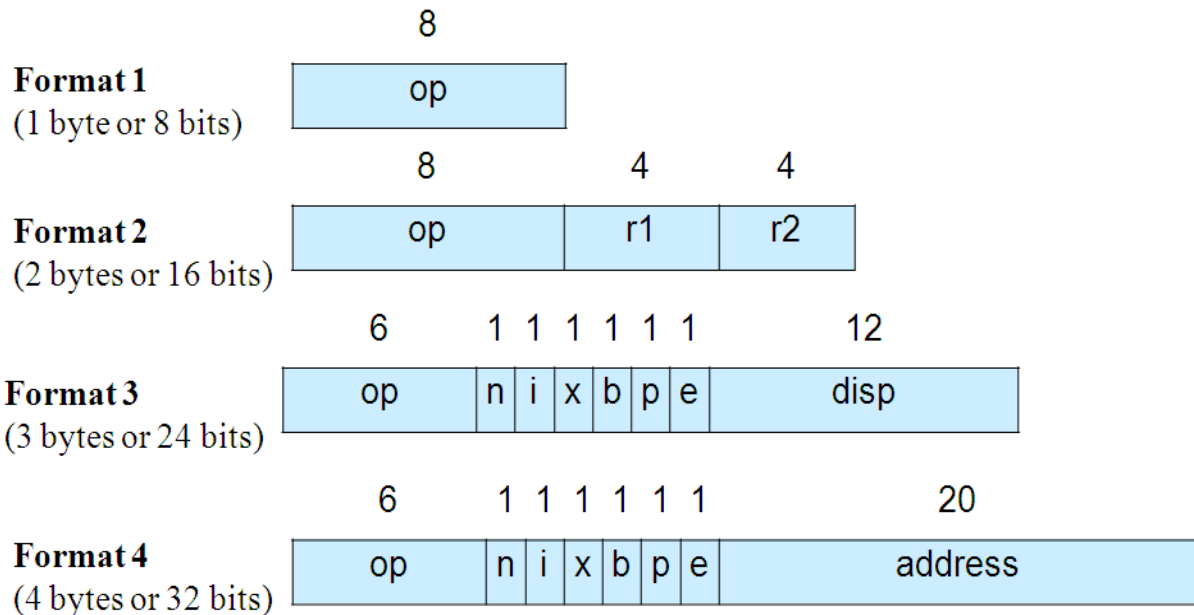
- Formats 1 and 2 do not reference memory at all
- Instruction format 3 and 4 introduce addressing mode flag bits.

    Flag e:

    e=0 use Format 3

    e=1 use Format 4

    Flag p – indicates **P**rogram counter relative addressing

    Flag b - indicates **B**ase register relative addressing

    Flag x – indicates inde**x**ing. (uses index register)

    Flag i – indicates **i**mmediate addressing

    Flag n – indicates i**n**direct addressing

Instruction Set

SIC/XE supports all of the instructions that are available on standard SIC. SIC/XE supports following instructions:

1. **Load and Store Registers (Data movement instruction)**

   - Ex: LDCH, STCH, LDB, STB.
   - LDCH – Load Accumulator with character, STCH – Store the character on Accumulator to specified location.
   - LDX – Load X Register, STX – Store the content of X Register to specified location

2. **Integer Arithmetic operating instructions**

   - Ex: ADD, SUB, MUL, DIV.
   - This involves register A and a word in memory, with the result being left in the A register.

3. **Floating Point Arithmetic Instruction**

   - Ex: ADDF, SUBF, MULF, DIVF
   - F- stands for Floating Point

4. **Compare Instruction**

   - Ex: COMP
   - Instruction COMP compares a value in A with a word in memory, and sets the condition code CC to indicate the result of $<$, $>$ and $=$.

5. **Branching instructions (Conditional Jump Instruction)**

   - Ex: JLT, JEQ, JGT.
   - These instructions test the settings of CC and jump accordingly.
   - JLT – Jump on Less than, JEQ- Jump on Equal, JGT – Jump on Greater than

6. **Subroutine linkage instructions**

   - Ex: 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.

7. **Register Related Instructions**

- This involves the instructions for register manipulation, operands-from-registers and register-to-register arithmetic operations.

- Ex: RMO , COMPR, SHIFTR, ADDR, SUBR, MULR, DIVR, etc

  RMO – Register Move Instruction

  ADDR, SUBR, MULR, DIVR – Register Arithmetic operations

  COMPR – Register Comparison

  SHIFTR- Register Shift Operation

8. **Supervisor Call Instruction (SVC)**

   - The instruction SVC generates an interrupt that can be used for communicating with the operating system.

Input and Output Operations

- Input/Output is 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

- There are 3 I/O instructions, each of which specifies device code as an operand.

  1. **TD (Test device)**

     - Tests whether the addressed device is ready to send or receive a byte of data
     - The condition code is set to indicate the result of this test.
     - If CC setting is < the device is ready, if setting is = device is not ready

  2. **RD (Read Data)**

     - for reading data

  3. **WD (Write Data)**

     - for writing data

- A program needing to transfer data must wait until the device is ready, and then execute RD or WD.

- This sequence must be repeated for each byte of data to be read or written.

- SIC/XE has capability for programmed I/O. Programmed I/O means I/O devices may input or output data while CPU is busy with other works.3 instructions are provided for handling programmed I/O features:
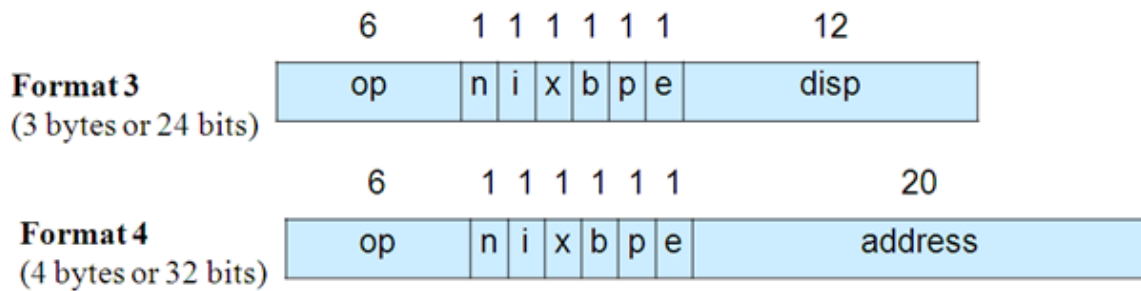
  SIO - Start I/O

HIO - Halt I/O

TIO - Test I/O

Addressing Modes

The term addressing modes refers to the way in which the operand of an instruction is specified. Instruction format 3 and 4 introduce addressing mode flag bits.

Flag e:

e=0 use Format 3

e=1 use Format 4

Flag p – indicates **P**rogram counter relative addressing

Flag b - indicates **B**ase register relative addressing

Flag x – indicates inde**x**ing. (uses index register)

Flag i – indicates **i**mmediate addressing

Flag n – indicates i**n**direct addressing

1. Indexed Addressing Mode

- If x=1, then it indicates indexed addressing mode.
- In this addressing mode, the content of x register is added with address/disp for the target address calculation.

TA=address +(X)

- Indexed addressing mode can be combined with PC relative and Base relative addressing mode. In this case, the content of X register is also added for the target address calculation.
- Indexing cannot be used with immediate or indirect addressing modes.

2. Immediate Addressing Mode

- If **n=0 and i=1**, then it indicates Immediate addressing mode.

- In this addressing mode, target address (TA) itself is used as the operand value.
- No memory reference is performed.
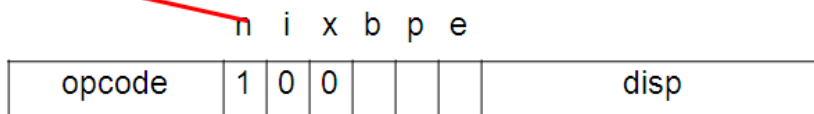
Immediate Addressing Mode

```
            n  i  x  b  p  e
opcode    | 0| 1| 0|  |  |  |      disp
```
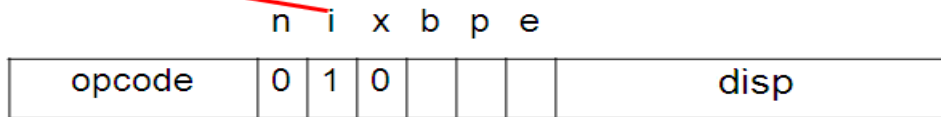
n=0, i=1, x=0, operand=disp

3. Indirect Addressing Mode

- If **n=1 and i=0**, then it indicates Indirect addressing mode.
- In this addressing mode, word at the location given by the target address is fetched. The value contained in this word is then taken as the address of the operand value.

Indirect Addressing Mode

```
            n  i  x  b  p  e
opcode    | 1| 0| 0|  |  |  |      disp
```

n=1, i=0, x=0, TA=(disp)

4. Simple Addressing Mode

- **If n=0 and i=0,** then it indicates simple addressing mode.
- In this addressing mode, TA is taken as the location of the operand
- If **n=1 and i=1,** it also indicates simple addressing mode(same as n=0 and i=0)

5. Program Counter Relative Addressing

- **If b=0 and p=1,** then it Indicates Program Counter (PC) relative addressing mode.
- In this addressing mode, the displacement given in instruction is added with the content of program counter (PC) for obtaining the target address.

    TA = (PC) +disp

Program-Counter Relative Addressing Mode

```
          n  i  x  b  p  e
+--------+--+--+--+--+--+--+----------------+
| opcode |  |  |  | 0| 1|  |      disp      |
+--------+--+--+--+--+--+--+----------------+
```
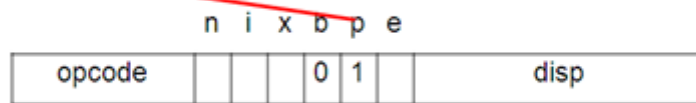
b=0, p=1, TA=(PC)+ disp

- Indexed addressing mode can be combined with PC relative addressing mode. In this case, the content of X register is also added for the target address calculation.

TA = (PC) +(X) +disp

6. **Base Relative Addressing Mode**

- **If b=1 and p=0,** then it **i**ndicates Base relative addressing mode.
- In this addressing mode, the displacement given in instruction is added with the content of base register for obtaining the target address.

TA= (B) +disp

Base Relative Addressing Mode

```
          n  i  x  b  p  e
+--------+--+--+--+--+--+--+----------------+
| opcode |  |  |  | 1| 0|  |      disp      |
+--------+--+--+--+--+--+--+----------------+
```

b=1, p=0, TA=(B)+disp

- Indexed addressing mode can be combined with Base relative addressing mode. In this case, the content of X register is also added for the target address calculation.
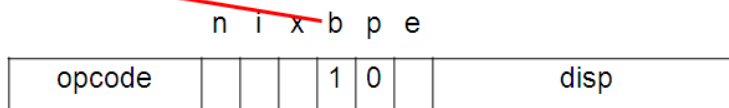
TA = (B) +(X) +disp

7. **Direct Addressing Mode**

- If **x=0**, then it indicates direct addressing mode.
- Then target address is the address/disp given in the instruction.

TA=address

Examples of SIC/XE instructions and addressing modes

- Following figure gives examples of different addressing modes available on SIC/XE.

- The contents of B, PC and X registers are given as (B)=006000, (PC)= 003000 and (X)=000090. All values are given in hexadecimal.
- Fig b gives the machine code for a series of LDA instruction (opcode for LDA is 00)
- The target address generated by each instruction and the value loaded into register A is also shown
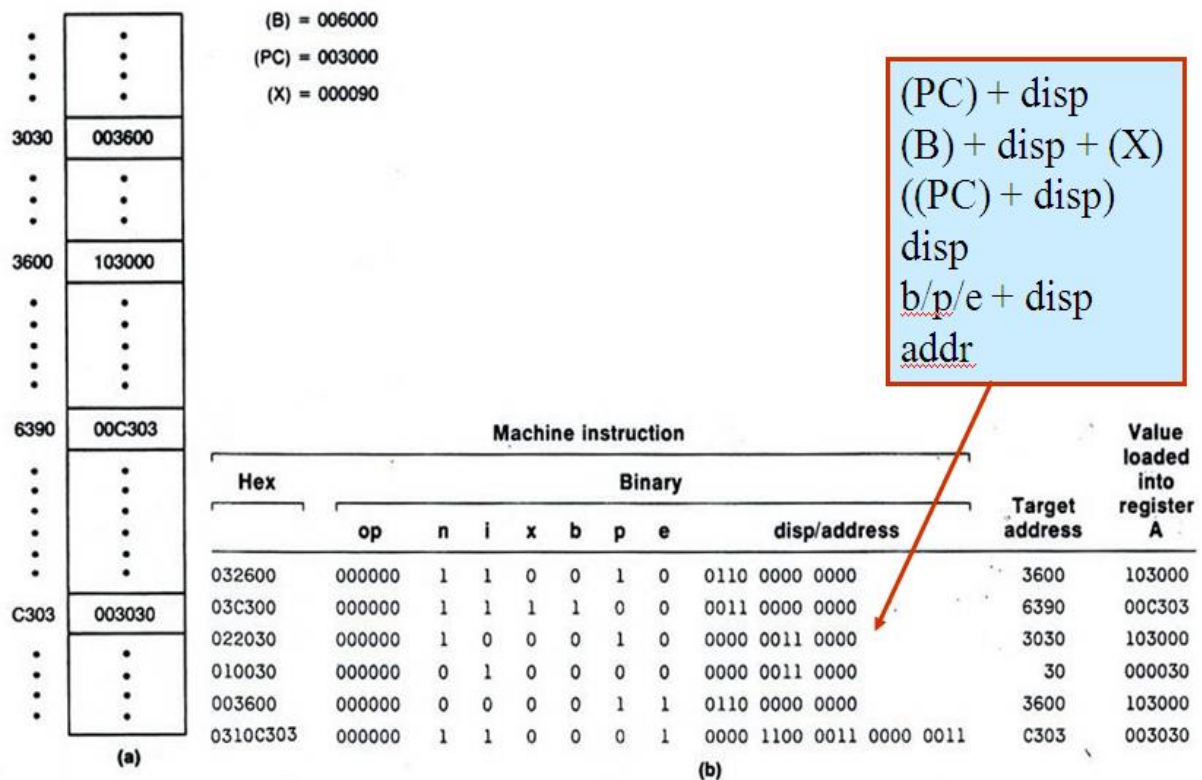
SIC Assembler Directives

Assembler directives are instructions that direct the assembler to do something.

The assembler directives in SIC are:

1. **START**
   - Indicates the start of program.
   - Used to define program name and starting address
   - Eg: COPY START 1000
     This means the program name is COPY and the starting address is 1000
2. **END**
   - Used to indicate the end of program.
   - Optionally indicates first executable instruction.
   - Eg: END ALPHA
     This means the program name ENDs at here. And the first executable instruction is ALPHA

3. **RESW**
   - Used to reserves specified word for a data area.
   - Eg: ALPHA RESW   4
   This is to reserve 4 words. (4 words means 4*3 =12 bytes)

4. **RESB**
   - Used to reserves specified byte for a data area.
   - Eg: A RESB 5
   This is to reserve 5 bytes.

5. **WORD**
   - Used to generate one word integer constant
   - Eg: B WORD 6
   This uses one word to store the integer constant 6 and the name B is assigned to the first location

6. **BYTE**
   - Generate character constant using required number of bytes
   - Eg: ALPHA BYTE C 'HAI'
   This generates number of bytes needed to store HAI.

SIC/XE Programming Examples

Program 1 //Sample data movement

Write a sequence of instructions for SIC/XE which stores the value 5 in ALPHA and the character Z in C1

	LDA	#5	Load value 5 into register A
	STA	ALPHA	Store in ALPHA
	LDCH	#90	Load ASCII code for 'Z' into register A
	STCH	C1	Store in character variable C1
	…		
	…		
	…		
ALPHA	RESW	1	Reserve one-word variable for ALPHA
C1	RESB	1	Reserve one-byte variable for ALPHA

Program 2 //Arithmetic operations

Write a sequence of instructions for SIC/XE which performs the following computations BETA=ALPHA+INCR-1 and DELTA=GAMMA+INCR-1

LDS	INCR	Load value of INCR into register S	
	LDA	ALPHA	Load ALPHA into register A
	ADDR	S,A	Add the value of INCR and ALPHA
	SUB	#1	Subtract 1
	STA	BETA	Store in BETA
	LDA	GAMMA	Load GAMMA into register A
	ADDR	S,A	Add the value of INCR and GAMMA
	SUB	#1	Subtract 1
	STA	DELTA	Store in DELTA
	…		
	…		
ALPHA	RESW	1	One-word variables
BETA	RESW	1	
GAMMA	RESW	1	
DELTA	RESW	1	
INCR	RESW	1	

Program 3 // looping and indexing

Write a sequence of instructions for SIC/XE which copies one string to another.

	LDT	#11	Initialize register T to 11
	LDX	#0	Initialize index register to 0
MOVECH	LDCH	STR1, X	Load character from STR1 to reg A
	STCH	STR2, X	Store character into STR2
	TIXR	T	Add 1 to index compares result with register T
	JLT	MOVECH	Loop if index is less than 11 .
	…		
	…		
	…		
STR1	BYTE	C'TEST STRING'	11 byte string constant
STR2	RESB	11	11 byte variable

Program 4 // looping and indexing

The variables ALPHA, BETA and GAMMA are arrays of 100 words each. Write a sequence of instructions for SIC/XE which add together the corresponding elements of ALPHA and BETA and storing the results in the elements of GAMMA.

	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 from ALPHA to reg A
	ADD	BETA, X	Add word from BETA
	STA	GAMMA, X	Store the result in A into GAMMA
	ADDR S, X		Add 3 to index value
	COMPR	X, T	Compare new index value to 300
	JLT ADDLP		Loop if index is less than 300
	...		
	...		
ALPHA	RESW 100	Array variables—100 words each	
BETA	RESW 100		
GAMMA	RESW 100		

Program 5 //Sample input and output operations

Write a sequence of instructions for SIC/XE which reads 1 byte of data from device F1and copies it to device 05.

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 F1
OUTDEV	BYTE	X'05'	Output device number O5
DATA	RESB	1	1 byte variable

Program 6 //Subroutine call

Write a subroutine to read a 100 –byte record from an input device into memory.

```
          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
```

-For the detailed explanation of program please go through the text book. (System Software – An Introduction to Systems Programming – Leland L. Beck, page number 12 to 20)