CST 305- SYSTEM SOFTWARE

MODULE 1

COURSE OUTCOMES

Prerequisite: A sound knowledge in Data Structures, and Computer Organization

Course Outcomes: After the completion of the course the student will be able to

CO#	Course Outcomes
CO1	Distinguish **softwares** into system and application software categories. **(Cognitive Knowledge Level: Understand)**
CO2	Identify **standard** and extended architectural features of machines. **(Cognitive Knowledge Level: Apply)**
CO3	Identify machine dependent features of system software **(Cognitive Knowledge Level: Apply)**
CO4	Identify machine independent features of system software. **(Cognitive Knowledge Level: Understand)**
CO5	Design **algorithms** for system softwares and analyze the effect of data structures. **(Cognitive Knowledge Level: Apply)**
CO6	Understand the features of device drivers and editing & debugging tools.**(Cognitive Knowledge Level: Understand)**

SYLLABUS

Module-1 (Introduction)

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.

Module-2 (Assembly language programming and Assemblers)

SIC/XE Programming, Basic Functions of Assembler, Assembler Output Format – Header, Text and End Records. Assembler Data Structures, Two Pass Assembler Algorithm, Hand Assembly of SIC/XE Programs.

Module-3 (Assembler Features and Design Options)

Machine Dependent Assembler Features-Instruction Format and Addressing Modes, Program Relocation. Machine Independent Assembler Features –Literals, Symbol Defining Statements, Expressions, Program Blocks, Control Sections and Program Linking. Assembler Design Options- One Pass Assembler, Multi Pass Assembler. Implementation Example-MASM Assembler.

Module-4 (Loader and Linker)

Basic Loader Functions - Design of Absolute Loader, Simple Bootstrap Loader. Machine Dependent Loader Features- Relocation, Program Linking, Algorithm and Data Structures of Two Pass Linking Loader. Machine Independent Loader Features -Automatic Library Search, Loader Options. Loader Design Options.

Module-5 (Macro Preprocessor ,Device driver, Text Editor and Debuggers)

Macro Preprocessor - Macro Instruction Definition and Expansion, One pass Macro processor Algorithm and data structures, Machine Independent Macro Processor Features, Macro processor design options. Device drivers - Anatomy of a device driver, Character and block device drivers, General design of device drivers. Text Editors- Overview of Editing, User Interface, Editor

MODULE 1

- **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**

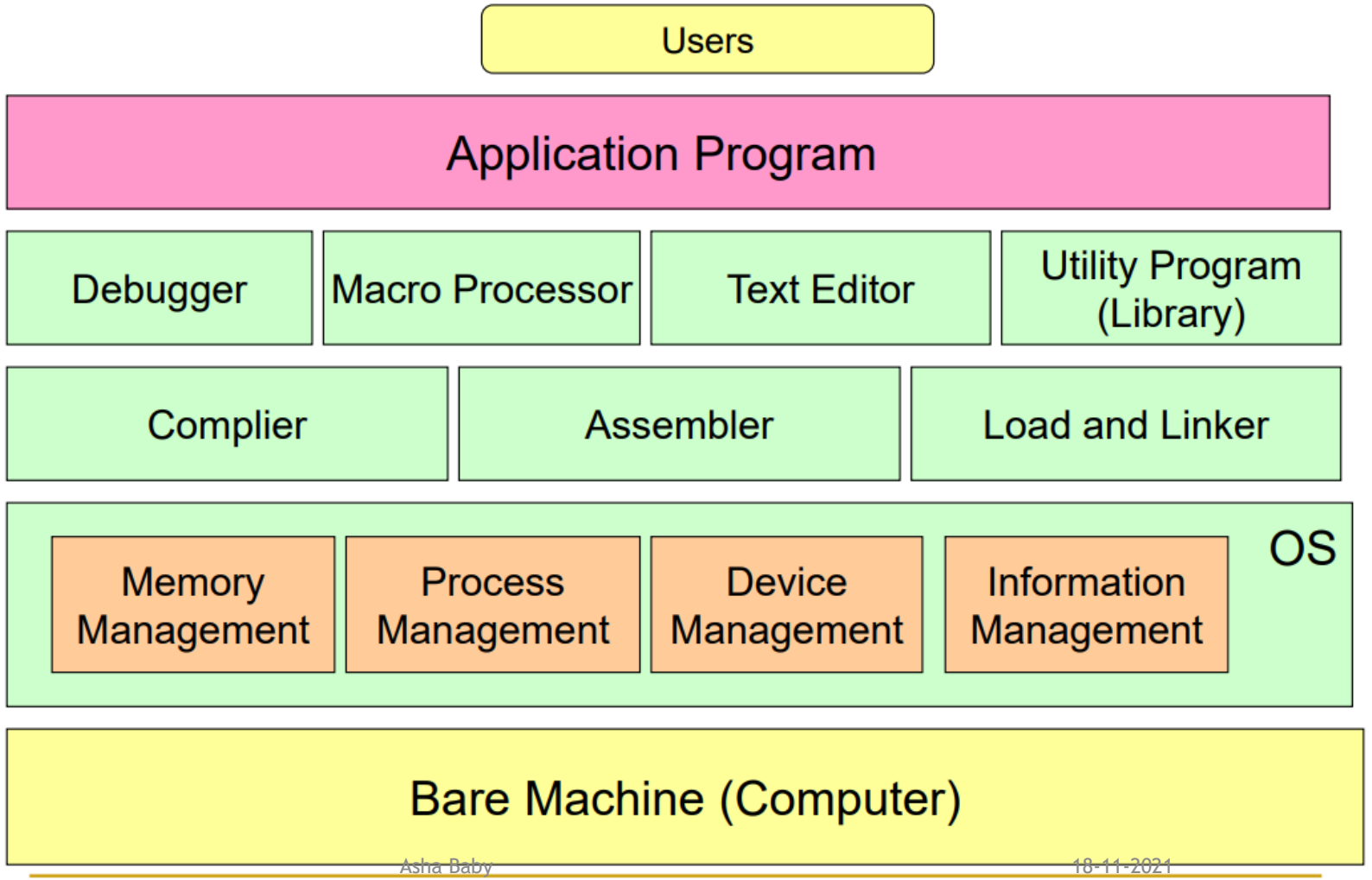
Addressing modes, Instruction set

- **Assembler Directives.**

System Software

- System Software is a set of programs that control and manage the operations of computer hardware.
- It also helps application programs to execute correctly.
- System Software are designed to control the operation and extend the processing functionalities of a computer system.
- System software makes the operation of a computer more fast, effective, and secure.

System Software Concept

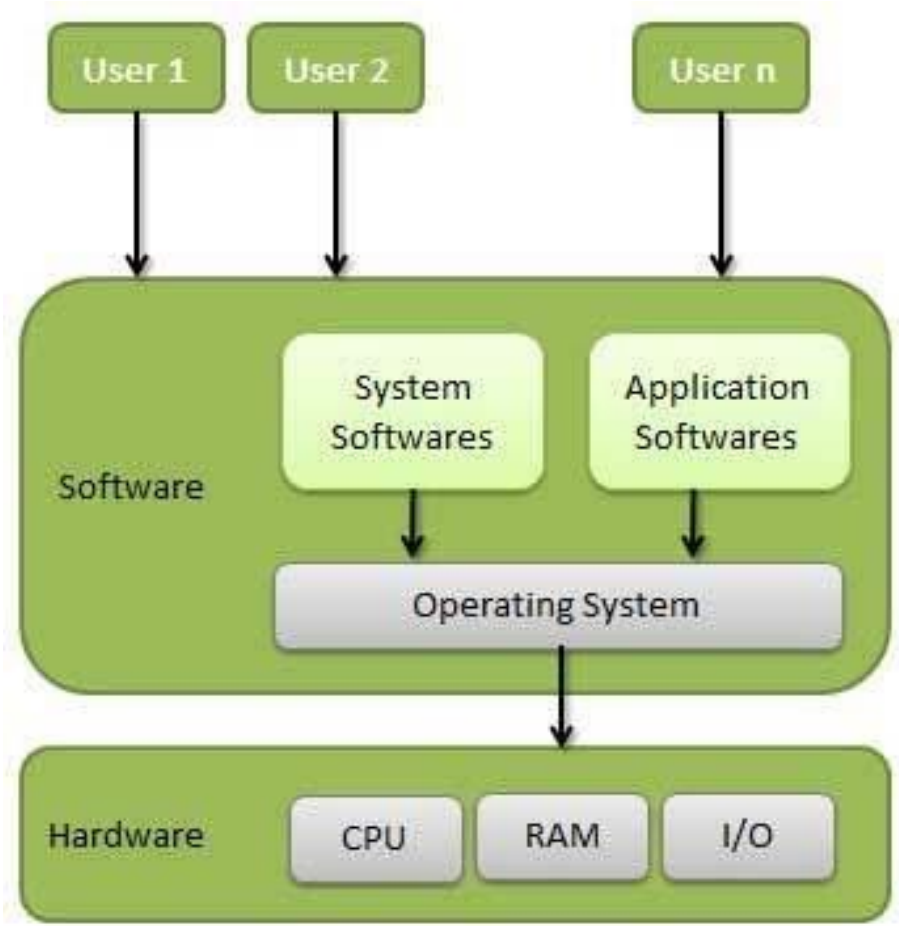


Operating System (OS)

- An **operating system** is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

- An operating system is a software which performs all the basic tasks **like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.**

- Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.


User 1
User 2
User n
Software
System Softwares
Application Softwares
Operating System
Hardware
CPU
RAM
I/O

Functions of an operating System

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Programming language translators

- Transforms the instructions prepared by developers in a programming language into a form that can be interpreted or compiled and executed by a computer system.

- Compiler
- Assembler
- Interpreter

- The high-level language is converted into binary language in various phases.

- A **compiler** is a program that converts high-level language to assembly language.
- Similarly, an **assembler** is a program that converts the assembly language to machine-level language.
- The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.
- Let us first understand how a program, using C compiler, is executed on a host machine.

- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program (low-level language).
- An assembler then translates the assembly program into machine code (object).

Interpreter

- An interpreter, like a compiler, translates high-level language into low-level machine language.
- The difference lies in the way they read the source code or input.

Interpreter translates just one statement of the program at a time into machine code.	Compiler scans the entire program and translates the whole of it into machine code at once.
An interpreter takes very less time to analyze the source code. However, the overall time to execute the process is much slower.	A compiler takes a lot of time to analyze the source code. However, the overall time taken to execute the process is much faster.
An interpreter does not generate an intermediary code. Hence, an interpreter is highly efficient in terms of its memory.	A compiler always generates an intermediary object code. It will need further linking. Hence more memory is needed.
Keeps translating the program continuously till the first error is encountered. If any error is spotted, it stops working and hence debugging becomes easy.	A compiler generates the error message only after it scans the complete program and hence debugging is relatively harder while working with a compiler.
Interpreters are used by programming languages like Ruby and Python for example.	Compliers are used by programming languages like C and C++ for example.

Sl. No.	Key	Compiler	Assembler
1	Operation	Compiler translates high level programming language code to machine level code.	Assembler converts the assembly level language to machine level code.
2	Input	Source code in high level programming language.	Assembly level code as input.
3	Conversion type	Compiler checks and converts the complete code at one time.	Assembler generally does not convert complete code at one time.
4	Components	lexical analyzer, Syntax analyzer, Semantic analyzer, Code optimizer, Code generator, and Error handler	Assembler does works in two passes.
5	Output	Mnemonic version of machine code.	Binary version of machine code.
6	Examples	C, C++ , Java compilers.	GNU assemblers(known as GAS).

Linker

- Linker is a computer program that links and merges various object files together in order to make an executable file.
- All these files might have been compiled by separate assemblers.
- The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

Loader

- Loader is a part of operating system and is responsible for loading executable files into memory and execute them.
- It calculates the size of a program (instructions and data) and creates memory space for it.
- It initializes various registers to initiate execution.

Macro processor

- A macro instruction (macro) is a notational convenience for the programmer.
- Allow the programmer to write a shorthand version of a program
- A macro represents a commonly used group of statements in the source programming language.

.

- A macro is a name given to a block of C statements as a pre-processor directive.
- Being a pre-processor, the block of code is communicated to the compiler before entering into the actual coding (main () function).
- A macro is defined with the preprocessor directive, #define.

Expanding the macros

- The macro processor replaces each macro instruction with the corresponding group of source language statements

Advantages

- The speed of the execution of the program is the major advantage of using a macro.
- It saves a lot of time that is spent by the compiler for invoking / calling the functions.
- It reduces the length of the program.

Text editors

- **Editors or text editors** are software programs that enable the user to create and edit text files.
- In the field of programming, the term editor usually refers to source code editors that include many special features for writing and editing code.
- Notepad, Wordpad are some of the common editors used on Windows OS and vi, emacs, Jed, pico are the editors on UNIX OS.
- Features normally associated with text editors are — moving the cursor, deleting, replacing, pasting, finding, finding and replacing, saving etc.

Types of Editors

Line editor:

- In this, you can only edit one line at a time or an integral number of lines.
- You cannot have a free-flowing sequence of characters.
- It will take care of only one line.

Ex : Teleprinter, edlin, teco

Stream editors:

- In this type of editors, the file is treated as continuous flow or sequence of characters instead of line numbers, which means here you can type paragraphs.

Ex : Sed editor in UNIX

Screen editors:

- In this type of editors, the user is able to see the cursor on the screen and can make a copy, cut, paste operation easily.
- It is very easy to use mouse pointer.
  Ex : vi, emacs, Notepad

Word Processor:

- Overcoming the limitations of screen editors, it allows one to use some format to insert images, files, videos, use font, size, style features.
- It majorly focuses on Natural language.

Structure Editor:

- Structure editor focuses on programming languages.
- It provides features to write and edit source code.
  Ex : Netbeans IDE, gEdit.

Device Driver

- It refers to a special kind of software program or a specific type of software application which **controls a specific hardware device** that **enables different hardware devices for communication with the computer's Operating System.**

- A device driver communicates with the computer hardware by computer subsystem or computer bus connected to the hardware.

- **Device Drivers** are very essential for a computer system to work properly because without device driver the particular **hardware fails to work accordingly** means it fails in doing a particular function/action for which it has been created.

Communication Software

- Communication software allows us to transfer data and programs from one computer system to another.
- Examples : file transfer protocol (FTP), messaging software and email

Utility programs

- Utility programs are a set of programs that help users in system maintenance tasks, and in performing tasks of routine nature.
- Examples : Antivirus, Compression tools, Disk Management tools, Disk cleanup tool, Backup utility.

Debuggers

- ***Debugging*** means locating (and then removing) *bugs*, i.e., faults, in programs.

- In the entire process of program development errors may occur at various stages and efforts to detect and remove them may also be made at various stages.

- The most common steps taken in debugging are to examine the **flow of control** during execution of the program, examine **values of variables** at different points in the program, examine **the values of parameters passed to functions and values returned** by the functions, examine the function call sequence, etc

- A debugger is a computer program used by programmers to test and debug a target program.
- Debuggers may use instruction-set simulators, rather than running a program directly on the processor to achieve a higher level of control over its execution.
- This allows debuggers to stop or halt the program according to specific conditions. However, use of simulators decreases execution speed.

- When a program crashes, debuggers show the position of the error in the target program.
- Most debuggers also are capable of running programs in a step-by-step mode, besides stopping on specific points.
- They also can often modify the state of programs while they are running.

Examples

- Arm DTT, formerly known as Allinea DDT.
- Eclipse debugger API used in a range of IDEs: Eclipse IDE (Java) Nodeclipse (JavaScript)
- Firefox JavaScript debugger.
- GDB - the GNU debugger.
- LLDB.
- Microsoft Visual Studio Debugger.
- Radare2.
- TotalView.

Features of System Software

- System Software is closer to the system
- Generally written in a low-level language
- The system software is difficult to design and understand
- Fast in speed
- Less interactive
- Smaller in size
- Hard to manipulate

Application Software

- Application Software is a program that does real work for the user.
- It is mostly created to **perform a specific task** for a user.
- Application Software acts as a mediator between the end-user and System Software.
- It is also known as an application package.
- This type of software is written using a high-level language like C, Java, VB. Net, etc.

- It is a user-specific and is designed to meet the requirements of the user.
- You can also install multiple Application Software on a single System Software.
- You can store this kind of software on CDs, DVDs, flash drive, or keychain storage devices.
- Example: Word-processing, Spreadsheet, Database, etc.

Types of Application Software

- **Word-processing software:-** It makes use of a computer for creating, modifying, viewing, storing, retrieving, and printing documents.
- **Spreadsheet software:-** Spreadsheet software is a numeric data-analysis tool that allows you to create a computerized ledger.
- **Database software:-** A database software is a collection of related data that is stored and retrieved according to user demand.

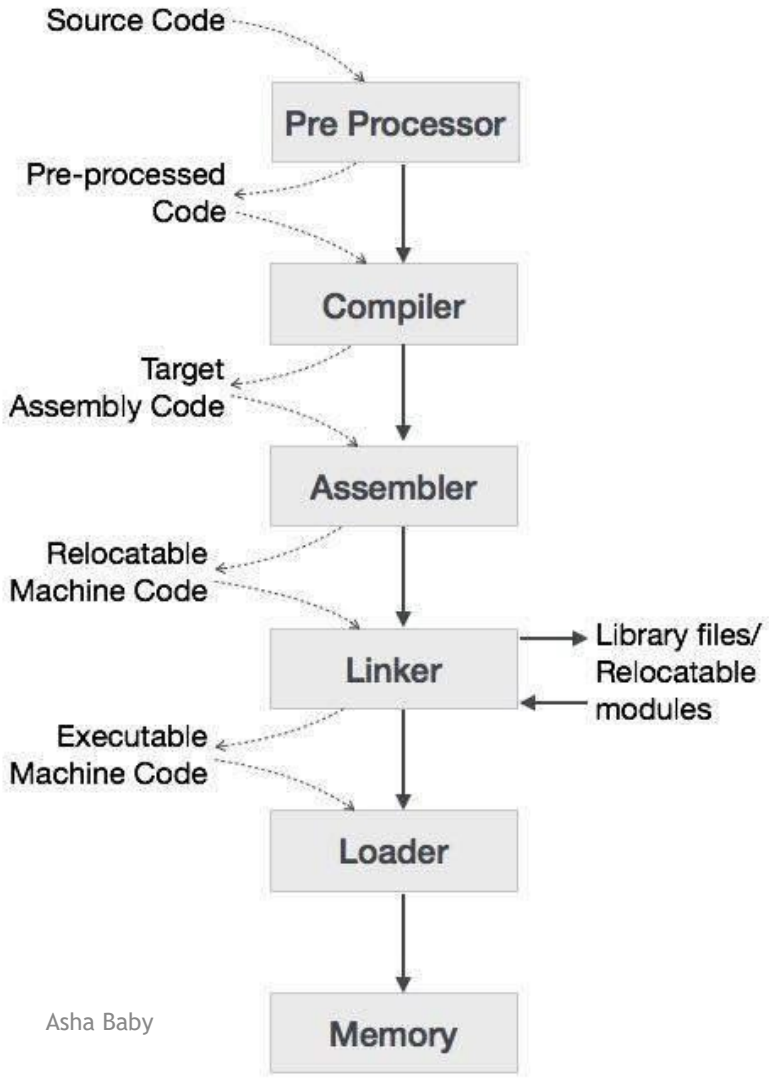
- **Graphics software:-** It allows computer systems for creating, editing, drawings, graphs, etc.
- **Education software:-** Education software allows a computer to be used as a learning and teaching tool.
- **Entertainment software:-** This type of app allows a computer to be used as an entertainment tool.

Features of Application Software

- Perform more specialized tasks like word processing, spreadsheets, email, photo editing, etc.
- It needs more storage space as it is bigger in size
- Easy to design and more interactive for the user
- Generally written in a high-level language

S.No.	System Software	Application Software
1.	System software is used for operating computer hardware.	Application software is used by user to perform specific task.
2.	System softwares are installed on the computer when operating system is installed.	Application softwares are installed according to user's requirements.
3.	In general, the user does not interact with system software because it works in the background.	In general, the user interacts with application sofwares.
4.	System software can run independently. It provides platform for running application softwares.	Application software can't run independently. They can't run without the presence of system software.
5.	Some examples of system softwares are compiler, assembler, debugger, driver, etc.	Some examples of application softwares are word processor, web browser, media player, etc.

Language Processing System



- Computers are a balanced mix of software and hardware.
- Hardware is just a piece of mechanical device and its functions are being controlled by a compatible software.
- Hardware understands instructions in the form of electronic charge, which is the counterpart of binary language in software programming.
- Binary language has only two alphabets, 0 and 1.
- To instruct, the hardware codes must be written in binary format, which is simply a series of 1s and 0s.
- It would be a difficult and cumbersome task for computer programmers to write such codes, which is why we have compilers to write such codes.

- We have learnt that any computer system is made of hardware and software.
- The hardware understands a language, which humans cannot understand.
- So we write programs in high-level language, which is easier for us to understand and remember.
- These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine.
- This is known as Language Processing System.

- The high-level language is converted into binary language in various phases.
- A **compiler** is a program that converts high-level language to assembly language.
- Similarly, an **assembler** is a program that converts the assembly language to machine-level language.
- Let us first understand how a program, using C compiler, is executed on a host machine.
- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program (low-level language).

- An assembler then translates the assembly program into machine code (object).
- A linker tool is used to link all the parts of the program together for execution (executable machine code).
- A loader loads all of them into memory and then the program is executed.
- Before diving straight into the concepts of compilers, we should understand a few other tools that work closely with compilers.

Preprocessor

- A preprocessor, generally considered as a part of compiler, is a tool that produces input for compilers.
- It deals with macro-processing, augmentation, file inclusion, language extension, etc.

Interpreter

- An interpreter, like a compiler, translates high-level language into low-level machine language.
- The difference lies in the way they read the source code or input.

- A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes.

- In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence.

- If an error occurs, an interpreter stops execution and reports it. whereas a compiler reads the whole program even if it encounters several errors.

Simplified Instructional Computer(SIC).

- Most system software is **machine dependent,** we must include real machines and real pieces of software in our study.
- But here we present the fundamental functions of each piece of software through discussion of a Simplified Instructional Computer (SIC).
- SIC is a **hypothetical computer** that has been carefully designed to include the hardware features most often found on real machines, while avoiding unusual or irrelevant complexities.

1.3 The Simplified Instructional Computer

- Like many other products, SIC comes in two versions
  - The standard model
  - An XE version
    - “extra equipments”, “extra expensive”
- The two versions has been designed to be upward compatible
- SIC (Simplified Instructional Computer)
- SIC/XE (Extra Equipment)

1.3 The Simplified Instructional Computer

- SIC
  - Upward compatible
  - Memory consists of 8-bit bytes, 3 consecutive bytes form a word (24 bits)
  - There are a total of 32768 bytes (32 KB) in the computer memory.
  - 5 registers, 24 bits in length
    - A 0 Accumulator
    - X 1 Index register
    - L 2 Linkage register (JSUB)
    - PC 8 Program counter
    - SW 9 Status word (Condition Code)

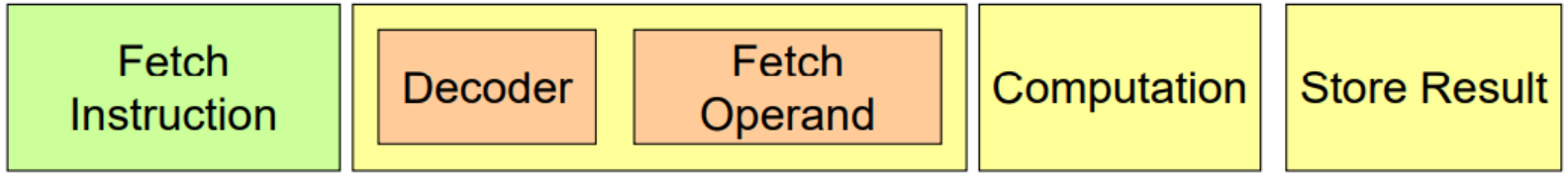
- **Accumulator(A)** is a special purpose register used for arithmetic operations.
- **Index register(X)** is used for addressing.
- **Linkage register(L)** stores the return address of the jump of subroutine instructions (JSUB).
- **Program counter(PC)** contains the address of the current instructions being executed.
- **Status word(SW)** contains a variety of information including the condition code.

1.3.1 SIC Machine Architecture

- **Data Formats**
  - Integers are stored as 24-bit binary number
  - 2's complement representation for negative values
  - Characters are stored using 8-bit ASCII codes
  - No floating-point hardware on the standard version of SIC

Instruction Cycle

- **CPU**
  - **Control Unit (CU)**
  - **Arithmetic and Logic Unit (ALU)**
  - **Register**
- **Instruction Cycle**
  - **Fetch Cycle**
  - **Execution Cycle**



1.3.1 SIC Machine Architecture

- **Instruction format**
  - 24-bit format
  - The flag bit x is used to indicate indexed-addressing mode

- **Addressing Modes**
  - There are two addressing modes available
    - Indicated by x bit in the instruction
    - (X) represents the contents of reg. X

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

1.3.1 SIC Machine Architecture

- Instruction set
  - Format 3
  - Load and store registers (LDA, LDX, STA, STX, etc.)
  - Integer arithmetic operations (ADD, SUB, MUL, DIV)
  - Compare instruction (COMP)
  - Conditional jump instructions (JLT, JEQ, JGT)
  - JSUB jumps to the subroutine, placing the return address in register L.
  - RSUB returns by jumping to the address contained in register L.

1.3.1 SIC Machine Architecture

- I/O
  - I/O 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 as an operand.
  - Test Device (TD): tests whether the addressed device is ready to send or receive
    - < ready = not ready
  - Read Data (RD)
  - Write Data (WD)

1.3.2 SIC/XE Machine Architecture

- 1 megabytes (1024 KB) in memory
- 3 additional registers, 24 bits in length
  - B 3 Base register; used for addressing
  - S 4 General working register
  - T 5 General working register
- 1 additional register, 48 bits in length
  - F 6 Floating-point accumulator (48 bits)

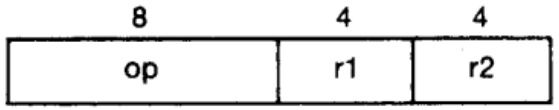
1.3.2 SIC/XE Machine Architecture

- **Data format**
  - 24-bit binary number for integer, 2's complement for negative values
  - 48-bit floating-point data type
  - The exponent is between 0 and 2047
  - $f*2^{(e-1024)}$
  - 0: set all bits to 0

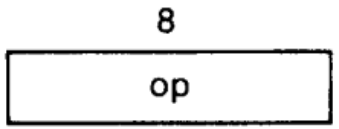
1.3.2 SIC/XE Machine Architecture

- **Instruction formats**
  - Relative addressing (相對位址) - format 3 (e=0)
  - Extend the address to 20 bits - format 4 (e=1)
  - Don't refer memory at all - formats 1 and 2

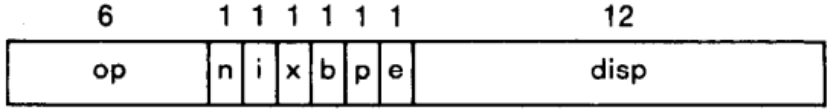
Format 2 (2 bytes):



Format 1 (1 byte):



Format 3 (3 bytes):



Format 4 (4 bytes):

1.3.2 SIC/XE Machine Architecture

- Addressing modes
  - n i x b p e
  - Simple n=0, i=0 (SIC) or n=1, i=1
  - Immediate n=0, i=1 TA=Valus
  - Indirect n=1, i=0 TA=(Operand)
  - Base relative b=1, p=0 TA=(B)+disp

    0 <= disp <= 4095
  - PC relative b=0, p=1 TA=(PC)+disp

    -2048 <= disp <= 2047

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)

1.3.2 SIC/XE Machine Architecture

- Addressing mode
  - Direct — b=0, p=0 — TA=disp
  - Index — x=1 — $TA_{new}=TA_{old}+(X)$
  - 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
  - Format 4 — e=1
- Appendix and Fig. 1.1 Example

Figure 1.1

(B) = 006000

(PC) = 003000

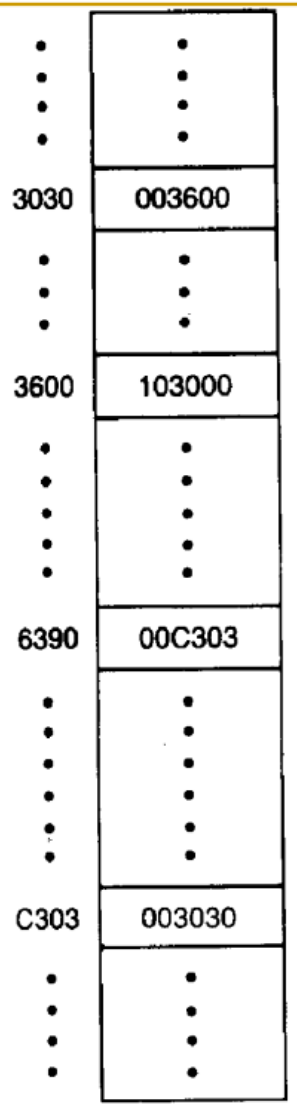
(X) = 000090

- Memory address
  - 00000

(0000 0000 0000 0000 0000)

  - ~FFFFF (Byte)

(1111 1111 1111 1111 1111)



(a)

(B) = 006000

(PC) = 003000

(X) = 000090

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
030300	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
03100303	000000	1	1	0	0	0	1	0000 1100 0011 0000 0011	C303	003030

(b)

1.3.2 SIC/XE Machine Architecture

- Instruction set
  - Format 1, 2, 3, or 4
  - Load and store registers (LDB, STB, etc.)
  - Floating-point arithmetic operations (ADDF, SUBF, MULF, DIVF)
  - Register-to-register arithmetic operations (ADDR, SUBR, MULR, DIVR)
  - A special supervisor call instruction (SVC) is provided
- I/O
  - 1 byte at a time, TD, RD, and WD
  - SIO, TIO, and HIO are used to start, test, and halt the operation of I/O channels.

Assembler directives

- Assembler directives are **pseudo instructions**. They provide instructions to the assembler itself and they are not translated into machine operation codes.

Assembler Directives

- **Pseudo-instructions**
  - Not translated into machine instructions
  - Provide instructions to the assembler itself
- Basic assembler directives
  - **START:** specify *name* and *starting address of the program*
  - **END:** specify *end of program* and (option) *the first executable instruction in the program*
    - If not specified, use the address of the first executable instruction
  - **BYTE:** direct the assembler to *generate constants*
  - **WORD**
  - **RESB:** : instruct the assembler to *reserve memory location* without generating data values
  - **RESW**

University questions..

1. What are assembler directives? List out any five assembler directives in SIC
2. List out the various registers used in SIC along with their purpose.
3. Distinguish between Application software and System Software.
4. What are the functions of Operating System.
5. Explain the instruction format and addressing modes of SIC standard.
6. Compare compiler and assembler.
7. Write short notes on device drivers.
8. Distinguish between interpreter and compiler.

9. List out the functions of the macro processor.
10. List out and explain four examples of application software.
11. What are the functions of debuggers? briefly explain different types of debuggers.
12. Explain the role of linkers and loaders in language processing systems.
13. Write notes on the architecture of SIC/XE.
14. Write notes on SIC standard machine architecture.
15. What are the various addressing modes supported by SIC/XE? With the help of an example, explain how to find target address during assembling in each case

17. Distinguish between SIC standard and SIC XE architecture.
18. Explain the following in detail

- SIC standard and SIC XE registers.
- Data formats supported by SIC standard and XE version.
- Instruction sets provided by SIC standard and XE version.

19. Explain the following software in detail.

- Operating system
- Device drivers.
- Debuggers.
- Macro processor.