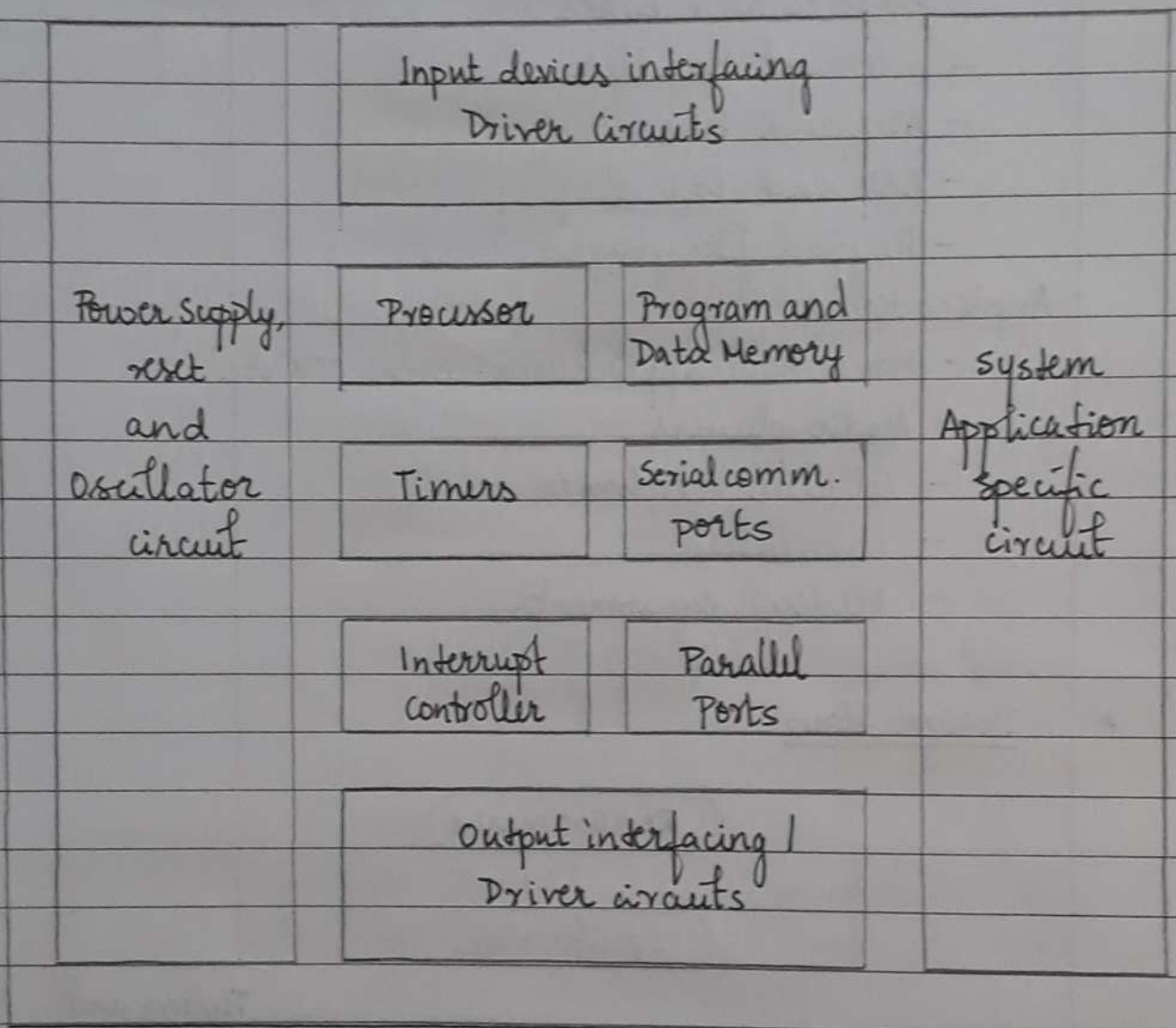
UNIT - 01

Introduction

* **components of Embedded systems**



An embedded system is one that has computer hardware with software embedded in it as one of its important components.

There are various processors

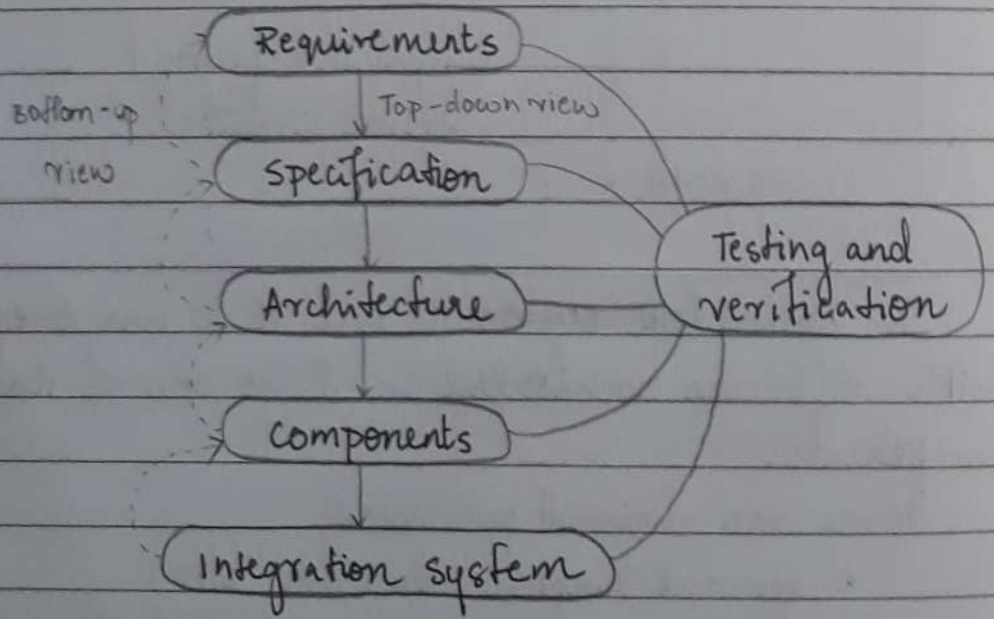
- General Purpose Processor (GPP)

(Microprocessor, microcontroller, Embedded processor, Digital Signal Process

- Application specific system processor (ASSP)
- system on chip (SOC)
- Application specific Integrated circuit (ASIC)

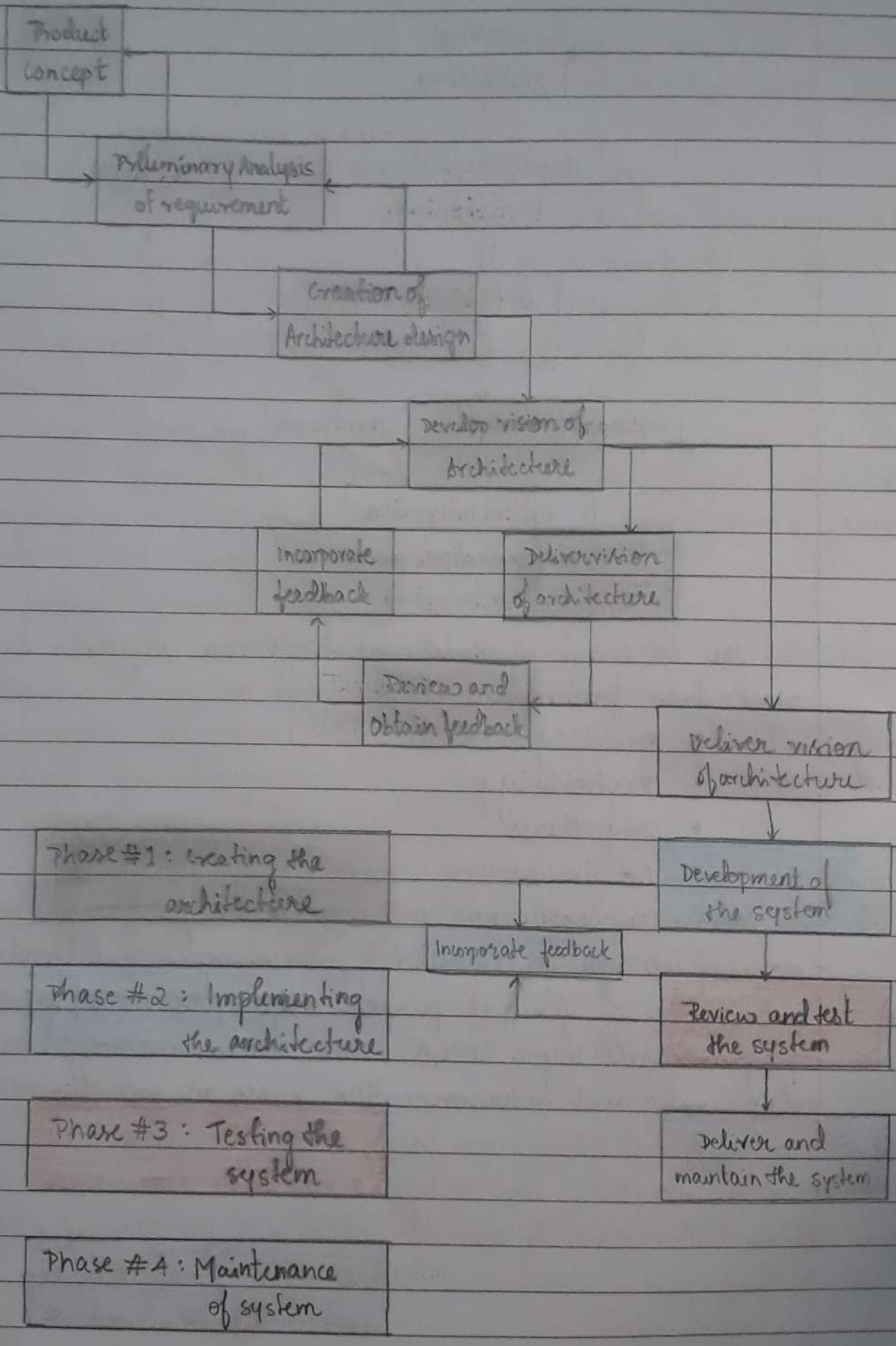
- Other Hardware
  - Power Source
  - Clock Oscillator
  - Real time clock (RTC)
  - Reset, Power-up reset and watchdog timer reset circuits
  - Memory
  - I/O ports and buses
  - Interrupt handler
  - ADC and DAC
  - LCD and LED display
  - keypad / keyboard.
- Applications.
  - House hold applications: ovens, TV etc
  - Audio players
  - cellular telephones
  - calculators
  - Medical equipments

* Design Flow:



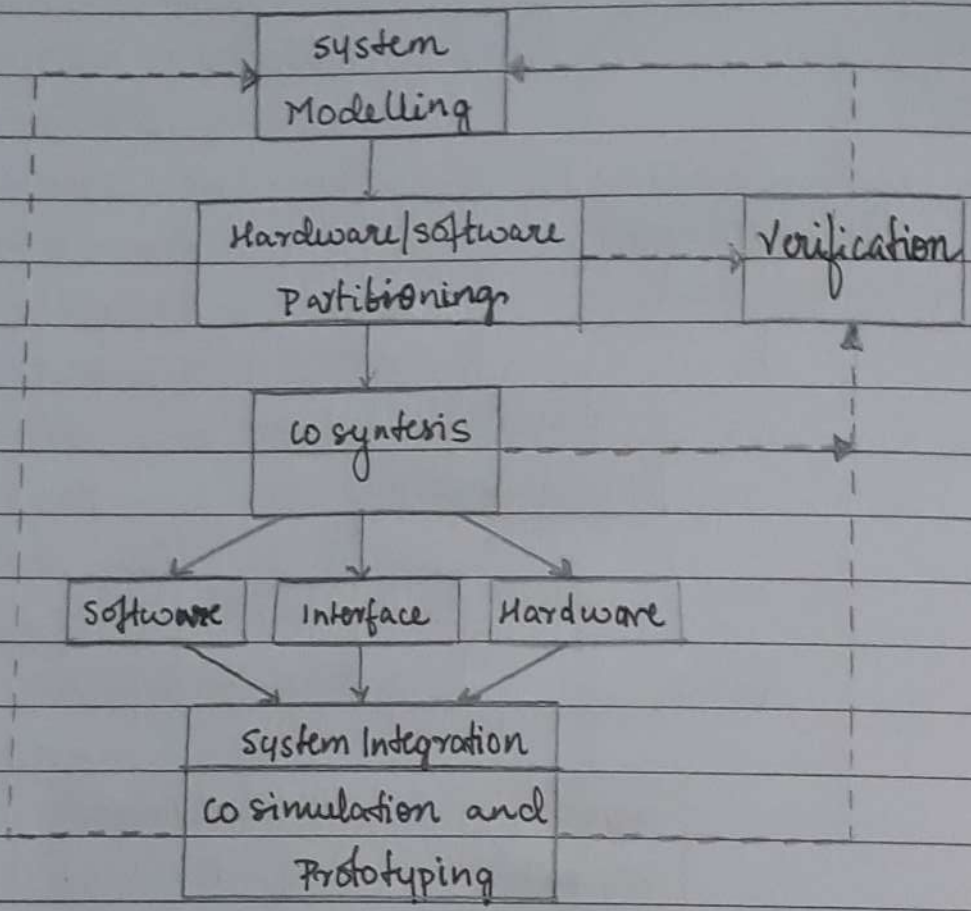
Page No.:
Date: YOUVA

* Embedded system design life cycle.



Date: YOUVA

* Hardware-Software codesign:



The co design of Hardware-Software systems may be viewed as four main phases:

- Modeling:
- Partitioning
- Cosynthesis
- Co simulation

Hardware-software codesign techniques target system on-chip design or embedded core design that involves integration of general purpose microprocessors, DSP structures, programmable logic (FPGA), ASIC cores, memory block peripherals and interconnection buses on one chip.

Page No.
Date
YOUVA

* Latest Technologies:

- Artificial Intelligence

It can be defined as the enabling of machine to perform the logical analysis, obtain knowledge and adapt to an environment that varies.

- Virtual Reality and Augmented Reality

It allows the user to interact with an environment that exists in a computer. It is a way to generate realistic images, sound and other sensations.

Augmented reality adds virtual stuff to real word environment.

- Deep Learning

It represents a rich and yet unexplored embedded systems market that has a range of applications from image processing to audio analysis.

- Embedded Security

It will emerge as crutial generators for identifying devices in an IOT network and as microcontroller security solutions that isolate security operations from normal operations.

- Cloud Connectivity

These technologies may design to simplify the process of connecting embedded systems with cloud services into reducing the underlying hardware - complexities.

* Processor:

Processor is the heart of an embedded system. It is the basic unit that takes inputs and produces an output after processing the data. It has two essential units:

- Program Flow control unit (CU)
- Execution Unit (EU)

The CU includes a fetch unit for fetching instructions from the memory. The EU has circuits that implement the instructions pertaining to data transfer operation and data conversion from one form to another.

The EU includes the ALU and also circuits that execute instructions for a program control task such as interrupt or jump to another set of instructions.

A processor runs the cycles of fetch and executes the instruction in the same sequence as they are fetched from memory.

Types of processors:

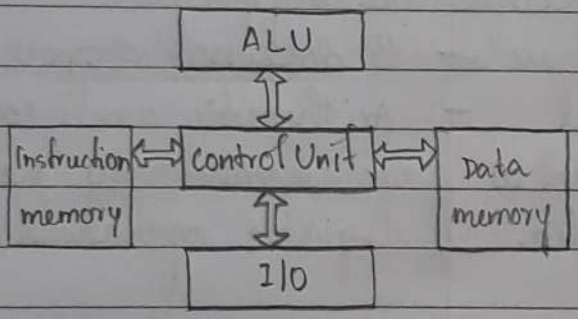
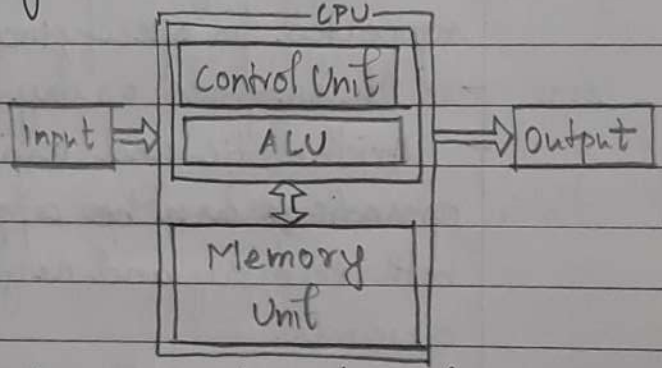
- General Purpose Processor (GPP)
  - microcontroller
  - microprocessor
  - embedded processor
  - digital signal processor
  - media processor
- Application Specific System Processor (ASSP)
- Application specific instruction Processors (ASIPs)

* Differences:

1. Microprocessor	Microcontroller
- Multitasking in nature, can perform multiple tasks at a time.	- Single task oriented. Designed to perform only specific tasks.
- RAM, ROM, I/O ports and timers can be interfaced externally.	- RAM, ROM, I/O ports and timers cannot be interfaced externally. They are already embedded on the chip.
- External interfacing makes the system heavier and costlier.	- They are light weight and cheaper that microprocessors.

Page No.:
Date:
YOUVA

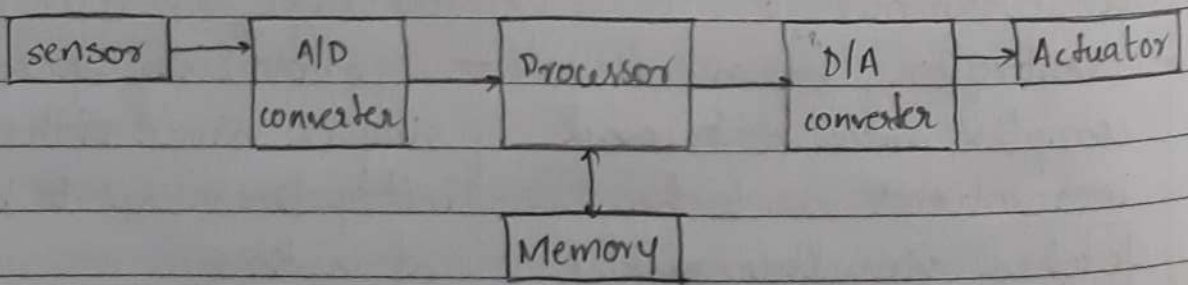
- External devices require more space and power consumption is higher.	- System consumes less power and occupies less space.

2. Harvard Architecture	Van Neumann Architecture
- The CPU is connected to both the data memory (RAM) and program memory (ROM) separately.	- There is no separate data and program memory. Instead a single memory connection is given to the CPU.
	
- Requires more hardware due to separate data and address bus for each memory.	- Requires less hardware due to common memory needs to be reached.
- Requires more space.	- Requires less space.
- Speed of execution is faster because the processor fetches data and instructions simultaneously.	- Speed of execution is slower since it cannot fetch the data and instructions at the same time.
- controlling becomes complex since data and instructions are to be fetched simultaneously.	- controlling becomes simpler since either data or instructions are to be fetched at a time

3.

CISC	RISC
- A large number of instructions are present in the architecture.	- Very fewer instructions are present in the architecture.
- some instructions are with long execution time.	- No instruction is with a long execution time due to very simple instruction set.
- Variable-length encodings of the instructions are used.	- Fixed-length encodings of the instructions are used.
- Multiple formats are supported for specifying operands.	- Simple addressing formats are supported.
- It supports array.	- It does not support array.
- Arithmetic and logical operations can be applied to both memory and register operands.	- Arithmetic and logical operations only use register operands.
- Condition codes are used.	- No condition codes are used.
- The stack is being used for procedure arguments and return addresses.	- Registers are being used for procedure arguments and return addresses.

* **Basic structure of Embedded system**



Page No.:
Date:
YOUVA

UNIT - 02

Embedded Linux System

* **Features of Linux:**
  - It is open source, anyone with programming knowledge can modify it.
  - cost effective, as source codes is freely available and its community based development project
  - Portable: The software can work on different types of hardware in same way.
  - Multiuser: Multiple users can access system resources at same time.
  - Multiprogramming: Multiple applications can run at same time.

* **Technical Reasons to use Embedded Linux:**

1. An Embedded Linux system includes software such as:
   - SSL/SSH: The open SSH project is the most commonly used encryption and security mechanism today.
   - Apache and other web servers: The Apache web server is used in embedded devices that need a full-featured web server.
   - The C library: The fully featured GNU C library.
   - Berkeley sockets (IP): Socket is a local end point of a network communication path. The Berkeley sockets provide a common interface for input and output to streams of data.
2. **STANDARD BASED**

   The Linux operating system and accompanying open source projects adhere to industry standards. In most cases the implementation available in open source is the canonical, or reference implementation of a standard.

3. PROCESS ISOLATION AND CONTROL: (Linux kernel)

- Manage tasks, isolating system resources from the kernel and each other.
- Provide a uniform interface for the systems hardware resources.
- serve as an arbiter to resources when contention exists.

* Creating Linux Distribution from scratch:

Creating Linux distribution involves these steps:

- Build a cross compiler
- Use the cross compiler to build a kernel
- Use the cross compiler to build a root file system
- Roll the root file system into something the kernel can use to boot.

* Cross compiler:

— A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.

Ex: A compiler that runs on a Linux x86 host that produces code to execute on an ARM9 target is a cross compiler.

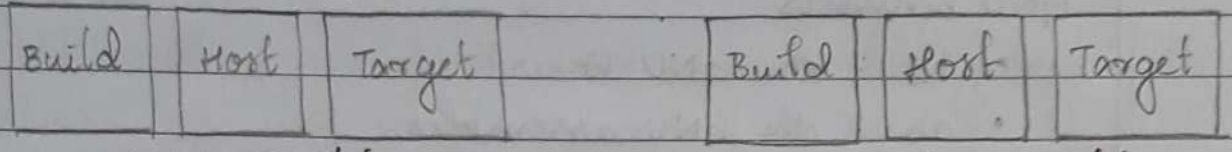
In Linux the cross compiler is referred to as a tool chain that work together to produce an executable: the compiler, assembler and linker.

— Some basic terminology used to describe the players in the process of building the compiler are:

- Build machine: The computer used to compile the code.
- Host machine: The computer where the compiler runs.
- Target machine: The computer for which gcc produces the code.

Page No.:
Date:
YOUVA

- There are two types of build:



Cross Build

It is used to build a tool chain that runs on your workstation but generates binaries for the target.

Canadian Build

It is used to build on architecture A, a toolchain that runs on architecture B and generates binaries for architecture C.
Here the build, host and target machines are all different.

- Need for cross compilers

The characteristics of most embedded systems is limited memory, diskless, no output screen, poor performance. Sometimes it is not possible to execute compiler, assembler, linker and debugger on target platform, so for embedded systems we need cross system development tools to help software development.

* Requirements for Embedded system:

- Hardware
- software (software development tools: compiler, assembler, linker, debugger and c standard library).

GNU compiler collections (GCC)

- developed by Richard Stallman
- The GNU toolchain includes:
  - GNU compiler collection: supports many languages.
  - GNU make: tool for compiling and building.
  - GNU binutils: a suit of binary utility tools including linker and assembler.
  - GNU Debugger (GDB)

- GNU autotools.

GNU binutils

- ld : the GNU linker
- as : the GNU assembler
- ar : creating, modifying and extracting from archives
- gprof: displays profiling information
- objcopy: copies and translates object files from one format to another.
- objdump: displays information from object files

* Boot loader:

It is a program first run by a computer so that a more sophisticated program can be loaded next. The following illustration describes the booting process from power-on until the first program runs:

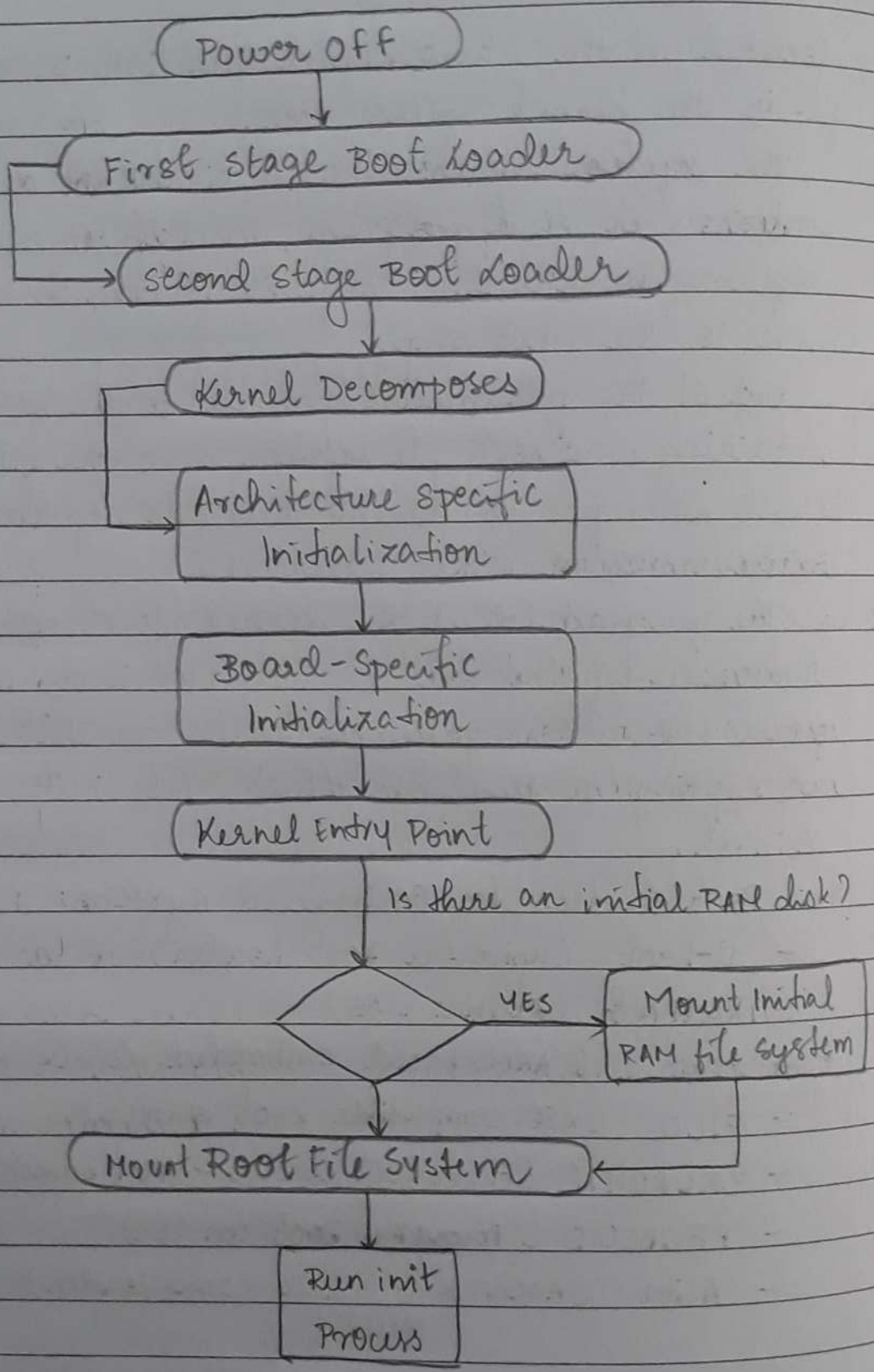
1. The first stage boot loader is the code in the processor that reads a location in memory and may put a message on the console or screen.
2. The second stage boot loader is responsible for loading what the system will run. in this case (Linux), but it can be any program.
3. If the kernel is compressed, its now uncompressed into memory. A jump instruction follows the decompression step that places the instruction at the next executable instruction.
4. Processor and board initialization runs. This low-level code performs hardware initialization. The code then reaches the kernel's entry point and processor independent code runs to get the kernel ready to run the init process.
5. The kernel entry point is the code that is in the architecture independent part of the kernel tree. This code is

located in the start_kernel function in the init/main.c file in the kernel source tree.

6. The system mounts the initial RAM disk, sees if it contains an /init program, and if so runs it. If this program doesn't exist or isn't found, the boot process moves to the next step.
7. One of the parameters passed to the kernel is the device containing the root file system and the file system type. The kernel attempts to mount this file system and panics if it isn't found or isn't mountable.
8. The program that the kernel first attempts to run is the value of the kernel parameter init. In lieu of this parameter, the system looks for an init executable by attempting to run /sbin/init, /etc/init, /bin/init and /bin.sh.

Boot Loaders for Embedded system:

- U-boot (Universal Boot Loader)
- RedBoot (RedHat eCos)
- rmload (ARM based embedded Linux systems)
- FILO (x86 compatible boot loader)
- CRL/OHH (Flash Boot Loader - ARM based embedded linux system)
- PPCBOOT (PowerPC Boot Loader)
- Alios (Assembler based Linux loader)



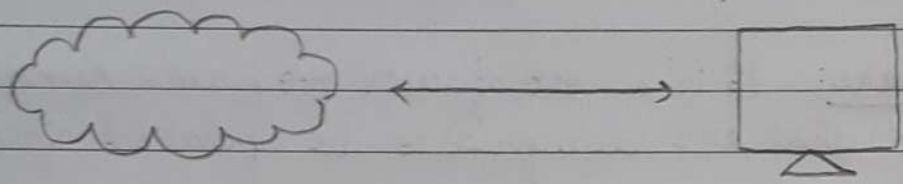
LINUX BOOT PROCESS

Page No.:
Date:

* System Design:

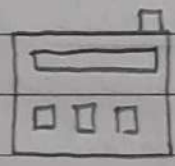
CAMPUS NETWORK
The connection to the network at the office and the internet.

DEVELOPMENT HOST
This is a desktop machine where the kernel and application are built.



Serial communication for accessing boot loader and console

Ethernet connection used by the board to download the kernel and root file system. Sometimes, the root file system resides on the development host and is accessed by the target over the network.



TARGET BOARD
The boot loader starts on the board first and then loads the kernel, which mounts the root file system containing the application.

ANATOMY OF AN EMBEDDED LINUX SYSTEM

At run time, an embedded Linux system contains the following software components:

1. Boot loader: It gets the operating system loaded and running on the board.
2. Kernel: The software that manages hardware and and the processes.
3. Root file system: Under the /directory, contains the programs run by the kernel. Every Linux system has a root file system. Embedded systems have a great amount of flexibility in this respect: the root file system can reside in flash, can be bundled with the

kernel, or can reside on another computer on the network.

4. Application: The program that runs on the board. It can be a single file or a collection of hundreds of executables.

- NOTE:

Software: It is a set of user programs running on a system that are designed to be updated often by those users.

Firmware: It is semipermanent software running on a system that may or maynot be updated often and usually requires higher level of skills or effort.

Hardware: It is the physical component of a system that the software and firmware run on and are updated least often.

* Root File System:

A file system is a way of representing a hierarchial collection of directories, where each directory can contain either more directories of files.

In linux, the file system mounted at the top node is called the root fil system.

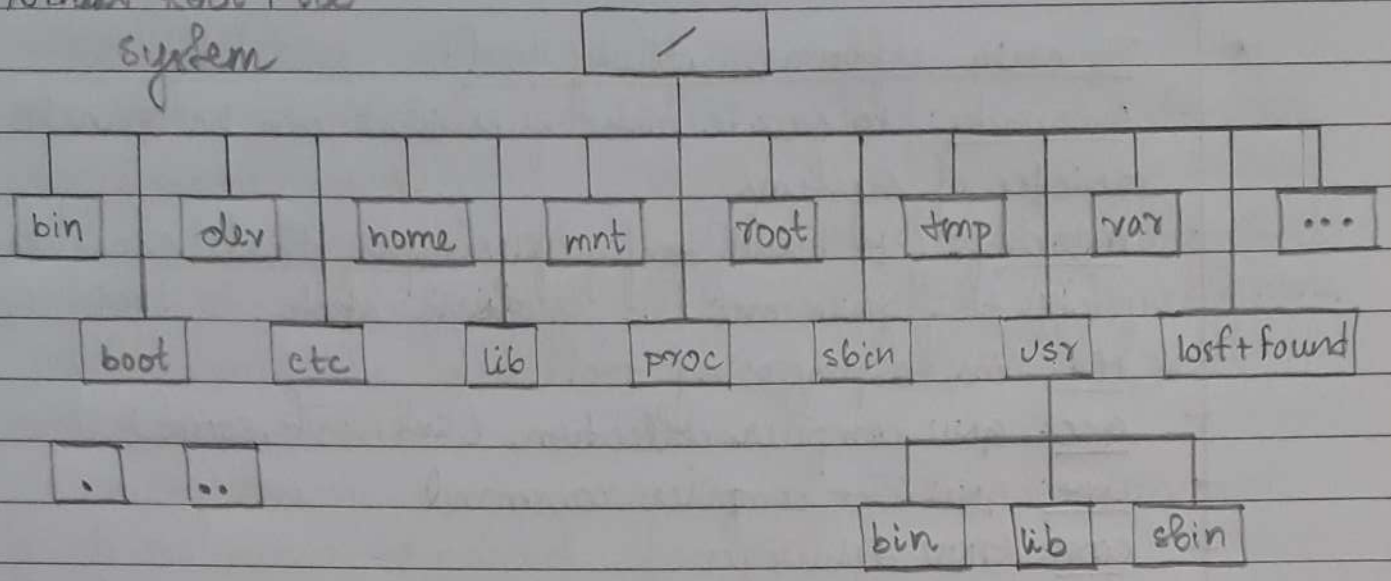
" The root file system is the file system that is contained on the same partition on which the root directory is located and it is the file system on which all the other file systems are mounted as the system is booted up."

- Assembling a Root File System:

1. Create the staging area.
2. Create a directory skeleton
3. Gather libraries and required files.
4. Create initilization scripts
5. Set permissions.

Keerthana Ashok

Linux Root File System



Directories	Contents
bin	user binaries
boot	Boot loader files
dev	device files
etc	configuration files including startup files
home	home directories
lib	system libraries (C libraries and kernel module)
mnt	mount directory
proc	process information
root	root user home directory
sbin	system binaries
tmp	temporary files
usr	user programs; it is secondary hierarchy containing most application and documents useful to most including the user and server documents.
var	variable files.
.. .	hidden files.
opt	optional add-on apps.
media	removable devices
srv	service data.

Page No.:
Date:

* The main software utility tool:
  - Automake: to create make files that can be run on a variety of systems.
  - Autoconf: to build configure scripts that scan the system to figure out the system's state.
  - M4: macro processing tool
  - GCC: GNU compiler collection (link, write, script)
  - G++: GNU C++ compiler command
  - GDB: GNU debugger.
  - dhcp3-server: Dynamic host configuration Protocol
  - nfs-user-server: Network file system
  - tftpd: Trivial File Transfer Protocol
  - minicom: send data communication used.

* Translation tools:

  QEMU (Quick Emulator) is a free and open-source emulator that performs hardware virtualization. It is a hosted virtual machine monitor: it emulates the machine's processor through dynamic binary translation and provides a set of different hardware and device models for the machine, enabling it to run a variety of guest operating systems.

• NOTE:

Busy Box: The software tool that provides several units of utilities in a single executable file.

Software Suits: It is a collection of computer program. Usually application software are programming software of related functionality often sharing a similar user interface and ability to easily exchange data with each other.

Keerthana Ashok

Unix utilities: It is list of commands specified by IEEE 1003.1 standards which is part of the single unix specification (SUS).

computer Program: It is collection of instructions that perform the specific task when executed by a computer. Instructions are executed through Instruction set Architecture (ISA).

Busy Box runs in a variety of POSIX environment such as Linux, Android, free BSD.

UNIT - 03

Embedded Hardware - I

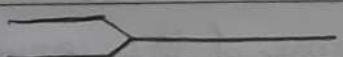
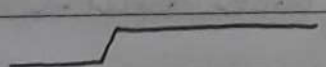
* Embedded Hardware Building Blocks:

Diagrams and symbols are the keys to quickly understand the most complex hardware design. They contain information that is needed to design any software that requires compatibility with the hardware.

There are several different types of engineering hardware drawings which includes:

1. Timing Diagrams:

Timing diagrams display timing graphs of various input and output signals of a circuit. It also displays the relationships between the various signals.

symbol	Input signals	Output signals
	Input signal must be valid.	Output signal will be valid
	Input signal doesn't affect system, it will work regardless	Indeterminate output signal
	Garbage signal	Output signal not driven, tristate, high impedance
	If the input signal rises	Output signal will rise
	If the input signal falls	Output signal will fall.

Date: YOUVA

2. Block Diagrams:

A block diagram is a basic overview of the hardware. It reflects the actual physical layout of a board containing major components such as processors, buses, I/O, memory.

It mainly shows how different components or units function together at a systems architecture level.

The symbols used within a block diagram are simple, such as squares or rectangles for chips and straight lines for buses. The block diagram are not detailed enough to write all of the low-level software accurately enough to control the hardware as it is just a basic overview of the hardware.

3. Schematics:

Schematics are electronic circuit diagrams that provide a more detailed view of all of the devices within a circuit or within a single component (processors to resistors).

It is not meant to depict the physical layout of the board but to provide information on the flow of data in the system such as how signals are assigned, how signal travels on various lines of a bus or appears on the pins of the processor and so on.

In case of schematic diagram, some of the conventions and rules include:

- A title section at the bottom of each schematic page with:
  - name of the circuit
  - name of the hardware engineer of the design
  - date and list of revisions, so on.
- Along with a symbol a label that details information about the component (size, type, power ratings). Labels for components of a symbol such as IC pin numbers, signal names
- Abbreviations and prefixes are used for common units of measurement (such as k-kilo and M-mega).

- Functional groups and subgroups of components are separated onto different pages.
- Positive voltage supply terminals are located at the top of the page and negative supply / ground terminals are at the bottom. Input components are usually on the left and output components are on the right.

4. Wiring Diagrams:

Wiring diagrams represent the bus connections between the major and minor components on a board or within a chip. The horizontal and vertical lines are used to represent the lines of a bus and either schematic symbols or more simplified symbols are used.

It represents an approximate depiction of the physical layout of a component or board.

5. Logic Diagrams/prints:

Logic Diagrams / prints are used to show a wide variety of circuit information using logical symbols (AND, OR, NOT etc) and logical inputs and outputs (1's and 0's).

These diagrams do not replace schematics but they can be useful in simplifying certain types of circuits in order to understand how they function.

- Traister and Lisk Method: (to learn to read or create a hardware diagram)

Step 1: Learn the basic symbols that can make up the type of diagram such as timing or schematic symbols.

Step 2: Read as many diagrams as possible until reading them becomes boring or comfortable.

Step 3: Write a diagram to practice simulating what has been read, again until it either becomes boring or comfortable.

Page No.:
Date:
YOUVA

* Embedded Processors:

Processors are the main functional units of an embedded board and are primarily responsible for processing instructions and data. An electronic device contains atleast one master processor acting as the central controlling device and can have additional slave processors that work with and are controlled by the master processor. These slave processor may either extend the instruction set of the master processor or act to manage memory, buses and I/O devices.

The complexity of the master processor usually determines whether it is called as microprocessor or a microcontroller. Usually microcontrollers contain a minimal set of integrated memory and I/O components whereas microcontrollers have most of the system memory and I/O components integrated on the chip. These days microprocessors are also increasingly becoming more integrated.

- Importance of integrated processor:

- some components like I/O may show a decrease in performance when integrated into a master processor, while others show an increase in performance as there will be no latencies due to transmitting data over buses between the processors.
- It simplifies the entire board design since there are fewer board components resulting in a board which is simpler to debug.
- The power requirements of components integrated into a chip is a lot less that those same components at the board level hence fewer components and lower power requirements results in smaller and cheaper board.
- But there is less flexibility in changing, adding or removing functionality since components are already integrated onto a processor.

* Architectures:

Despite the sheer number of available designs, the embedded processors can be separated into various groups called architectures. Processors are considered as of the same architecture when they can execute the same set of machine code instructions.

- ISA Architecture Models:

The features that are built into an architecture's instruction set are commonly referred to as the Instruction set Architecture (ISA). It defines such features as:

- Operations: used by programmers to create programs
- Operands (data): they are accepted and processed by architecture
- storage
- Addressing modes: used to gain access to and process operands and the handling of interrupts.

ISA implementation is a determining factor in defining important characteristics of an embedded design such as performance, design time, available functionality and cost.

- Operations:

Operations are made up of one or more instructions that execute certain commands. Operations are the functions that can be performed on the data, which includes the following types:

- Computations: math operations
- Movement: moving data
- Branches: conditional/unconditional moves
- Input/Output operations: Data transmission
- Context Switching Operations: temporary switching to routines and switch back to original instruction stream.

Ex: Math and logical - add, subtract, multiply, divide, AND, OR, XOR ...
Shift and rotate - Logical shift left, logical shift right, rotate right ...
Load/Store - stack PUSH, stack POP, load, store ...
Compare Instructions - Move and branch instructions. ---

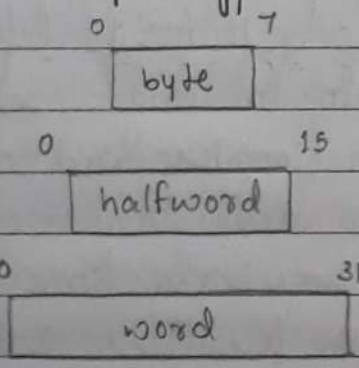
Operation Formats:

The format of an operation is the actual number and combination of bits (1's and 0's) that represent the operation and is refered to as operation code or opcode

- Operands:

Operands are the data that operations manipulate. An ISA defines the types and formats of operands for a particular architecture. Ex: MPE823: Motorola ; SA-1110: Intel strong ARM. An ISA also defines the operand formats that a particular architecture can support such as binary, decimal and hexadecimal.

An ISA defines the simple operand types of bytes (8 bits), halfwords (16 bits) and words (32 bits). Complex data types such as integers, floating point or character are based on the simple types.



(a) Simple operand types

→ MOV registerX, 10d
Move decimal value 10 into registerX
→ MOV registerX, $0ah
Move hexadecimal value A to registerX
→ MOV registerX, 00001010b
Move decimal value 00001010 to registerX

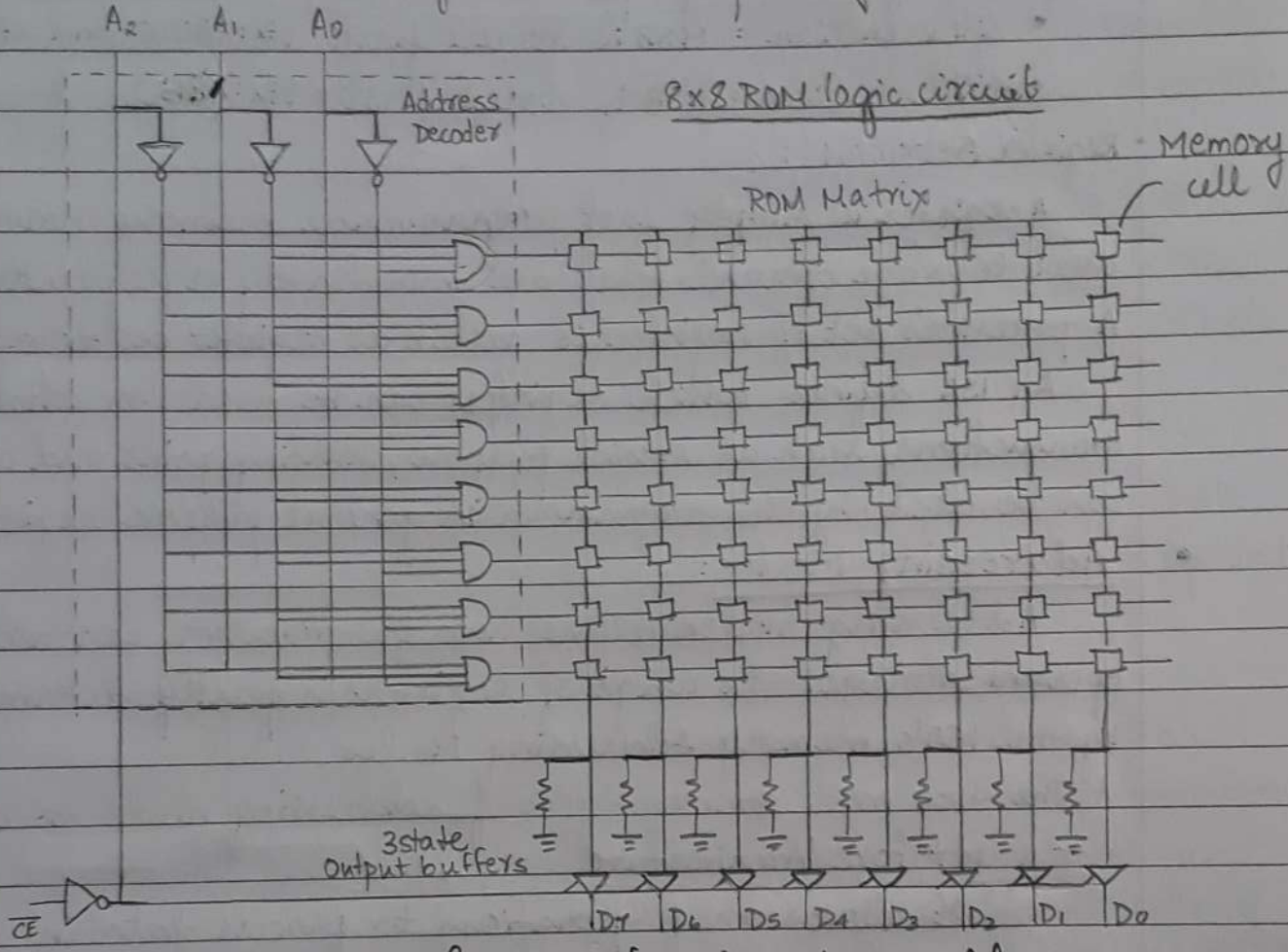
(b) Operand formats pseudocode example

- Storage:

The programmable storage is used to store the data that is being operated on. Memory is simply an array of programmable storage that stores data including operations, operands and so on. The indices of this array are locations referred to as memory addresses where each location is a unit of memory that can be addressed separately. The actual physical or virtual range of addresses available to a processor is referred to as the address space.

Page No.:
Date:
YOUVA

Block Diagram of memory array



An ISA defines specific characteristics of the address space as:

- Linear: A linear address space is in which specific memory locations are represented incrementally from 0 to 2^{N-1} where N is the address width in bits.
- Segmented: A segmented address space is a portion of memory that is divided into sections called segments. Specific memory location can only be accessed by specifying a segment identifier, a segment number that can be explicitly defined or implicitly obtained from a register and specifying the offset within a specific segment within the segmented address space. The offset within the segment contains a base address and a limit which map to another portion of memory that is set up as a linear address space. If the offset is less than or equal to the limit, the offset is added to the base address, giving the unsegmented address within the linear address space.

Page
Date: YOUVA

The two byte ordering approaches are:
- Little endian : MSB is stored first (Ex: 68000 and SPARC)
- Little endian : LSB is stored first (Ex: X86)

- Register Set:

A register is simply fast programmable memory normally used to store operands that are immediately or frequently used. A processor's set of registers is called as register set or register file.

An ISA defines which registers can be used for what transactions, such as special purpose, floating point and which can be used by the programmer as general purpose registers.

- Addressing Modes:

Addressing modes define how the processor can access operand storage. The usage of registers is partly determined by the ISA's memory Addressing Modes.

The two most common types of addressing mode modes are:

- Load-store Architecture

It allows only operations to process data in registers, not anywhere else in memory (PowerPC Architecture)

- Register-Memory Architecture

It allows operations to be processed both within registers and other types of memory (Intel's i960Jx processor)

- Interrupts and Exception Handling:

Interrupts (Exceptions/traps) are mechanisms that stop the standard flow of the program in order to execute another set of code in response to some event such as problems in the hardware, reset, etc.

* ISA Models:

There are several different ISA models that architectures are based upon, each with their own definations for the various features. The most commonly implemented ISA models are:

1. Application Specific ISA Models:

They define processors that are intended for specific embedded applications. There are several types of application specific ISA models:

- Controller Model:

The controller ISA is implemented in processors that are not required to perform complex data manipulation such as video and audio processors. They are used as slave processors on a TV board.

- Datapath Model:

The datapath ISA is implemented in processors whose purpose is to repeatedly perform fixed computations on different sets of data. Ex: digital signal processors (DSP's).

- Finite State Machine with Datapath (FSMD) Model:

The FSMD ISA is an implementation based upon a combination of the Datapath ISA and the controller ISA for processors that are not required to perform complex data manipulation and must repeatedly perform fixed computations on different sets of data. Ex: Application Specific Integrated circuits (ASIC's), Programmable logic devices (PLD's) and field-programmable gate arrays (FPGA's)

- Java Virtual Machine (JVM) model:

The JVM ISA is based upon one of the Java Virtual Machine standards Sun Microsystem's Java language.

2. General Purpose ISA Models:

They are typically implemented in processors targeted to be used in a wide variety of systems rather than only in specific types of systems. The most common types are:

- Complex Instruction Set computing (CISC) Model:

The CISC ISA defines complex operations made up of several instructions. Ex: Intel's X86

- Reduced Instruction Set computing (RISC) Mode:

The RISC ISA defines simpler and fewer operations made up of fewer instructions. Ex: ARM, PowerPC, SPARC

Page No.:
Date:

3. Instruction Level Parallelism ISA Models:

They are similar to general purpose ISA's except that they excute multiple instructions in parallel. They are considered higher evolutions of the RISC ISA which typically has one-cycle operations, one of the main reasons why RISCs are the basis for parallelism.

- Single Instruction Multiple Data (SIMD) Model

The SIMD machine ISA is designed to process an instruction simultaneously on multiple data components that require action to be performed on them.

- Super-Scalar Machine Model

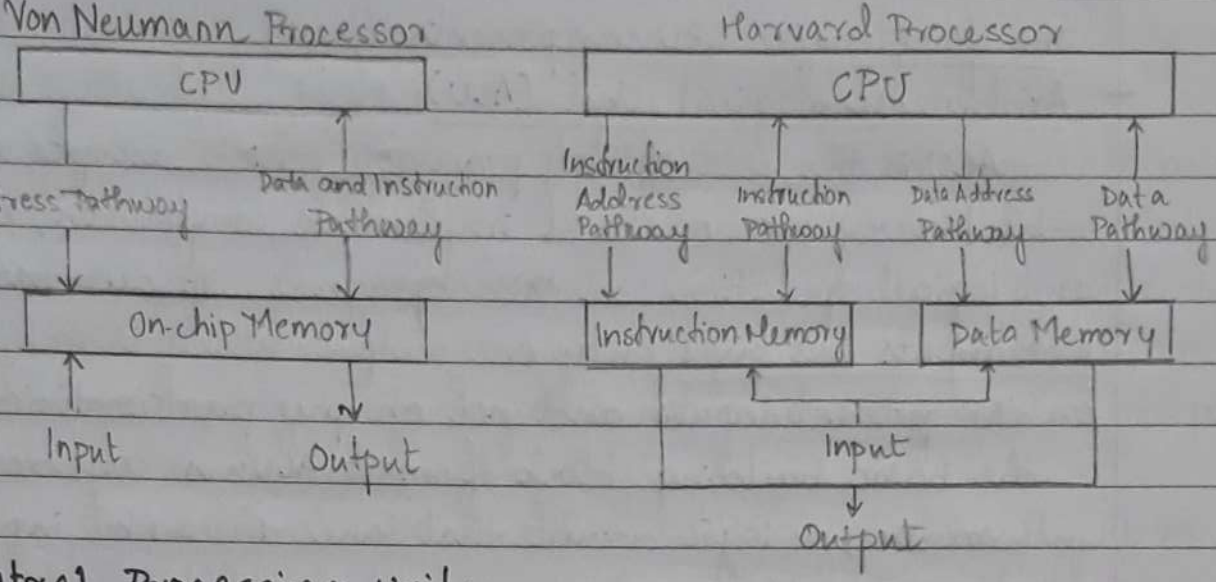
The superscalar ISA is able to process multiple instructions simultaneously within one clock cycle through the implementation of multiple functional components within the processor.

- Very Long Instruction Word Computing (VLIW) Model

The VLIW ISA defines an architecture in which a very long instruction word is made up of multiple operations. These operations are then broken down and processed in parallel by multiple execution units within the processor

INTERNAL PROCESSOR DESIGN:

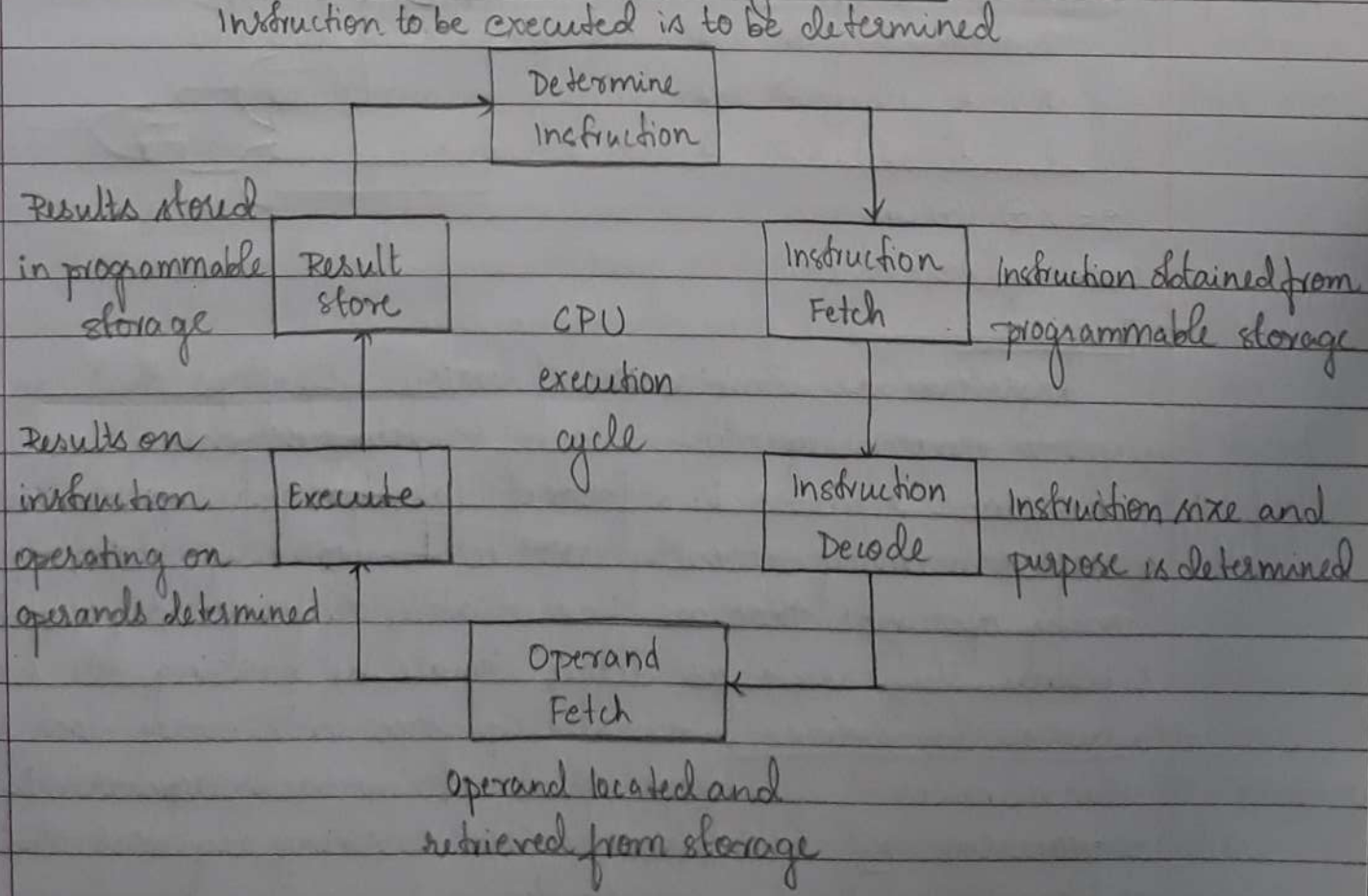
Processor's hardware design is based on Von Neumann or Harvard Architecture model. Von Neumann architecture defines a single memory space to store instructions and data. A Harvard architecture defines separate memory space for instruction and data. The main reason for the usage of these architectures is performance.



★ **Central Processing Unit:**

The actual processing unit within a processor is CPU. It is responsible for executing the cycle of fetching, decoding and executing instructions. This three-step process is called as three-stage pipeline. This cycle is implemented through some combination of four major CPU components.

Fetch, Decode and Execution Cycle of CPU



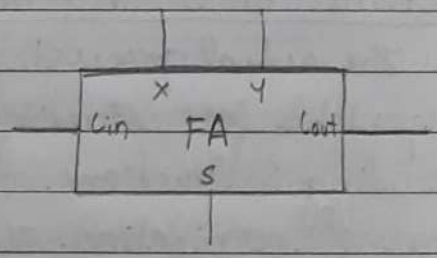
The four major CPU components are:

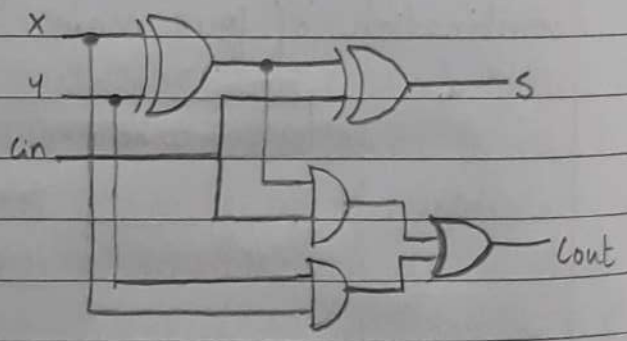
- Arithmetic Logical Unit (ALU)

ALU is the core of any processor which accepts multiple n-bit binary operands and implements comparison, mathematical and logical operations on these operands. It can have one or more inputs but gives only one output which is only dependent on the present inputs and not on any past conditions.

The basic building block of most ALU's is considered the full adder, a logic circuit that takes three 1 bit inputs and produces two 1 bit number.

X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1





$S = X \oplus Y \oplus Cin$

$Cout = Cin(X \oplus Y) + XY$

- Registers:

Registers are combination of various flip-flops that can be used to temporarily store data or to delay signals.

A storage register is a form of fast programmable internal processor memory usually used to temporarily store, copy and modify operands that are immediately or frequently used by the system. Shift registers delay signals by passing the signal between the various internal flip-flops with every clock pulse. The number of flip-flops that are in each register describe a processor. Ex: 32 bit processor has working registers that are 32 bit wide containing 32 flip flops.

The number of flip flops in these registers also determines the width of the data buses used in the system.

General Purpose Registers: Used to store and manipulate any type of data.

Special Purpose Registers: used to hold results for specific types of computations, have predetermined flags, acts as counters, for controlling I/O ports. shift registers are inherently special purpose because of their limited functionality.

Flags: Used to indicate to other circuitry that an event or a state change has occurred

counters: programmed to increment or decrement either asynchronously or synchronously such as with a procesor's program counter (PC) or timers that count clock cycles. By adding an additional flip-flop for every extra digit the size of the counter can be increased.

An asynchronous counter is a register whose flip-flops are not driven by the same central clock signal. whereas a synchronous counter is a register whose flip-flops are driven by the same central clock signal.

- Control Unit (CU):

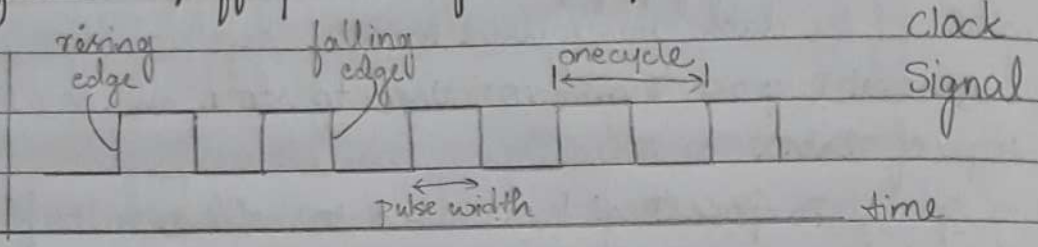
The control Unit is primarily responsible for generating timing signals as well as controlling and coordinating the fetching, decoding and execution of instructions in the CPU.

After the instruction has been fetched from memory and decoded the control unit then determines what operation will be performed by the ALU and selects and writes signals appropriate to each functional unit within or outside of the CPU.

The CPU and the System (Master) clock:

A processor's execution is ultimately synchronized by an external system or master clock, located on the board. The master clock is an oscillator along with a few other components, such as a crystal. It produces a fixed frequency sequence

of regular on/off pulse signals (square waves)



- Internal CPU buses:

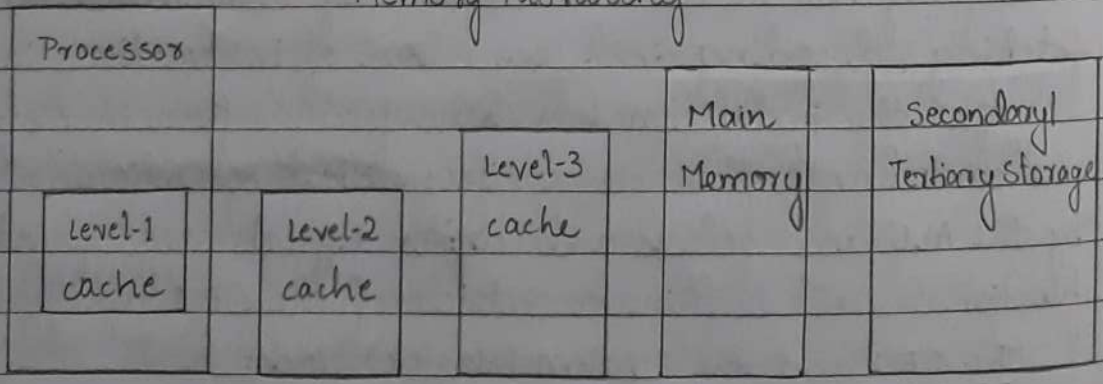
The CPU buses are the mechanisms that interconnect the CPU's other components: ALU, CU and registers. Each bus's wire is typically divided into logical functions such as:

- data - carries data bidirectionly between registers and the ALU.
- address - carries locations of the registers that contain the data to be transferred.
- control - carries control signals such as timing and control signals between the registers, ALU and CU.

★ On-Chip Memory:

Memory Hierarchy is a collection of different types of memory each with unique speeds, sizes and usages. Some of these memory can be physically integrated on the processor such as registers, read only memory (ROM), certain types of random access memory (RAM) and level-1 cache

Memory Hierarchy

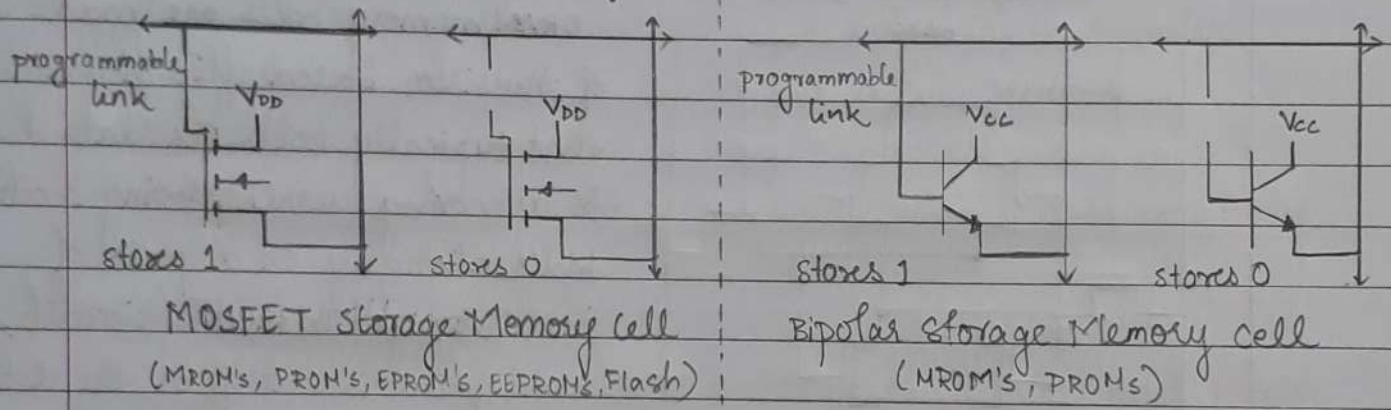


- ROM (Read Only Memory):

On chip ROM is memory integrated into a processor that contains data or instructions that remain even when there is no power in the system, due to a small longer-life battery and

therefore is considered to be a non volatile memory (NVM). The content of on chip ROM usually can only be read by the system it is used in. (Ex: 8×8 ROM logic circuit) Pg 1

Ex: 8×8 ROM means it can store eight different 8-bit bytes i.e., 64 bits of information. Every intersection of a row and column is a memory cell, which contains either a bipolar or MOSFET transistor and a fusible link.



When a programmable link is in place, the transistor is biased ON resulting in a '1' being stored. When writing to ROM, a '0' is stored by breaking the programmable link.

Types of on-chip ROMs:

- MROM (mask ROM): It is permanently etched into the microchip during the manufacturing of the processor, hence cannot be modified later.
- PROM (programmable ROM) or OTP (one-time programmable): It can be programmed once by a PROM programmer.
- EPROM (erasable programmable ROM): It can be erased and reprogrammed more than once using special devices.
- EEPROM (electrically erasable programmable ROM): It can be erased and re-programmed more than once without any special devices. In EEPROM erasing is done entirely unlike EPROM which can be erased selectively.
- Flash: A cheaper and faster variation of EEPROM. Flash can be written and erased in blocks or sectors whereas EEPROM are written and erased at the byte level.

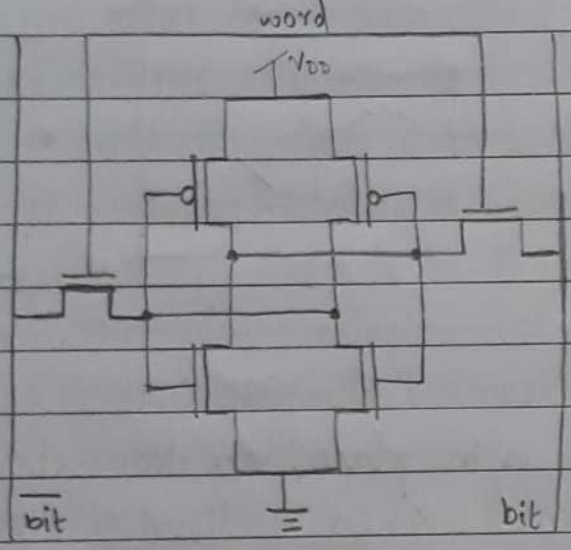
Page:
Date: YOUVA

- RAM (Random Access Memory):

It is commonly referred to as main memory. It is memory in which any location within it can be accessed directly and whose content can be changed more than once.

The contents of RAM are easily erased if RAM loses power, thus RAM is volatile. The two main types of RAM are:

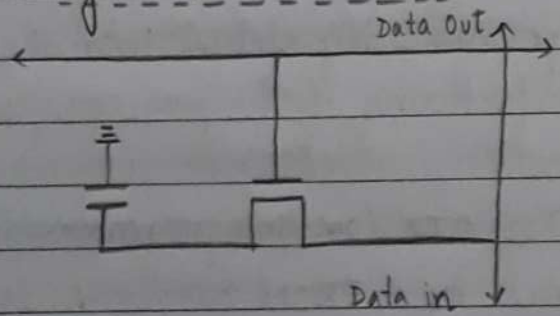
- Static RAM (SRAM)



6 transistor SRAM cell

SRAM memory cells are made up of transistor based flip-flop circuitary that typically holds its data due to a moving current being switched bidirectionally on a pair of inverting gates in the circuit, until power is cut off or the data is overwritten.

- Dynamic RAM (DRAM)



Capacitor based DRAM memory cell

DRAM memory cells are circuits with capacitors that hold a charge in place. The DRAM capacitors need to be refreshed frequently with power in order to maintain their respective charges and to recharge capacitors after DRAM is read (reading DRAM discharges the capacitor). The cycle of discharging and recharging of memory cells is why this type of RAM is called dynamic.

Page No.:
Date:
YOUVA

Differences between SRAM and DRAM

Static RAM	Dynamic RAM
• SRAM uses transistor to store a single bit of data.	• DRAM uses a separate capacitor to store each bit of data.
• It does not need periodic refreshment to maintain data	• It needs periodic refreshment to maintain the charge in the capacitors for data.
• structure is complex	• structure is simpler.
• Expensive	• less expensive
• Faster	• slower
• Used in cache memory	• Used in main memory.

- Cache (Level-1 Cache):

Cache is a level of memory between the CPU and main memory in the memory hierachy. It can be integrated into a processor or can be off-chip.

Cache is used to store subsets of main memory that are usually accessed often. Some processors have one cache for both instructions and data while other processors have separate on-chip caches for each (Von Neumann and Harvard Architecture)

→ Different strategies are used when writing to and reading data from level-1 cache and main memory:

- Transferring data between memory and cache in either one-word or multiword blocks. These blocks are made up of data from mainmemory along with the location of that data in the main memory (called tags).
- For writing to memory, given some memory address from the CPU which is translated to determine its equivalent location in level-1 cache. writing must be done in both cache and main memory inorder to ensure that the cache and main memory are consistent (ie same value). There are two common write strategies:

a. write-through: data is written to both cache and main-memory every time.

b. write-back: data is initially written only into cache, only when it is to be bumped and replaced from cache it is written into main memory.

- When the CPU wants to read data from memory, level-1 cache is checked first. If the data is in cache it is called cache hit. Then the data is returned to the CPU and the memory access procss is complete. If the data is not present in level-1 cache it is called cache miss. Off-chip caches are then checked if present. If it is a miss then the main memory is accessed to retrieve and return the data to the CPU.

→ Data is usually stored in cache in either of the following three schemes:

- Direct Mapped: data in cache is located by its associated block address in memory (using tags)
- Set Associative: cache is divided into sets, in which multiple blocks can be placed. Blocks are located according to an index field that maps into a cache's particular set.
- Full Associative: blocks are placed anywhere in cache and must be located by searching the entire cache everytime.

→ The most common cache selection and replacement schemes include:

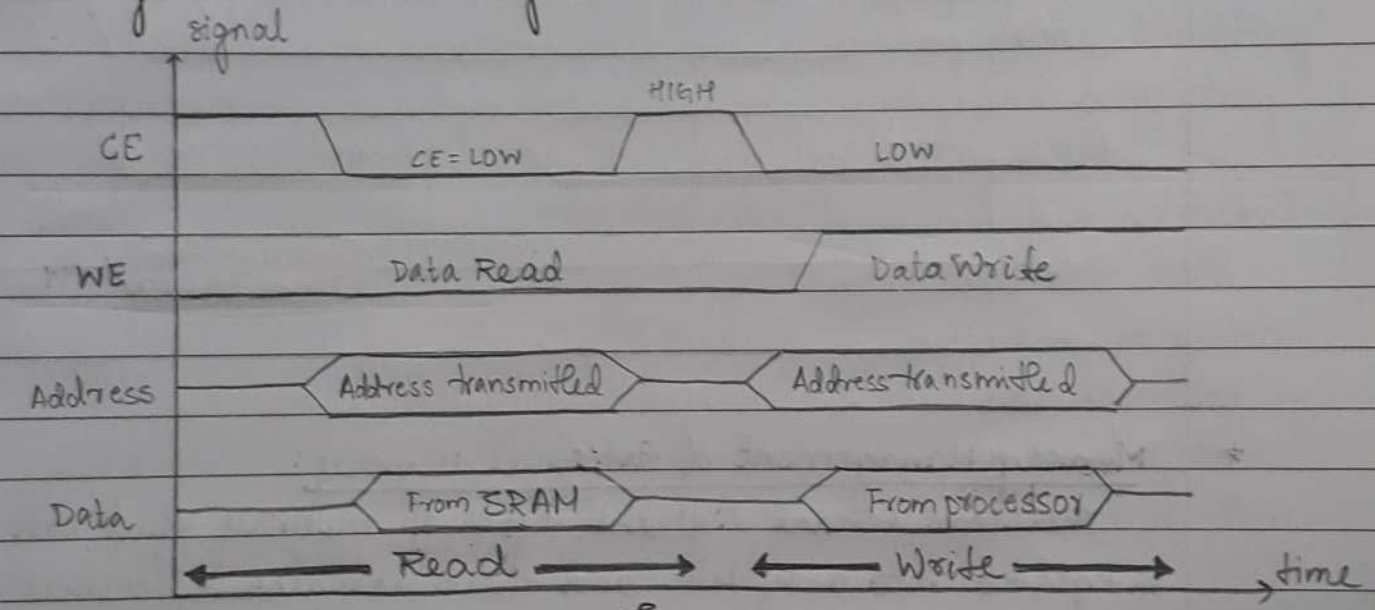
- Optimal: using future reference time, swapping out pages that won't be used in the near future.
- Least Recently Used (LRU): swaps out pages that were used the least recently.
- FIFO (First in First out): swaps out the pages that are the oldest regardless of how often they are accessed

by the system. It is less efficient.

- Not Recently Used (NRU): swaps out pages that were not used within a certain time period.
- Clock Paging: pages replaced according to clock (how long they have been in memory) in clock order, if they haven't been accessed.

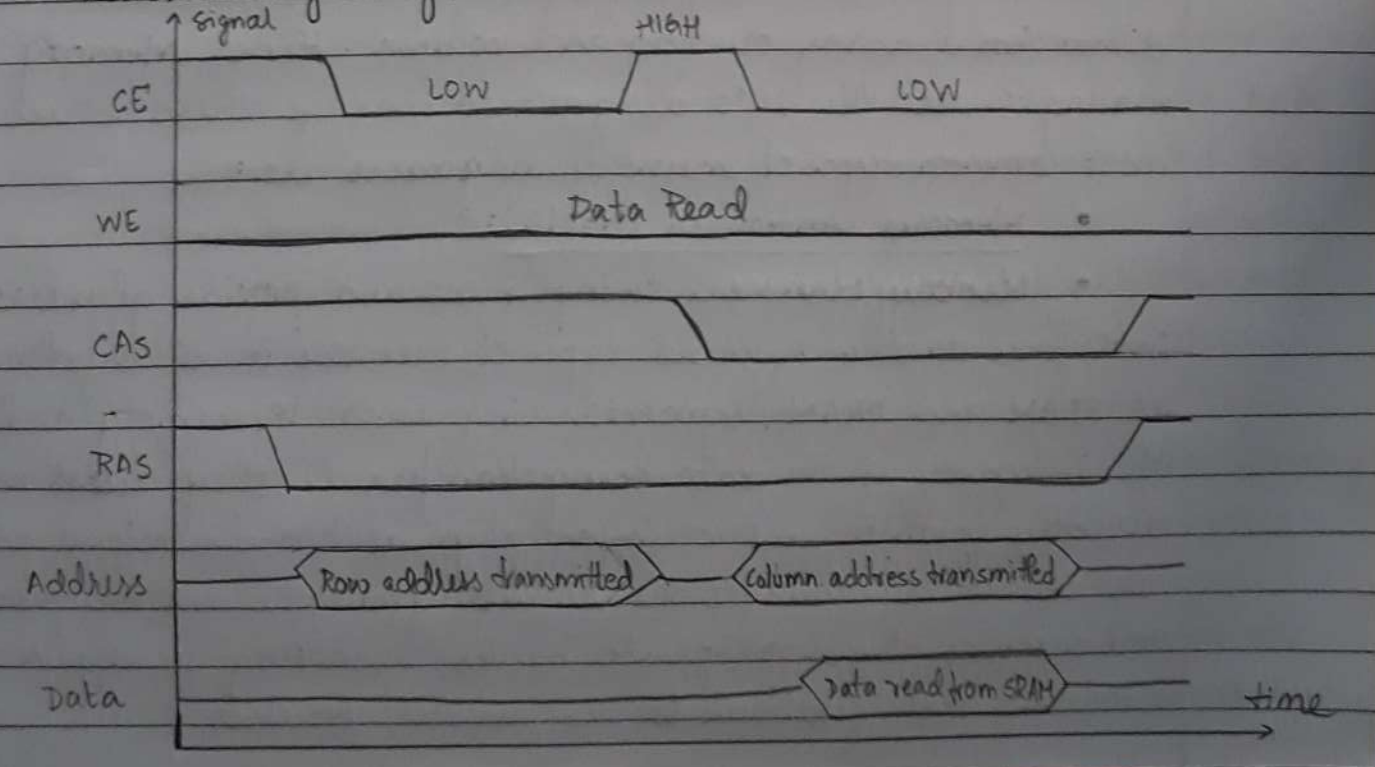
– Timing Diagrams:

1. Memory read and memory write in SRAM.

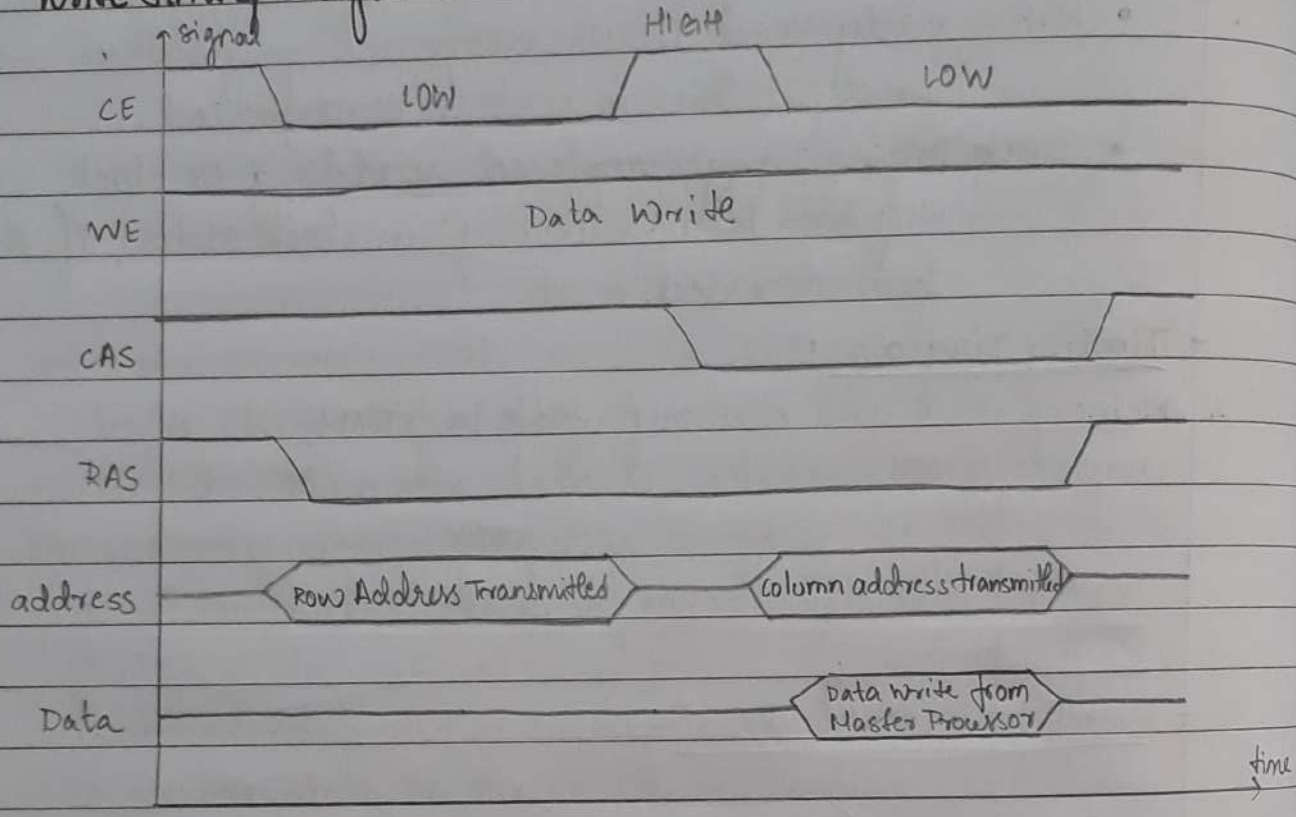


2. Memory read and memory write in DRAM

Read timing diagram



Write timing diagram



* Memory Management of External Memory:

There are several different types of memory that can be integrated into a system and there are also differences in how software running on the CPU views logical / virtual memory addresses and the actual physical memory addresses - the two dimensional array of row and column. Hence Memory managers are the IC's designed to manage these issues. The most common types of memory managers are:

- Memory Controllers (MEMC):

They are used to implement and provide glueless interfaces to the different types of memory in the system, such as SRAM and DRAM, synchronising access to memory and verifying the integrity of the data being transferred. They access memory directly with the memory's one physical two-dimensional addresses. The controller manages the request from the master processor and accesses the appropriate banks, awaiting feedback and returning that feedback to the master processor.

In some cases, the memory controller is mainly managing one type of memory, then it may be referred by that memory's name such as DRAM controller, cache controller etc.

- Memory Management Units (MMUs)

They mainly allow for the flexibility in a system of having a larger virtual memory space within an actual smaller physical memory. It can exist outside the master processor and is used to translate virtual addresses into physical addresses as well as handle memory security, controlling cache, handling bus arbitration between the CPU and memory and generating appropriate exceptions.

MMUs support the various schemes in translating addresses mainly segmentation, paging or some combination of both. In general segmentation is the division of virtual memory into larger variable size sections whereas paging is the dividing up of virtual memory into smaller fixed size units. In case of both the virtual memory is first divided into segments and then divided into pages.

* Board Memory and Performance:

The most common measures of a processor's performance is its throughput (bandwidth) or the CPU's average execution rate. The performance throughput can be negatively impacted by main memory especially since the DRAM used for main memory can have a much lower bandwidth than that of the processors. The solutions for improving the bandwidth of the main memory include:

- Harvard Architecture: separate instruction and data memory buffers and ports leads to high number of memory accesses and computations on a large amount of data.
- Using DRAMs such as DRDRAM and SLDRAM integrate the bus signals into one line, to decrease the time it takes to arbitrate

the memory bus to access memory.

- Using more memory interface connections (pins) increases transfer bandwidth.
- Using higher signalling rate on memory interface connections
- Implementing a memory hierarchy with multiple levels of cache which has faster memory access time than those of other types of memory.

There are specific timing parameters associated with memory (memory access time, refresh cycle time for DRAM etc) that act as indicators of memory performance. Memory performance can be improved by.

- Introducing cache means fewer slower DRAM accesses with a decrease in the average main memory access time, non blocking cache will especially decrease any cache miss penalties.
- Implementing pipeling, which is the process of breaking down the various functions associated with accessing memory into steps and overlapping some of these steps. which helps to increase throughput, by decreasing the time it takes for cache writes.
- Increasing the number of smaller multi-level caches rather than one big-cache, since smaller caches reduce the cache's miss penalty and average access time (hit time) whereas a larger cache has a longer cycletime.
- Integrating main memory onto the master processor which is cheaper as well as since on-chip bandwidth is usually cheaper than pin bandwidth.

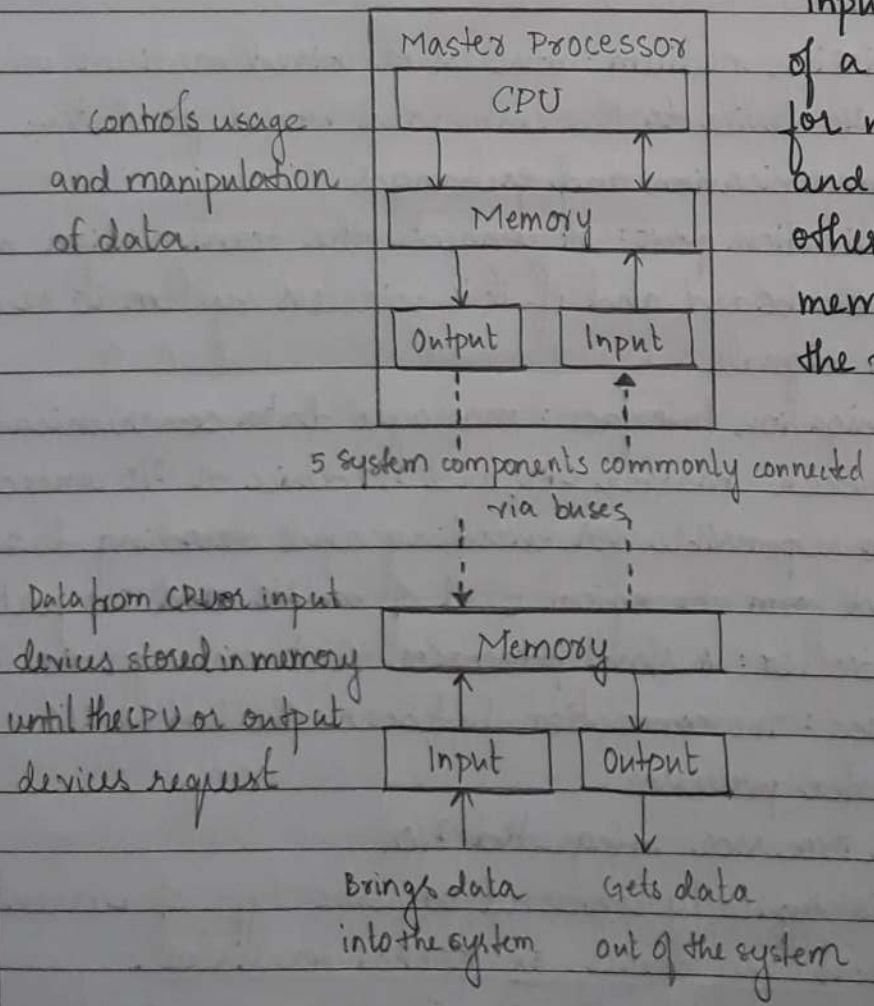
Page No.:
Date:
YOUVA

* Memory Organisation:

Memory organisation includes not only the makeup of the memory hierarchy of the particular platform but also the internal organization of memory, specifically what different portions of memory may or may not be used for as well as how all the different types of memory are organized and accessed by the rest of the system.

The master processor along with the software treats memory as one large one-dimensional array called a memory map. This map serves to clearly define what address or set of addresses are occupied by what components.

* Processor Input/Output (I/O):



Input/output components of a processor are responsible for moving information to and from the processor's other components to any memory and I/O outside of the processor on the board.

The processor I/O components can be organised into categories based on the functions they support.

- networking and communication I/O (physical layer of OSI model)
- input (keyboard, mouse, voice etc)
- graphics and output I/O (touch screens, CRT, printers, LEDs, etc.)
- storage I/O (optical disk controllers, magnetic disk controllers)
- debugging I/O (BDM, JTAG, serial port, parallel port etc)
- real-time and miscellaneous I/O (timers/counters, ADC, DAC, watchdog)

An I/O subsystem can be as simple as a basic electronic circuit that connects the master processor directly to an I/O device to more complex I/O subsystem circuitry that includes several units.

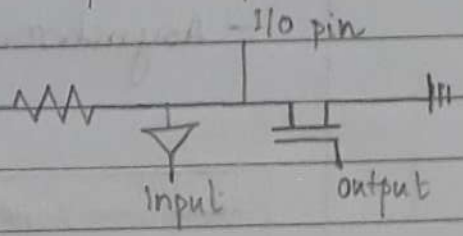
I/O hardware are made up of combination of six main logical units:

- Transmission medium: wireless or wired medium connecting the I/O device to the embedded board for data communication and exchanges.
- Communication port: It connects the transmission medium to the board and if it is wireless system it receives the signals.
- Communication Interface: manages data communication between master CPU and I/O device or I/O controller. Also responsible for encoding and decoding data to and from the logical level of an I/C and I/O port.
- I/O controller: A slave processor that manages the I/O device
- I/O Buses: The connection between the board I/O and master processor.
- Master Processor Integrated I/O.

An I/O pin is typically connected to some type of current source and switching device. Ex: MOSFET transistor.

In the following example the circuit allows the pin to be used for both input and output.

I/O port sample circuit



When the transistor is turned OFF (open switch), the pin acts as an input pin and when the switch is ON (closed switch), the pin acts as an output port.

- Managing I/O data

Serial I/O

Processor I/O that can transmit and receive data serially is made up of components in which data is stored, transferred and received one bit at a time.

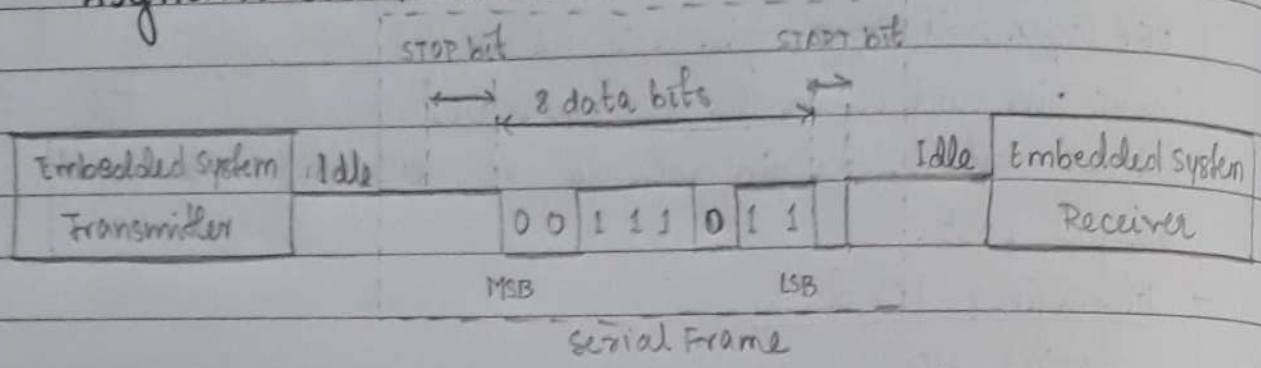
Serial communication includes: a serial port and a serial interface. Serial interfaces manage the serial data transmission and reception between the master CPU and either the I/O device or its controller. They include reception and transmission buffers to store and encode data.

Data can be transmitted between two devices in the following ways:

- A simplex scheme for serial I/O data communication can only transmit and receive data stream in one direction.
- A half duplex scheme can transmit and receive a data stream in either direction but in only one direction at any one time.
- A full duplex scheme can transmit and receive a data stream in either direction simultaneously.

Within the actual data stream, serial I/O transfers can occur either as a steady / continuous stream at regular intervals regulated by the CPU's clock referred to as a synchronous transfer or intermittently at irregular/random intervals referred to as an asynchronous transfer.

- Asynchronous Transmission



In an asynchronous transfer the data being transmitted can be stored and modified within a serial interface transmission buffer or registers. The serial interface at the transmitter divides the data stream into packets. Each of these packets is then encapsulated in frames that is transmitted separately.

The frames are packets that are modified before transmission by the serial interface to include a START bit at the start of the stream and a STOP bit or bits at the end of the data stream. Within the frame after the data bits and before the STOP bit a parity bit may also be appended.

- A START bit indicates the start of a frame.
- A STOP bit indicates the end of a frame.
- A parity bit is optional which is used for very basic error checking.

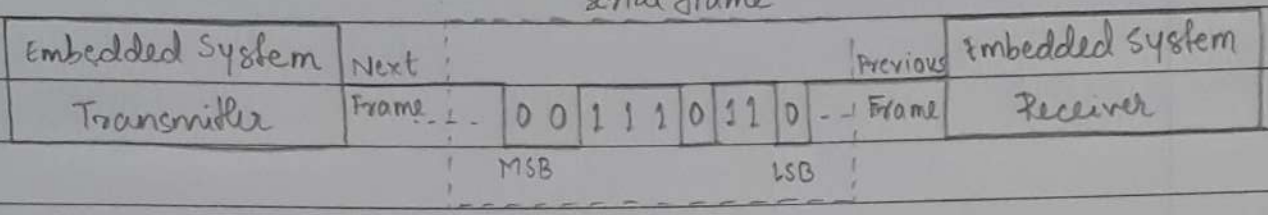
Between the transmission of frames, the communication channel is kept in an idle state i.e., logic level 1 or non return to zero (NRZ) state is maintained.

The serial interface receiver then receives frames by synchronizing to START bit of a frame, delays for a brief period and then shifts one bit at a time into its receiver buffer until reaching the STOP bit.

Bit rate : $\frac{\text{number of actual data bits per frame}}{\text{total number of bits per frame}} \times \text{baud rate}$

Baud rate: Total number of bits that can be transmitted.

- Synchronous Transmission



In a synchronous transmission there are no START or STOP bits appended to the data stream or there is no idle period. The devices involved in a synchronous transmission are synchronized by one common clock which does not start or stop with each new frame. In some synchronous serial interfaces if there is no separate clock line then the clock signal may even be transmitted along with the data bits.

Examples:

UART: Universal Asynchronous Receiver - Transmitter is an example of a serial interface that does asynchronous serial transmission.

SPI: Serial Peripheral Interface is an example of a synchronous serial interface.

- Differences between serial I/O and Parallel I/O:

		SERIAL TRANSMISSION	PARALLEL TRANSMISSION
1.	Transmission	Data flows in bidirection, bit by bit.	Multiple lines are used to send data (8 bit at a time).
2.	Cost	Economical	Expensive
3.	Bits/clock pulse	1 bit	8 bits
4.	Speed	Slow	Fast
5.	Number of communication channel required	Only one	N number of channels are required.
6.	Applications	Long distance communication	Short distance Communication
7.	Need of converters	Required to convert the signal according to the need.	Not required