**MODULE I**

**Introduction**

* 1. • Software is set of instructions or programs written to carry out certain task on digital computers. It is classified into two.

o System software

* 1. o Application software

**System Software**

• It is a set of programs that supports the operation of a computer.

• It is a set of programs to perform a variety of system functions as file editing, resource Management, I/O management and storage management.

• System software is 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 run.

• It acts as an intermediary between computer hardware and application programs.

• It controls the computer system and enhances its performance.

• Eg: Operating system, Compiler, Debugger, Assembler, Interpreter etc.

**Application Software**

• Application software consists of programs designed to perform specific tasks for users.

• It performs particular functions for the user.

• It is program written for or by a user to perform a particular job.

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

• Application software lies on the top of the system software, because it is unable to run without the operating system and system utilities.

• Application software can be used to assist with graphics and multimedia projects, to support home, personal and educational activities and to facilitate communications.

Eg: Oracle, MS Access, Word processors, Web browsers, Spread sheets, Media Player, MS PowerPoint etc.

|  |  |  |
| --- | --- | --- |
| **Category** | **System Software** | **Application Software** |
| Definition | It is designed to operate the computer hardware and to provide a platform for running application software | It is designed to help the user to perform specific task |
| Purpose | It is a general purpose software | It is a specific purpose software |
| Classification | Time sharing OS, Real time OS, Multiprocessing OS, Distributed OS | Package program, Customized program |
| User Interaction | In general, the user does not interact with system software because it works in the background. | In general, the user interacts with application software. |
| Environment | It creates its own environment to run itself and run other applications | It performs in an environment which is created by Operating System |
| Execution Time | It executes all the time in computer | It executes as and when required |
| Essentials | System software is essential for a computer | Application is not essential for a computer |
| Number | Number of system software is less than application software | Number of application software is much more than system software |
| Machine Dependency | System software supports the use and operation of a computer. So it is **machine dependent** | It focuses on applications rather than computer. So it is **machine independent**. |



**Different System Softwares**

System software is a computer software designed to operate and control the computer hardware and to provide a platform for running application software.

Different System Softwares are:

1. Macro processor

2. Assembler

3. Linker

4. Loader

5. Text Editor

6. Debugger

7. Device Driver

8. Compiler

9. Interpreter

10. Database Management System

11. Operating System

**Macro Processor**

• Macro represents a commonly used group of statements in the source program.

• Macro processor is a program that copies a stream of text from one place to another, by making a systematic set of replacements.

• It 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 **macro**.

• Macro processors are embedded in other programs such as assemblers and compilers.

**Assembler**

• An assembler is a program that translates an assembly language program into a machine language program.



* 1. Two types: o **Self assembler**/**Resident assembler:** An assembler runs on a computer and produces machine code for the same computer.
  2. o **Cross assembler:** An assembler runs on a computer and produces machine code for other computer.

**Linker**

• Linker is a program which helps to link a object modules of program into a single object file. It performs the process of linking.

• Combines two or more separate object programs and supplies the information needed to allow references between them.



**Loader**

• Loader is a program that loads machine codes of a program into system memory.

• Loader is a part of operating system that is responsible for loading program.

• Loading a program involves reading the contents of executable file into memory.

• Once loading is completed, the OS starts the program by passing control to the loaded program code.

• In many OS, the loader is permanently resident in memory.

**Text Editor**

• A text editor is tool that allows a user to create and revise documents in a computer.

• Depending on how editing is performed and the type of output that can be generated, editors are of different types, such as:

o Line Editor(edlin editor in MS DOS system)

o Word Editor(Microsoft wordpad, Notepad)

o Screen Editor(Emacs in UNIX system)

Text editors provides some salient features such as:

o Interactive user interface

o Appropriate format for storing the document in file

o Efficient transfer of information between user interface and the file in secondary storage

**Debugger**

• Debugging means locating and removing bugs, ie faults in programs.

* 1. • The main steps that are taken in debugging are:
  2. o To examine the flow of control during the execution of the program.
  3. o Examine values of variables at different points in the program.
  4. Examine the values of parameters passed to functions and values returned by functions.
  5. o Examine function call sequence.

• A debugger provides an interactive interface to the programmer to control the execution of the program.

• One of the main operations supported by the debugger is setting Breakpoints at various positions in the program. The breakpoints are points in the program at which the programmer wishes to suspend normal execution of program and perform other tasks.

**Device Driver**

• A device driver is a computer program that operates or controls a particular type of device that is attached to the computer.

• A driver provides a software interface to hardware devices, so that the OS and other computer programs can access the peripherals devices without knowing the precise details of the hardware being used.

• Drivers are hardware dependent and OS specific.

• Eg: keyboard, printer, scanner, digital camera, external storage devices etc.

• Device driver acts as a translator between the OS and the devices connected to it.

• For most of the devices, the necessary drivers are built into the OS.

• When you plug in a device, the OS starts looking for the right driver installs it and ready to start using the device. This is referred to as plug-and-play.



**Compiler**

• It is a program that translates a high level language into a machine language.



Compiler is more intelligent than an assembler.

Compiler checks all kinds of limits, ranges, errors etc.

Its program run time is more and occupies a larger part of memory. .

Since the compiler goes through the entire program and then translates the entire program in to machine codes, it has slow speed.

Eg:C program, C++, C# etc.

**Interpreter**

• It is a program that translates statements of a program into machine code.

• It translates only one statement of a program at a time. It reads only one statement, translate it and execute it. Then it reads the next statement and do the same till all the statements are translated and executed.

• The main difference with that of compiler is, compiler goes through the entire program and then translate the entire program into machine codes.

• So compiler is 5 to 25 times faster than interpreter.

* An interpreter is a small program compared to compiler. It occupies less memory space, so it can be used in a smaller system which has very limited memory space.

• By compiler, the machine codes are saved permanently for future references. On the other hand, machine codes produced by interpreter are not saved.

**Database Management System**

• A DBMS is a system software for creating and managing databases.

• DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.

• DBMS serves as an interface between the database and end users or application programs.

• DBMS is useful for providing a centralized view of data that can be accessed by multiple users from multiple locations in a controlled manner.

• Eg: MS Access, Oracle, MySQL

**Operating System**

• It act as an interface between the users and the system.

* 1. • The functions of OS are:
  2. o Process management
  3. o Memory management
  4. o Resource management
  5. o I/O operations
  6. o Data management
  7. o Providing security to user’s job.
  8. • An OS is a system software that manages computer hardware and software resources and provides common services for computer programs.
  9. • OS is a component of system software in a computer system. Application programs usually require an OS to function.
  10. • OS acts as an intermediary between programs and the computer hardware.
  11. • Eg: UNIX, Windows, iOS, Android, Fedora etc.

**The Simplifies Instructional Computer(SIC)**

• SIC is a hypothetical computer system.

• It is similar to a typical microcomputer.

• This machine has been designed to include the hardware features and concepts found on real machines, while avoiding unusual or irrelevant complexities.

• SIC comes in two versions

o The standard model

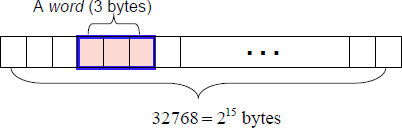
o An XE version (“Extra Equipment” or “Extra Expensive”)

The two versions have been designed to be upward compatible and an object program for the standard SIC machine will also execute properly on a SIC/XE system

* **SIC & SIC/XE Architecture**
  + **Memory**

# SIC

* + - * Memory consists of 8-bit
      * Any 3 consecutive bytes form a word (1 word=24 bits)
      * All addresses on SIC are byte addresses.
      * Words are addressed by the lowest number byte
      * Maximum memory size = 215 bytes = 32768 bytes = 32KB

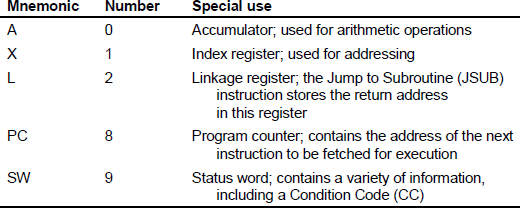


# SIC\XE

* + - * Memory consists of 8-bit
      * Any 3 consecutive bytes form a word (1 word=24 bits)
      * All addresses on SIC are byte addresses.
      * Words are addressed by the lowest number byte
      * Maximum memory size = 220 bytes =1048576bytes = 1 Megabyte
  + **Register**

# SIC

* + - * Five registers
      * Each register is 24 bits in length



* + - * SIC does not have any stack. It uses the linkage register to store the return address. It is difficult to write the recursive program. A programmer has to maintain memory for return addresses when he writes more than one layer of function call.

# SIC\XE

* + - * Total nine registers
      * Each register is 24 bits in length except F



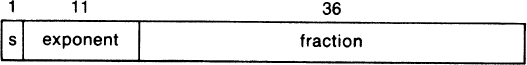
* + **Data Formats**

# SIC

* + - * Characters
        + 8-bit ASCII codes
      * Integers
        + 24-bit binary numbers
        + 2’s complement for negative values

# SIC\XE

* + - * Characters
        + 8-bit ASCII codes
      * Integers
        + 24-bit binary numbers
        + 2’s complement for negative values
      * Floating-point number
        + 48 bit



* + - * + s : Sign bit ( 0-positive, 1-negative)
        + exponent (e): unsigned binary number between 0 and 2047
        + fraction (f): a value between 0 and 1

The higher order bit of fraction must be 1

o value = (-1)s x 0.f x 2(e-1024)

* + - * + 0 is represented by setting all bits to 0
  + **Instruction Formats**

# SIC

* + - * 24-bits format



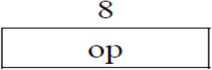
* + - * The flag bit ***x*** is used to indicate addressing modes

# SIC\XE

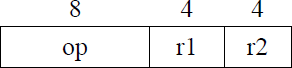
* + - * Since the memory used by SIC/XE may be 220 bytes, the instruction format of SIC is not enough.
        + Solutions

Use relative addressing

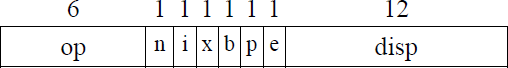
Extend the address field to 20 bits

* + - * 4 types of instructions.
        + Format 1 (1 byte)

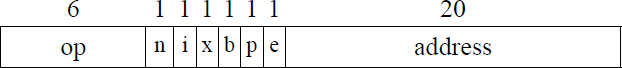
Eg: RSUB - Return to subroutine

* + - * + Format 2 (2 bytes)

Eg: COMPR A,S -Compare the contents of A and S

* + - * + Format 3 (3 bytes)

Eg: LDA #3 - Load 3 to register A

* + - * + Format 4 (4 bytes)

Eg: +JSUB RDREC - Jump to the address in RDREC

* + - * Formats 1 and 2 do not refer memory at all
      * Both format 3 and format 4 have six-bit flag values:
        + **n** : Indirect addressing flag
        + **i** : Immediate addressing flag
        + **x** : Indexed addressing flag
        + **b** : Base address-relative flag
        + **p** : Program counter-relative flag
        + **e** : Format 4 instruction flag

e=0 for format 3

e=1 for format 4

* + **Addressing Modes**

# SIC

* + - * There are two types of addressing modes
        + Direct-addressing mode
        + Indexed-addressing mode



* + - * (X) represents the contents of reg X
      * Eg:
        + Direct addressing mode

LDA TEN

Effective address = 1000

Content of address 1000 is loaded to Accumulator.

* + - * + Indexed addressing mode

STCH BUFFER, X

Effective Address = 1000+(X)

= 1000 + contents of the index register X

# SIC\XE

* + - * The following addressing modes are available
        + Direct-addressing mode
        + Indexed-addressing mode



Direct addressing mode: b=0, p=0, x=0

Format 3: TA= disp

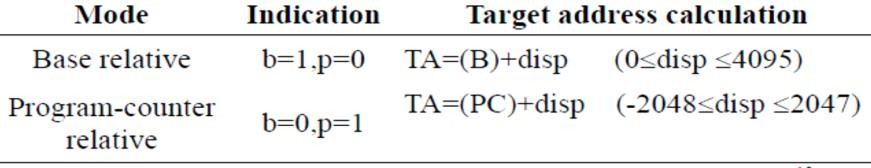
Format 4: TA=address

Indexed addressing mode: b=0, p=0, x=1

* + - * + Relative addressing mode: used in format 3

Base relative

Program Counter Relative



* + - * + Simple addressing mode: n=0,i=0 (or n=1,i=1)

TA is the location of the operand (TA=disp)

* + - * + Immediate addressing mode: n=0,i=1

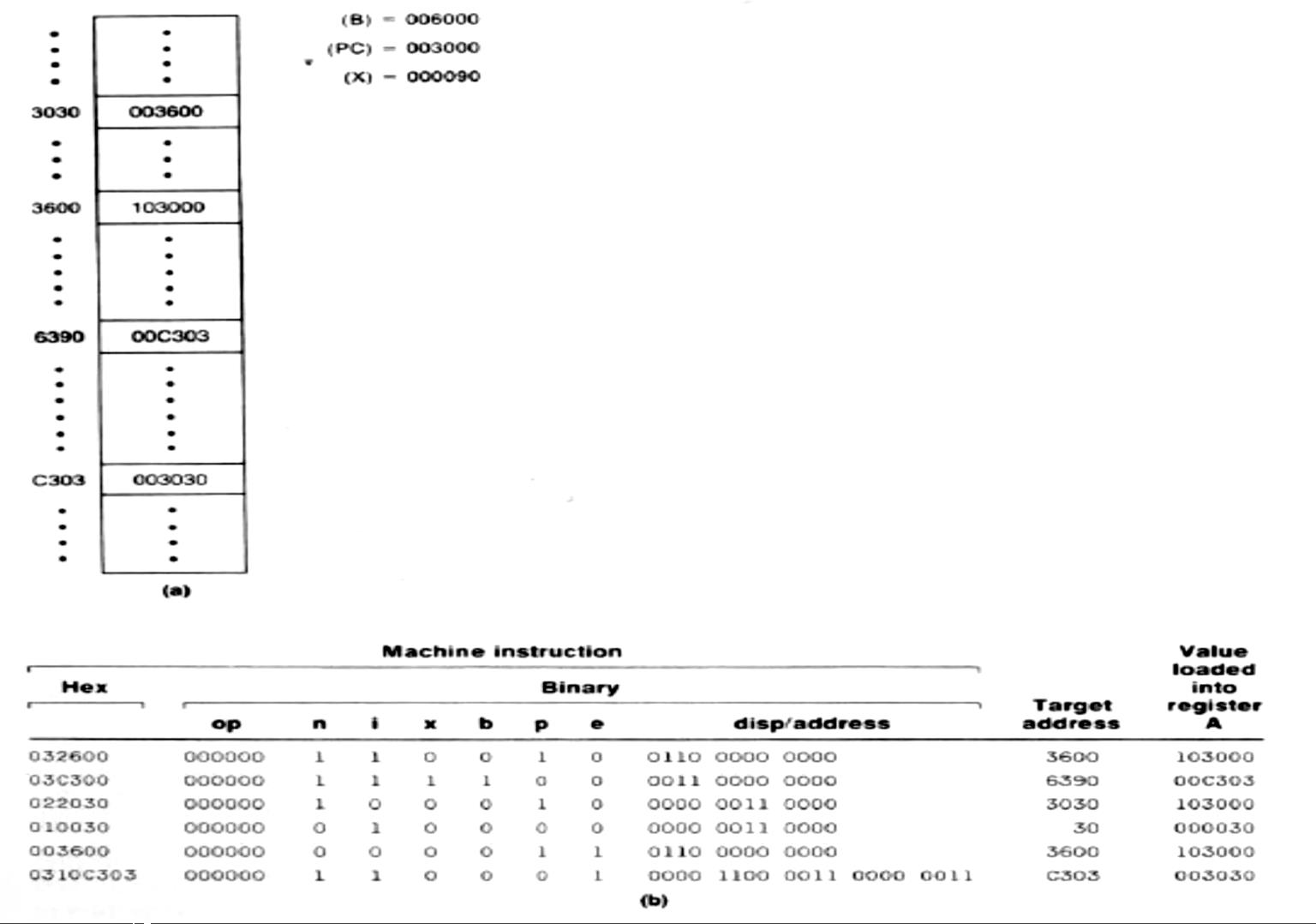
Target address is itself is used as the operand value

Eg: LDA #9

* + - * + Indirect addressing mode: n=1,i=0

The 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.

* + - * + Index + Base relative: x=1,b=1,p=0 TA=(B)+disp+(X)
        + Index + PC relative: x=1, b=0, p=1 TA=(PC)+disp+(X)
        + Index + Direct: x=1, b=0, p=0 TA=disp+(X)



* + **Instruction Set**

# SIC

* + - * SIC provides 26 instructions
        + Load and store registers: LDA, LDX, STA, STX.
        + Integer arithmetic operations: ADD, SUB, MUL, DIV

All arithmetic operations involve register A and a word in memory, with the result being left in A

* + - * + Compare Instruction: COMP

Involves register A and a word in memory.

This instruction sets a condition code (CC) of SW to indicate the result(<,>,==)

* + - * + Conditional jump instructions: JLT, JEQ, JGT

Test the setting of CC, and jump accordingly

* + - * + Subroutine linkage

JSUB: Jumps to the subroutine and places the return address in register L

RSUB: Returns to the address in L

# SIC\XE

* + - * SIC/XE provides 59 instructions. It is divided into 9 categories.
        + Load and store registers: LDA, LDX, STA, STX, LDB, STB
        + Integer arithmetic operations: ADD, SUB, MUL, DIV

All arithmetic operations involve register A and a word in memory, with the result being left in A

* + - * + Floating-point arithmetic operations

Eg: ADDF, SUBF, MULF, DIVF

* + - * + Register move instruction: Eg: RMO
        + Register-to-register arithmetic operations

Eg: ADDR, SUBR, MULR, DIVR

* + - * + Compare Instruction: COMP

Involves register A and a word in memory.

This instruction sets a condition code (CC) of SW to indicate the result(<,>,==)

* + - * + Conditional jump instructions: JLT, JEQ, JGT

Test the setting of CC, and jump accordingly

* + - * + Subroutine linkage

JSUB: Jumps to the subroutine and places the return address in register L

RSUB: Returns to the address in L

* + - * + Supervisor call instruction: generates an interrupt for OS

Eg: SVC

* + **Input and Output**

# 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-bits code
      * Three I/O instructions
        + Test Device (TD) instruction

Tests whether the addressed device is ready to send or receive a byte of data. The conditional code(CC) is set to indicate the result of this test.

CC : < means device is ready

CC : = means device is not ready

* + - * + Read Data (RD)
        + Write Data (WD)

# SIC\XE

* + - * 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-bits code
      * Three I/O instructions
        + Test Device (TD) instruction

Tests whether the addressed device is ready to send or receive a byte of data. The conditional code(CC) is set to indicate the result of this test.

CC : < means device is ready

CC : = means device is not ready

* + - * + Read Data (RD)
        + Write Data (WD)
      * There are I/O channels that can be used to perform input and output while the CPU is executing other instructions
        + SIO: start the operation of I/O channels
        + TIO: test the operation of I/O channels
        + HIO: halt the operation of I/O channels

**Assembler**

Source Program Object Code

ASSEMBLE

* Assembler translates mnemonic operation codes to their machine language equivalents
* It also assigns machine addresses to symbolic labels

**Assembler Directives (Psuedo Instructions)**

* They provide instructions to the assembler itself
* They are not Translated into machine instructions
* SIC Assembler directives are:
  + START : Specify name & starting address of the program
  + END : Indicate the end of the source program and specify the first executable instruction in the program.
  + BYTE : Generate character\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 a data area
  + RESW : Reserve the indicated no of words for a data area