EMBEDDED AND REAL TIME SYSTEMS

UNIT 1 ⇒ Complex system and Microprocessor

* Embedded system is one that has computer hadward with software embedded in it as one of the important components.

* An embedded system is a specific computer system design to perform one or a few dedicated functions (tasks) often with real-time computing constrains.

* It is a sbopisticated system that has a computer hardware with Application software and Real-time Operating system (RTOS) embedded in it as one of its components.

→ It executes (runs) a predefined and dedicated task for an application.

→ The embedded system may be an independent system or a part of a larger system.

Embedded systems are electronic systems that consists of microprocessor or microcontroller, memory, Input/output devices but It is not a computer.

An embedded system has three main components

1. Hardware ; Embedded system has hardware similar to a computer. As its software locates in the ROM or Flash memory, it does not need a secondary hard disk and CD/DVD memory as in a computer.

2. Application software : The application software may concurrently performs a series of tasks or processes or threads.

3. RTOS [Real Time Operating Systems]
   * It supervises the application software running on the hardware.
   * It organizes access to a resource according to the priorities of tasks in the systems.
   * It provides a mechanism to the processor run a process/task as scheduled & context-switch between the various tasks.
   * It sets the rules during the execution of the application software.
   * A small scale embedded system may not need an RTOS.

* KERNEL : The kernal is the central part (component) of most computer OS. It is responsible for task management, Inter task-communication and synchronization. It stays in RAM. & it runs until the system turned off or crashes. User Program make use of kernal via the sys-call interface

Application Specific System Processors (ASSP)

⇒ An ASSP is dedicated to specific task and provides a faster solution

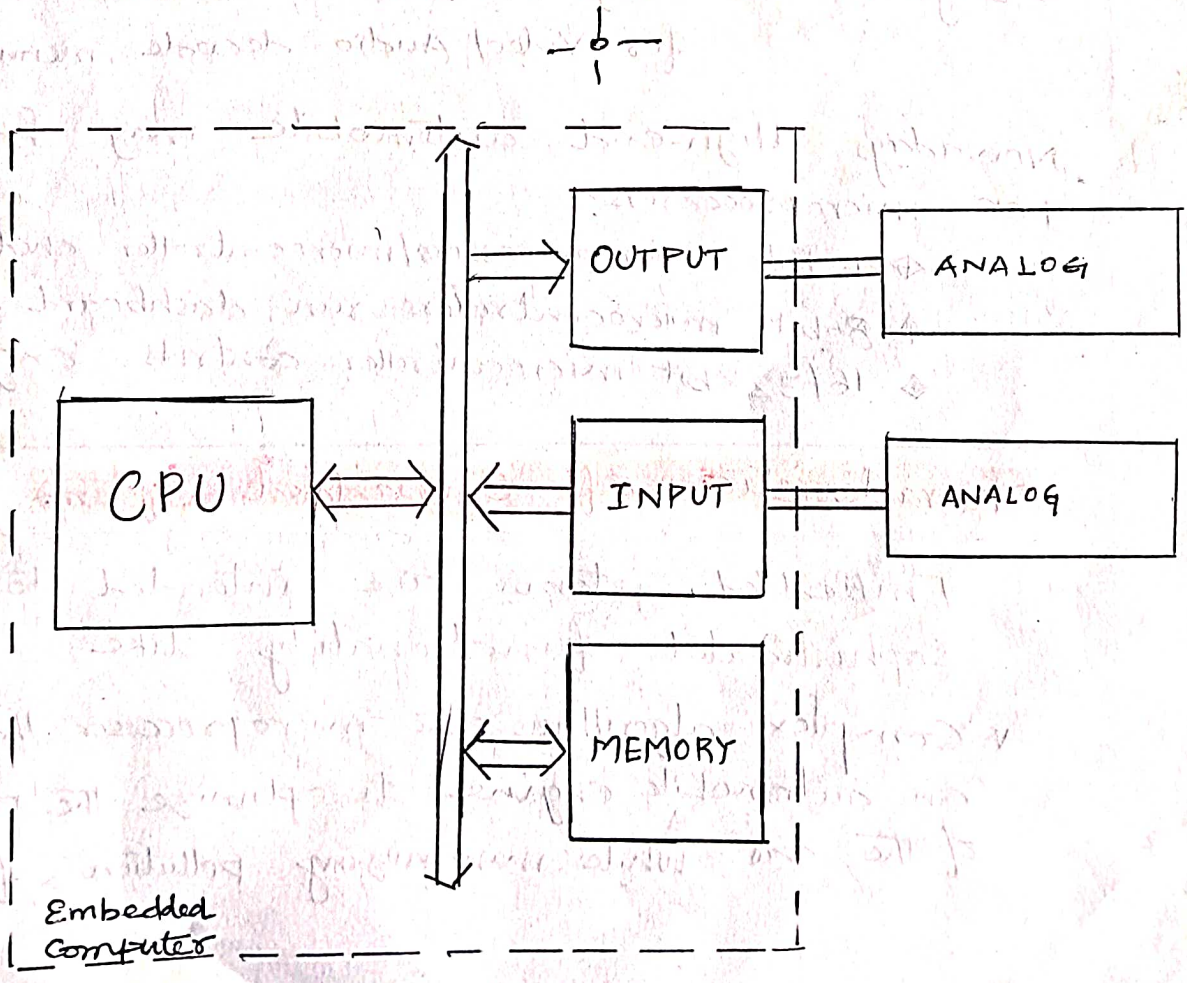
⇒ An ASSP is used as an additional processing unit for running the applications in place of using embedded software.

Ex: IIM 7100

Multi-processor System using General Purpose Processors

⇒ Multiple processors are used when a single processor does not meet the needs of the different tasks that have to be performed concurrently

⇒ The operations of all the processors are synchronized to obtain an optimum performance



Examples of embedded System

- Cell phone
- Printer
- Automobile : engine, brakes, dash etc.
- Airplane : engine, flight controls, nav/comm.
- Digital TV
- Household appliances.

Early history

→ Late 1940's : MIT Whirlwind computer was designed for Real-time operations.

→ HP-35 calculator used several chips to implement a microprocessor in 1972.

→ Automobiles used microprocess-based engine controllers starting in 1970's

→ Canon EOS3 has three microprocessors.
- 32-bit RISC CPU runs autofocus & eye control system.

→ Digital TV : Programmable CPUs, Hardwired logic for Video/Audio decode, menus etc.,

Nowadays High-end automobile may have 100 microprocessors:
- 4-bit microprocessor/microcontroller checks seat belt
- 8-bit microcontrollers run dashboard devices
- 16/32-bit microcontroller controls engine.

Characteristics of Embedded Systems.

Embedded systems are intended to provide sophisticated functionality like

* Complex algorithms : the microprocessor that controls an automobile engine to optimize the performance of the car while minimizing pollution & fuel utilization.

User interfaces : Microprocessors are frequently used to control complex user interfaces like multiple menus & many operations

Ex : Moving map in GPS navigation.

So, to meet these kinds of embedded computing operations needs some important characteristics. like

→ Real time operation
→ Multirate
→ Cost
→ Power & energy

→ Sophisticated functionality
→ Designed to tight deadlines by small teams.

Real Time :

The tasks must be completed by deadlines.
There two real time operations

1. Hard real-time : missing operation causes unsafe and can even endanger lives.
2. Soft real-time : missing deadline results in degraded performance

Multirate :

- The embedded computing system have several real-time activities performing on at the same time
- The tasks may have different rates.
- Ex. Multimedia applications, The audio & video have different rates but they must be synchronized. Otherwise it spoils the perception of the entire presentation.

Manufacturing cost: It is important in many cases. The cost is determined by many factors including the types of microprocessor used, size of memory required and the types of I/O devices.

Power and Energy:

Power consumption - affects the cost of the hardware since a larger power supply is required.

Energy consumption - affects the battery life

Sophisticated functionality:

→ often have to run complex or multiple algorithms - Ex: cell phone, laser printer.

→ often provide sophisticated user interface.

Design team:

→ often designed by a small team of designer.

Reason to select microprocessor for an Embedded System

→ Microprocessors are a very efficient way to implement digital systems.

→ Easier to design families of product with various features at different prices.

→ It can be extended to provide new features to keep up with rapidly changing markets.

→ It executes programs very efficiently.

→ Microprocessors are very efficient utilizers of logic.

→ Different algorithms can be used by changing the programs.

→ Several function can be implemented in a single processor.

Challenges in Embedded Computing System Design

Some important problems that makes difficult in embedded system design.

→ **Hardware**

An embedded system consist of microprocessor, amount of memory, I/O ports and application demanded peripherals.

To meet performance deadline & manufacturing cost the selection of hardware is important

too little hardware – The system fails to meet its deadlines.

too much hardware – The system becomes too expensive & consumes more power.

→ **Deadline**

- To meet the deadline, have to increase the speed of the hardware but the system will be more expensive.
- On the other way can increase the clock rate of the CPU, Program speed may be limited by the system memory.

Power consumption

→ Reduce the power consumption is a very important task in an embedded system.

→ Excessive power consumption can increase heat dissipation.

→ To reduce the power consumption, slowdown the system speed. But it lead to missed deadlines.

→ Therefore careful design is required to slow down the non-critical parts of the system for power consumption. with meeting the deadline.

Design for Upgradability

→ Same hardware may be used for the upgrade version of a product in the same generation

→ Adding or changing of features may be performed by changing the software.

→ But we could design a system that will provide the required performance for software without changing it.

The Reliability of the system is another important factor, particularly in safty-critical system

The nature of embedded computing system makes their design more difficult. They are

1) Complex testing
2) Limited observability & controllability
3) Restricted development environments

Complex testing :

Testing of an embedded system is very difficult because the embedded system is an integration of software and hardware machine.

limited Observability and Controllability :

In Real-time embedded systems, It is not easy to observe the tasks runs in the embedded system and also stop it.

Restricted Development Environment :

→ The development environment (IDE) for embedded systems are ~~ver~~ very less & limited.

→ The code is compiled on one type of PC & download it onto the embedded system.

→ To debug the code, we must rely on program that runs on the PC and then look inside the embedded system.

Embedded System Design Process :

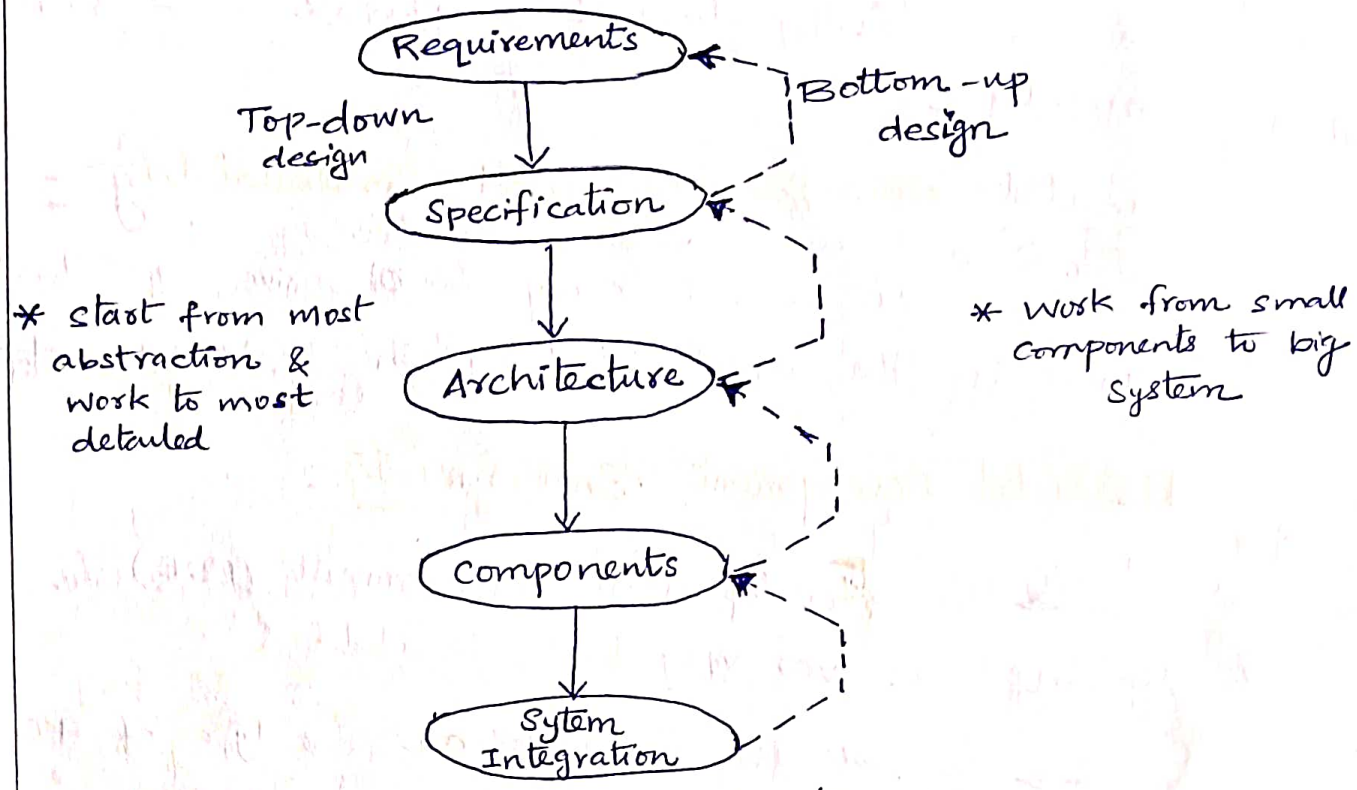
It has two objectives

1) Introduction to the various steps in embedded system design before developed in detailed design.
2) Design Methodology

The design methodology is important for three reasons

1) It allow us to test the performance for optimizing.
2) It allow us to develop computer-aided design tools.
3) Makes the design much easier for members of a design team to communicate.

They are 5 major steps in the embedded system design process. The design can approach either with Top-Down or Bottom-up design.



Major levels of abstraction in the Design process

➪ The bottom-up design is to help us refine the system.

➪ The major goals of the design are
- manufacturing cost
- Performance (speed & deadlines)
- Power consumption
- cost

➪ **Requirements Analysis**

The designing process is generally proceed in two phases
1) we gather an informal description from the customer known as requirements.
2) we refine the requirements into a specification that contain enough information to begin designing the system architecture.

→Requirement may be functional or nonfunctional descriptions.

→Both descriptions should be include for a design.

Functional requirements ; Output as function of input

Non-functional requirements ; Time required to compute O/P.,
Size, weight, etc.,
Power consumption,
realiability, etc.
cost,
Performance etc.

Performance

→ System speed & cost are major considerations.

→ the performance may be a combination of soft performance such as approximate time to operate a user-level function & Hard deadlines (completion of the task with in the time)

Cost:

The purchase price of the system includes manufacturing cost (components & assembly) & nonrecurring engineering cost (Personal & other expences)

Physical Size & Weight

Industrial / Instrument: Size – greatly depending on the application & Weight – ~~not~~ no limitation ~~on~~ Weight

Handheld device: size & weight should be minimum

Power Consumption:

Power can be specified in the requirement stage in terms of battery life.

A Sample requirement form of a system may be as follows

1) Name
2) Purpose
3) Inputs
4) Outputs
5) Functions
6) Performance
7) Manufacturing cost
8) Power
9) Physical Size & weight

Specifications

→ The Specification is more precise it serves as the contract between the customer and the architects.

→ It should be clearly written then only system meets customer's requirements.

→ It is essential to design with minimum of designer effort.

→ It will guide the designer towards the purpose of the system & deadline.

→ It should be understandable enough so that someone can verify that it meets system requirements & overall expectations of the customer.

Architecture Design

→ The architecture is a plan for the overall structure of the system.

→ The creation of of the architecture is the first phase of the design

→ The architecture in the form of block diagram that shows major operations and dataflow among them.

→ The block diagram is still quite abstract and not specified software running on the CPU & special purpose hardware.

→ The architectural description must be designed to satisfy both functional & non-functional requirements.

→ Modify and refine the block diagram to seperate hardware and software architectures for meeting all specifications.

→ When creating hardware & software architecture, should concentrate on functional elements and non-functional constrains.

Designing with computing platforms.

⇨ Designing Hardware and Software Components

→ The component design effort builds Understanding of architecture & specifications.

→ The components includs both hardwares & software modules.

→ Some of the components are readly available in the market example CPU, and memory chip and many other components

System Integration:

⇒ In this phase, the components are put together as per the architectural plan.

⇒ More bugs are found during integration and it can easily find if a good planning is adopted.

⇒ A simple bug in early stage will be a complex one at later.

⇒ By building up the system in phases and running properly chosen test, then can easily find the bugs.
i.e. test the functions relatively independently.

⇒ System integration is difficult & challenging because it usually uncovers problems.

Formalisms for System Design.

⇒ There are number of design tasks at different levels of abstraction in embedded design.

⇒ They are requirements, specification, architecturing the system, designing the code & designing test.

⇒ There is a visual language that can be used to capture all these design tasks known as Unified Modeling Language (UML).

⇒ UML was designed to be useful to many levels of abstraction in the design process.

⇒ UML is very useful because it supports design process by successive refinement & further adding details to the design rather than redesign.

⇒ UML is an object-oriented modeling language and it shows two concepts of importance

1) It supports the design to be described as a number of interacting objects rather than a large blocks of code.

2) Objects will correspond to the real pieces of software or hardware in the system.

The object oriented specification can be seen in two concepts of complementary ways:

i) Object-oriented specification allows a system to be described in a way that closely models real-world objects and their interactions.

ii) Object-Oriented specification provides a basic set of primitives that can be used to describe systems with particular attributes, irrespective of the relationships of those systems component to real-world objects.

Object Oriented Specification may not be executable when compared with Object Oriented Programming language.

But both languages provides similar basic methods for structuring large systems.

UML is a large & rich, having many graphical elements. It needs more care to use correct drawing to describe something.

⇒ Model Train Controller : A design example.

Structural Description

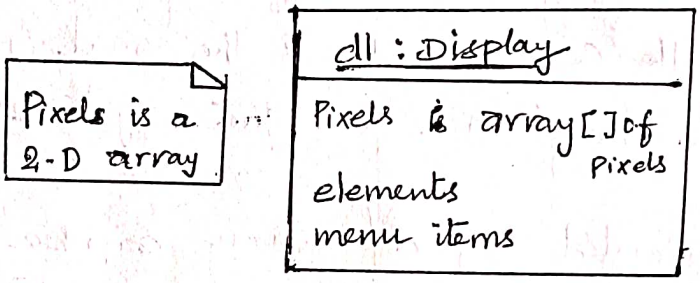
→ The structural description gives basic components of the system.

→ Designer can easily learn to describe those components.

→ The principle component of an object oriented design is the object.

→ An object includes a set of attributes that defines its internal state

→ An object describing a display is shown in UML notation as shown below



An object in UML notation

→ The object is identified in two ways

1) It has a unique name & it is a member of a class
2) The name is underlined to show that this is a description of an object and not of a class.

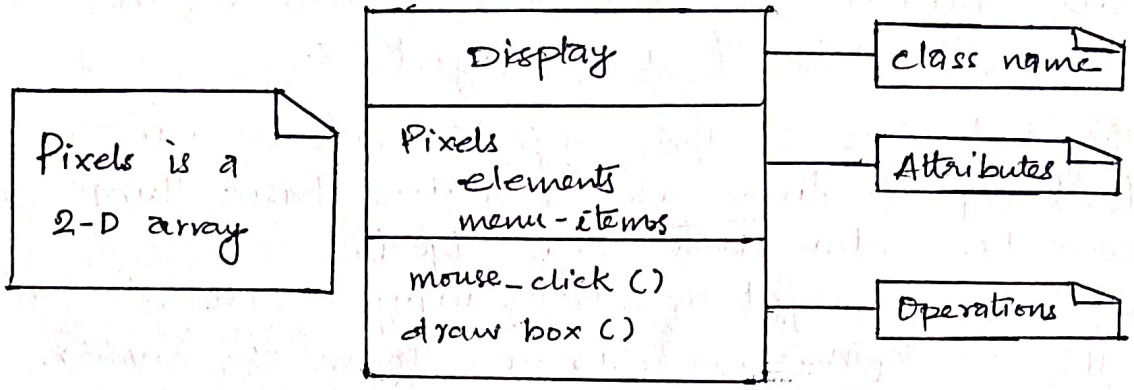
→ A class is a form of type definition all objects derived from the same class have the same characteristics, but their attributes may have different values.

→ A class defines the attributes that an object may have.

→ It also defines the operations that determine how the object interacts with the rest of the world.

The UML description of the display class is shown as follows.

A class in UML notation



→ A class defines both the interfaces for a particular type of object and that objects implementation.

→ When we use an object, we do not directly manipulate its attributes, we can only read or modify the objects state through the operations that define the interface to the object.

→ The proper interface must provide ways to access the objects state as well as ways to update the state.

→ There are several types of relationships that can exist between objects & classes.

1) **Association** It occurs between objects that communicate with each other but have no ownership relationship between them.

2) **Aggregation** It describes a complex object made of smaller objects.

3) **Composition** It is a type of aggregation in which the owner does not allow access to the component objects.

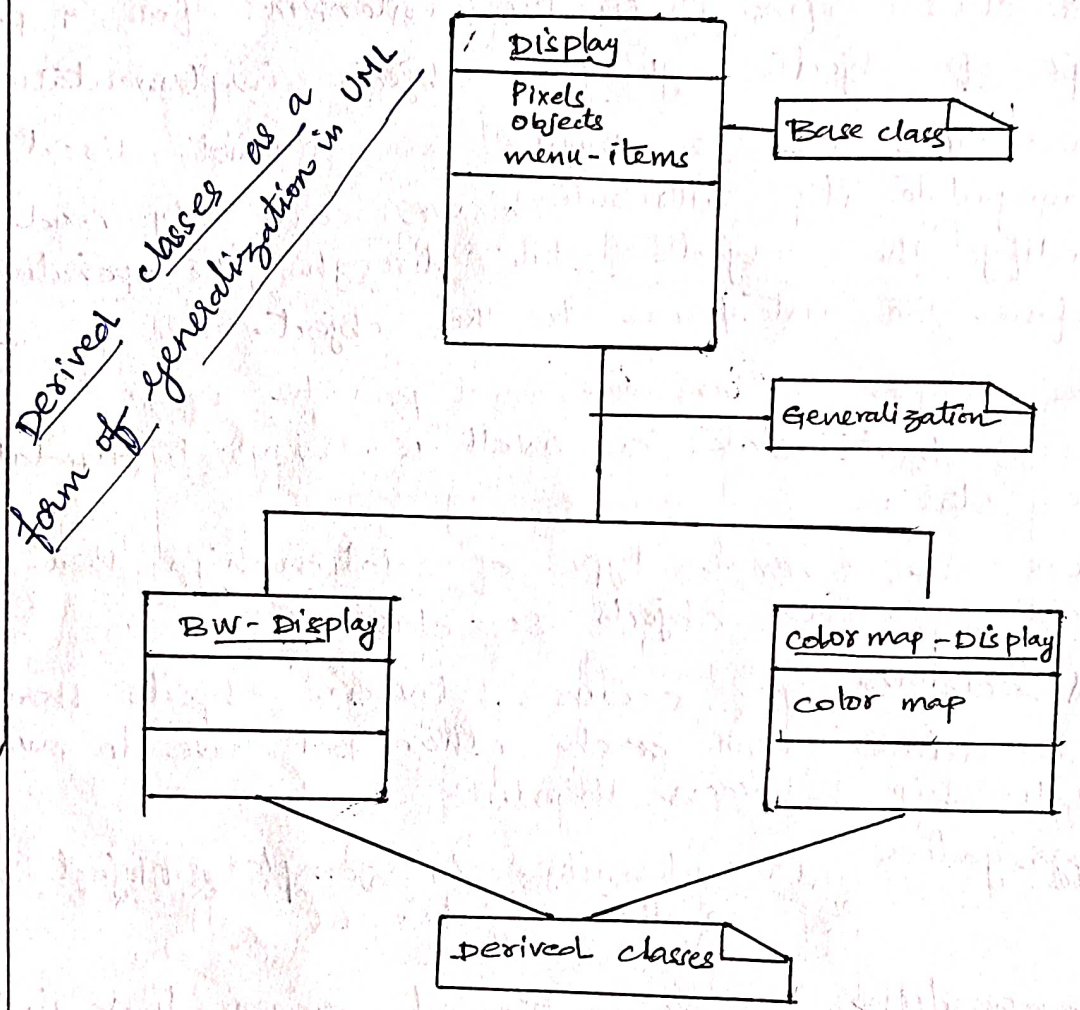
4) **Generalization** It allows us to define one class in terms of another.

The elements of a UML class or object do not necessarily directly correspond to statements in a programming language, if the UML is intended to describe something more abstract than program, there may be a significant gap between the UML & a programming.

→ An attribute is some value that reflects the current state of the object.

→ The behavious of the object must be in a higher level specification and contains basic things that can be done with an object.

→ like most-object-oriented languges UML also allows us to define one class in terms of another.



→ A derived class inherits all the attributes & operations from its base class.

→ A derived class is defined to include all the attributes of its base class.

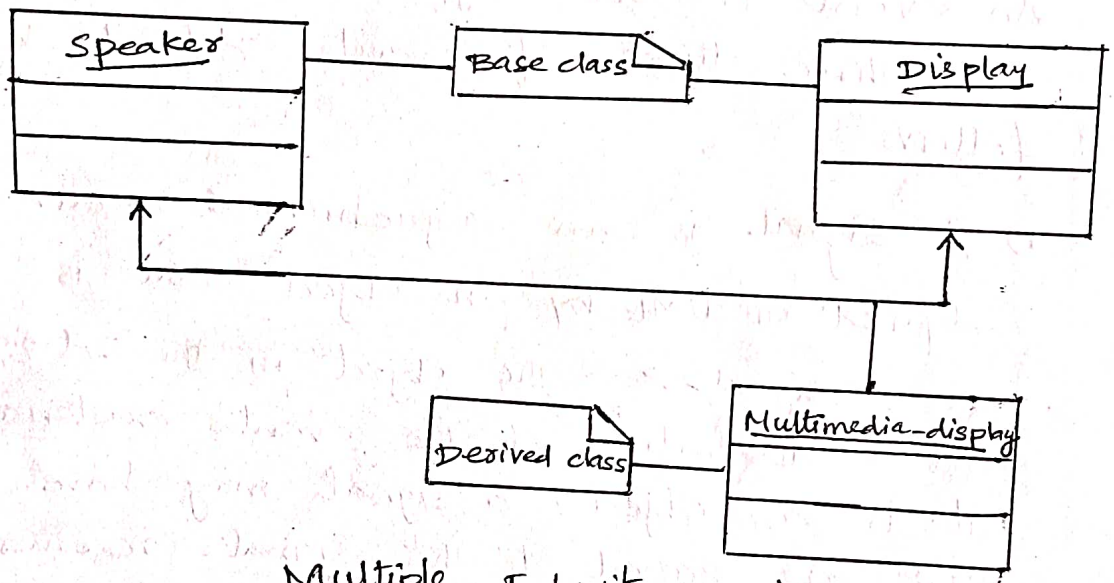
→ From our example display is the base class & BW display and color map display are the two derived classes.

Inheritance

⇨ UML considers inheritance to be one form of generalization

⇨ A generalization relationship is shown in UML diagram as an arrow with an open arrow head.

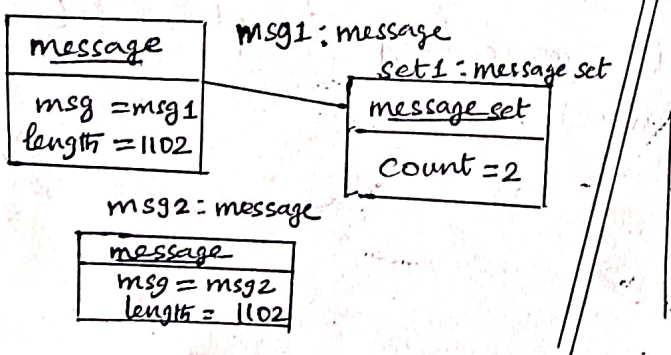
⇨ Both BW-display & color-map display are specific versions of display UML also allows us to define multiple inheritance.



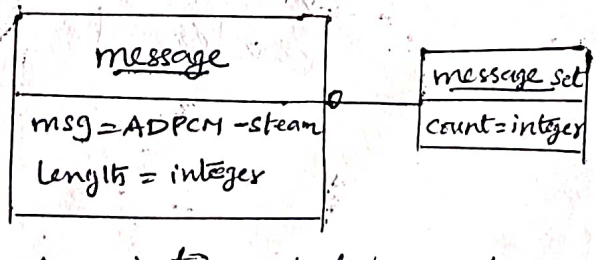
Multiple Inheritance in UML

⇨ Multiple inheritance means which a class is derived from more than one base class.

⇨ A like describes a relationship between objects.
⇨ Association is to like as class is to object.
⇨ Link used to make objects to stand and association capture type information about these links.



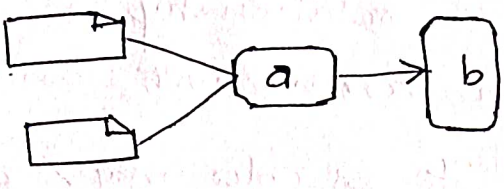
Link between Objects



Association between classes

Behavioral Description

⇒ One way to specify the behaviour of an operation is a state machine.



This state machine shows change from one state to another are triggered by the occurrence of events.

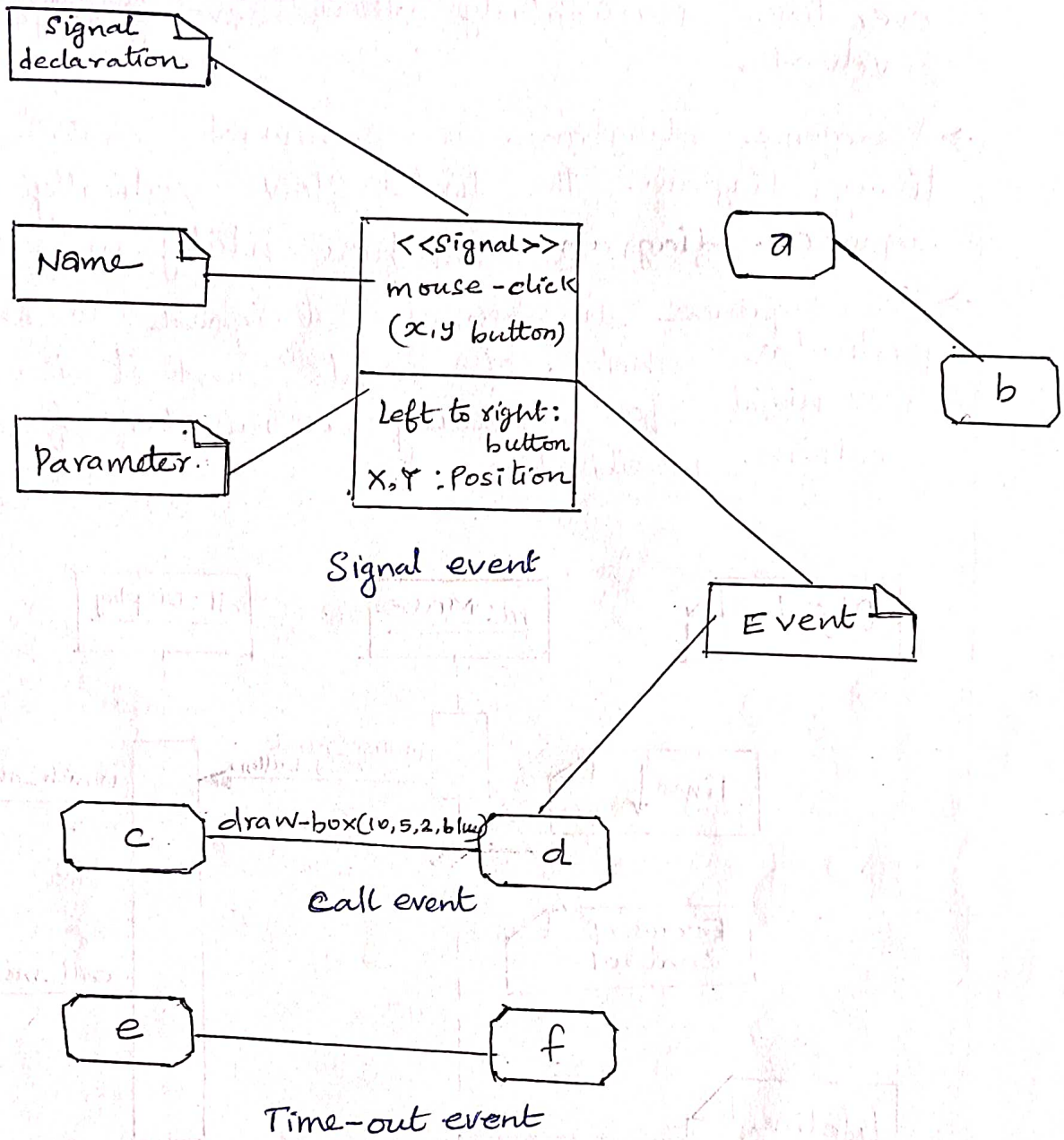
An event is some type of action & there are three types of events defined by UML as follows

1) A **signal** is an asynchronous occurance. It is defined in UML by an object that is labeled as a <<signal>>. The object in the diagram serves as a decleration of the event's existence. Because it is an object, a signal may have parameters that are passed to the signal's receiver.

2) A **call event** follows the model of a procedure call in programming language.

3) A **time-out event** causes the machine to leave a state after a certain amount of time. The label tm(time-value) on the edge gives the amount of time after which the transition occurs. A timeout is generally implemented with an external timer.

Behavioral Description

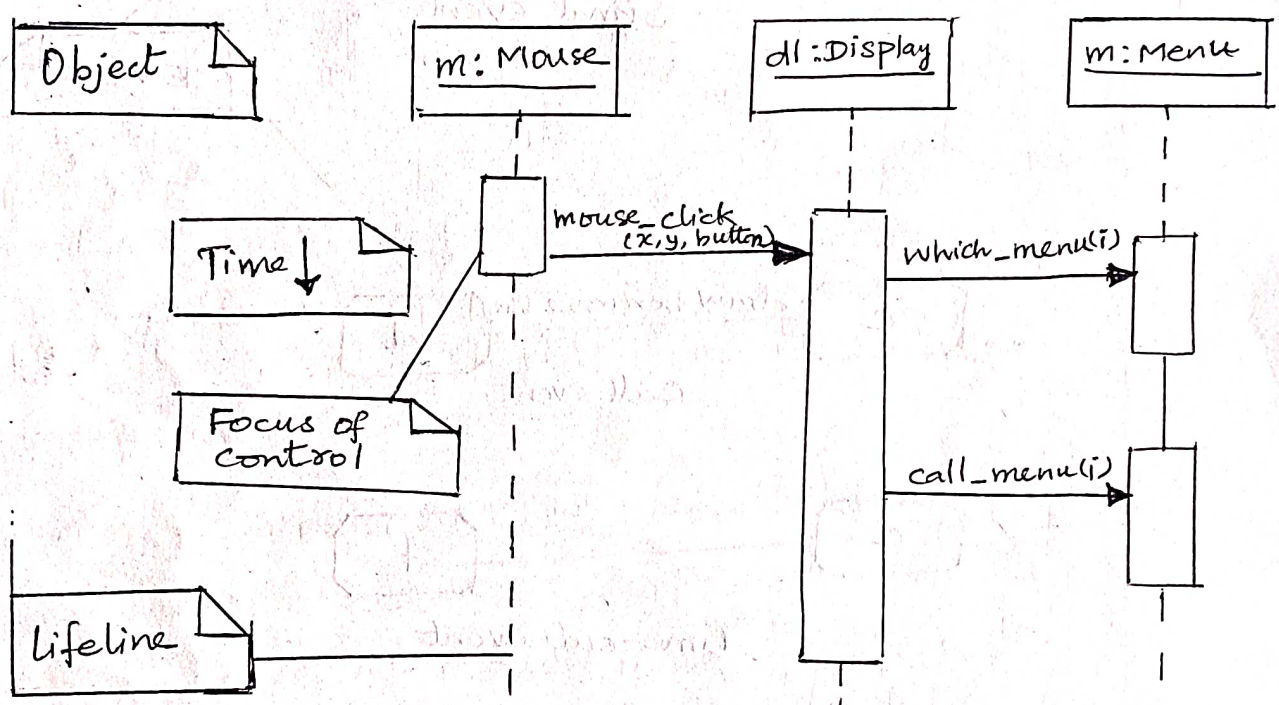


Sequence Diagram

→ It is useful to show the sequence of operations over time, particularly when several objects are involved.

→ A sequence diagram is somewhat similar to hardware timing diagram, the time flows vertically in a sequence diagram & horizontally in a timing diagram.

→ The sequence diagram is designed to show a particular choice of events and it is not convenient for showing a number of mutually exclusive possibilities.



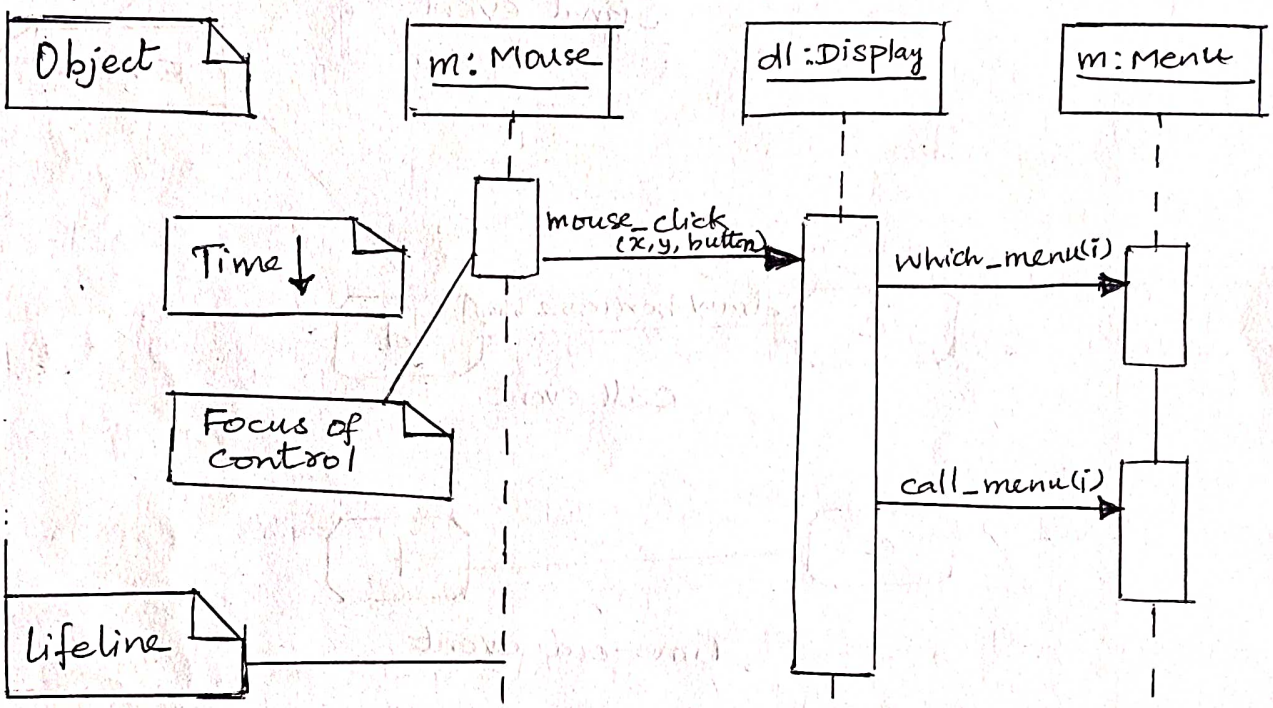
A sequence diagram in UML

Sequence Diagram

→ It is useful to show the sequence of operations over time, particularly when several objects are involved.

→ A sequence diagram is somewhat similar to hardware timing diagram, the time flows vertically in a sequence diagram & horizontally in a timing diagram.

→ The sequence diagram is designed to show a particular choice of events and it is not convenient for showing a number of mutually exclusive possibilities.



A sequence diagram in UML

If messages can interfere with each other in the network, analyzing communication delay becomes difficult

The message delay $\boxed{t_y = t_d + t_x}$

Where t_d = the network availability time delay incurred (got) waiting for the network to become available.

t_d depends on the type arbitration used.

⇒ If the network uses fixed priority arbitration, the network availability delay is infinite for all but the highest-priority device.

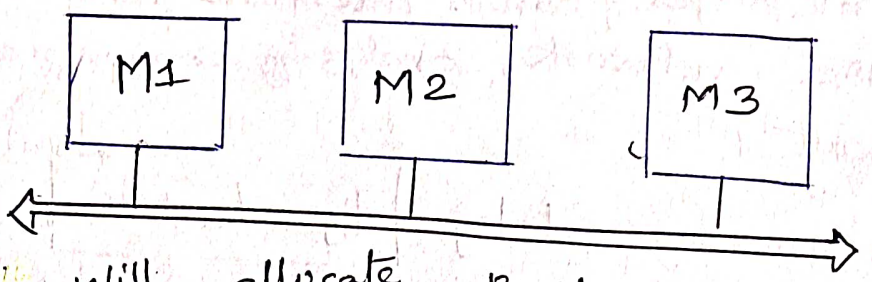
⇒ Since the highest-priority device always gets the network first, unless there is an application-specific limit on how long it will transmit before terminate the network, it keeps blocking the other devices indefinitely.

⇒ If the network uses fair arbitration. the network availability delay is finite.

⇒ In the case of round robine arbitration, if there are N devices, then the worst-case network availability delay is $N(t_x + t_{arb})$, where t_{arb} is the delay got for arbitration & it is small compared to transmission time.

⇒ Even when round-robine arbitration is used to bound the network availability delay, the waiting time can be very long

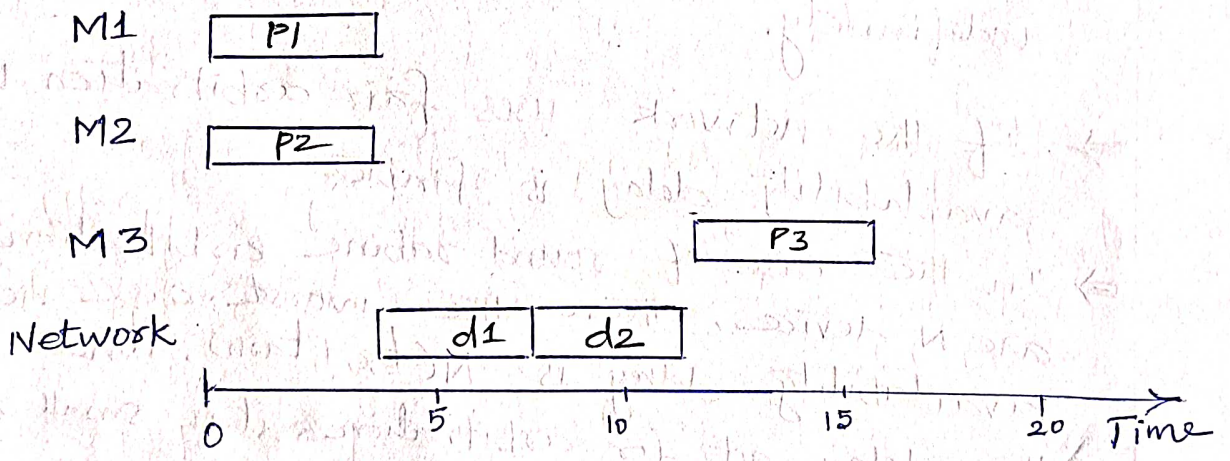
⇒ It is worthwhile to examine the application to determine whether the message structure can be readjusted to reduce t_d



⇒ We will allocate P1 to M1, P2 to M2 & P3 to M3. P1 & P2 runs for three time units while P3 runs for 4 time units.

⇒ A comple transmission of either d1 or d2 takes 4 time units. The task graph shows that P3 cannot start until it receives its data from both P1 & P2 over the bus network.

⇒ The simplest implementation transmits all the required data in one large message, which is 4 packets long in this case. Appearing below is a schedule based on that message structure



⇒ P3 does not start until time 11, when the transmission of the second message has been completed. The total schedule length is 15

⇒ Let's redesign P3 so that it doesn't require all of both messages to begin. We modify the program so that it reads one packet of data each from d1 & d2 and start computing on that. Now it picks up the packets & keeps computing.

⇒ This organization allows us to take advantages of concurrency between M_3 (PE) & the network shown by the schedule below.

⇒ Reorganizing & concurrent execution reduces the scheduling length from 15 to 12,

⇒ When a low priority message is on the network, the network is effectively allocated to that low-priority message, allowing it to block higher-priority messages, but it can slow down critical communication.

⇒ The only solution is to analyze network behaviour to determine whether priority inversion cause some messages to be delayed for too long.

⇒ A round-robine arbitration networks put all communications at same priority. This does not eliminate the priority inversion problem because processess still have priorities.

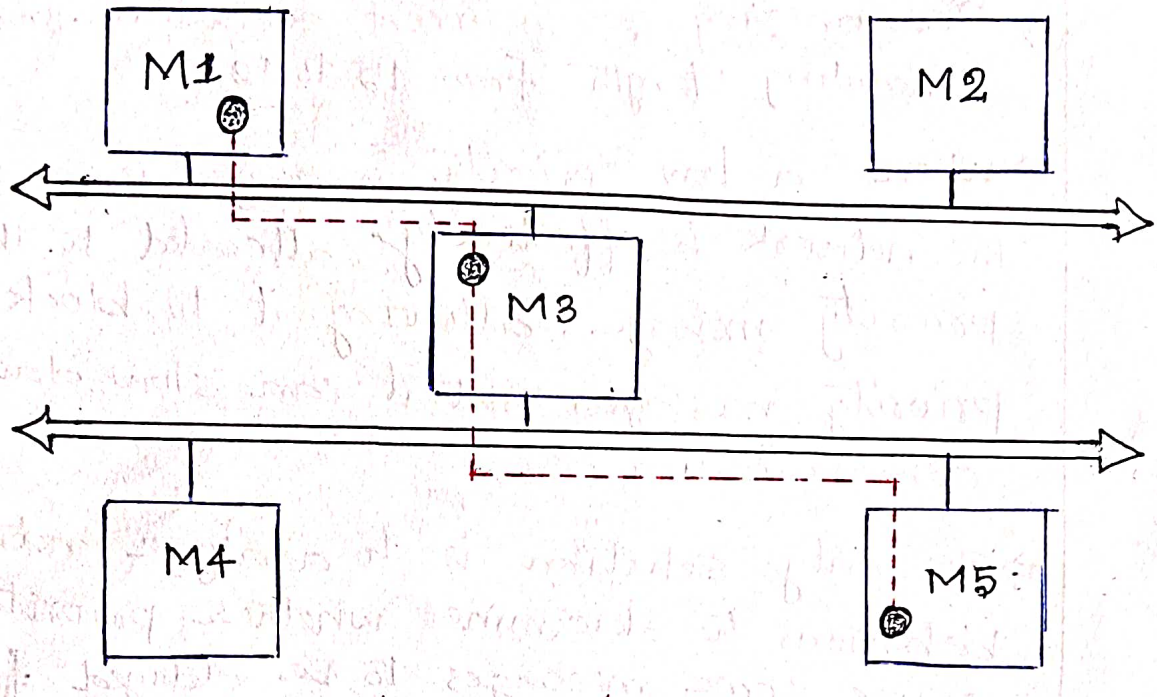
Single-hop network: A mess is received at its intended destination directly from the source, without going through any other networks node.

Multihop networks in which messages are routed through network nodes to go to their destinations.

⇒ Here the hardware platform has two seperate networks, but there is no direct path from M_1 to M_5. The message is therefore routed through M3, which read it from one network & send it on the other one.

⇒ Analysing delay is very difficult, because the time that the message is held at M3 depends on both computational load of M3 & the other message that it must handle.

⇒ If there is more than one network, we must allocate communication to the networks so that it works based on priority.



A Multihop communication

⇒ Scheduling and allocation of computations and communications are clearly interrelated

System Performance Analysis

⇒ Analyzing the performance of distributed embedded system is very difficult.

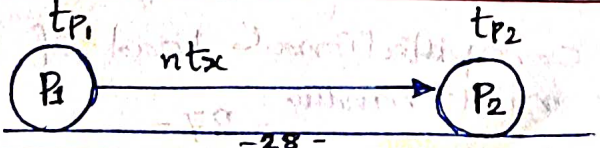
⇒ The figure below in this page shows a very simple task graph with two processes and one n-packet data communication.

⇒ The worst-case execution times of the processes are t_{P1} and t_{P2} and the communication takes ntx time units.

⇒ There is no inteference from outside element, two processes occurs seperately at a single rate.

∴ The worst-case execution time = $t_{P1} + ntx + t_{P2}$

Delay through a simple task graph

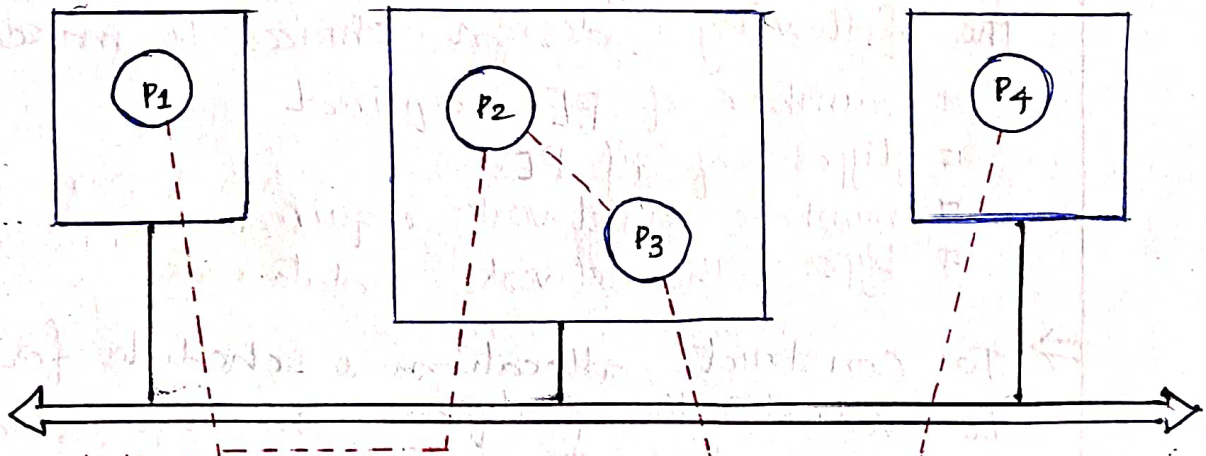


Interference between tasks

⇒ Performance analysis becomes much harder in computations and communications to interfere with each other.

⇒ We have superimpose the task graph on the target architecture, P_2 & P_3 run on the same PE, which helps enable the following chain of events that can affect the whole system.

- The data dependency from P_1 to P_2 translates any uncertainty in the execution time of P_1 into uncertainty about the start time of P_2.
- The coallocation of P_2 & P_3 processing elements M3 means the variations in the ready time of P_2 can affect the completion time of P_3. ofcourse, variations in the execution time of P_2 also affect P_3.
- The data dependency from P_3 to P_4 translates variation in the completion time of P_3 to the start time of P_4.



A distributed System with multirate concurrency

⇒ Even through P_2 & P_3 are seperate tasks but allocated in same PE, therefor messages from the two tasks can interfere with each other on the bus, causing more variation in completion time.

⇨ complex distributed embedded system requires CAD tool to accurately analyze performance.

⇨ By using hand-designing anylize performance, it is necessary take care about to meet hand real-time deadlines, non critical tasks can be turned off temporarily

⇨ When there are several critical tasks must occur simultaneously, hand design requires allocating them to share nothing - no PEs nor communication links.

⇨ While this is a conventional design strategy, it makes hand analysis feasible.

⇨ **Designing with computing Platforms:**

Hardware Platform Design, Allocation and Scheduling.

⇨ Designing the hardward platform is necessarily closely related to our choices in scheduling & allocating processes.

⇨ Creating that schedule requires an allocation of processes to PEs, which in turn requires knowing the available hardware.

⇨ When designing the hardware platform, we have the following design choices to make:

- number of PEs required
- types of all PEs
- numbers of networks required
- types of the networks & data rates.

⇨ To construct allocations & schedules for the processes to evaluate the platform. In turn, allocation & scheduling are driven by system performance analysis.

⇨ A lower bound on the computational needs of the system can be obtained by summing up the worse-case execution times of the processes of follows.

$$t_c = \sum_{\text{Process } i} \frac{T_l}{T_i} t_{Pi}$$

Where t_{pi} = the execution time of process P_i &
T_l = the least-common multiple of the periods T_i

This formula computes the total execution time over the schedule unrolled to the least-common multiple of the periods.

Similarly, we can compute the communication volume over the least-common multiple of the periods

$$n_c = \sum_{\text{Process } i} l_i$$

this formula computes the total number of bytes transmitted in the unrolled schedule by counting the output bytes of all process in the system

⇒ Depending on the type of system we are designing, the following two strategies may be useful to help us quickly come up with an efficient system:

- for I/O-intensive systems we will start with the I/O devices and their associated processing
- For computation-intensive systems we will start with the processes.

I/O-intensive system design.

⇒ I/O are important devices, to support the I/O devices

* Inventory the required I/O devices
* I/O devices that do not require local processing may be attached to the network with the simplest available interface.

* Determine which devices can share a processing element or networ interface.
* Analyze communication time to satisfy communication deadlines.
* Allocate the minimum required PE to go with each I/O device.
* Design the rest of the system using procedure for computation intensive system.

computation-intensive system design

for this we should consider the processes & their deadlines and communication as follows:

* Start with the tasks with the shortest deadlines. If a high-priority-task shares a PE with a low-priority task not only will a more expensive PE be required, but scheduling overhead will be paid for at the nonlinear rate.
* Analyze communication to determine whether critical communications may interfere with each other.
* Allocate low-priority tasks to share PEs where possible.

⇒ The designed systems should meets our performance goals, power consumption & other requirements.

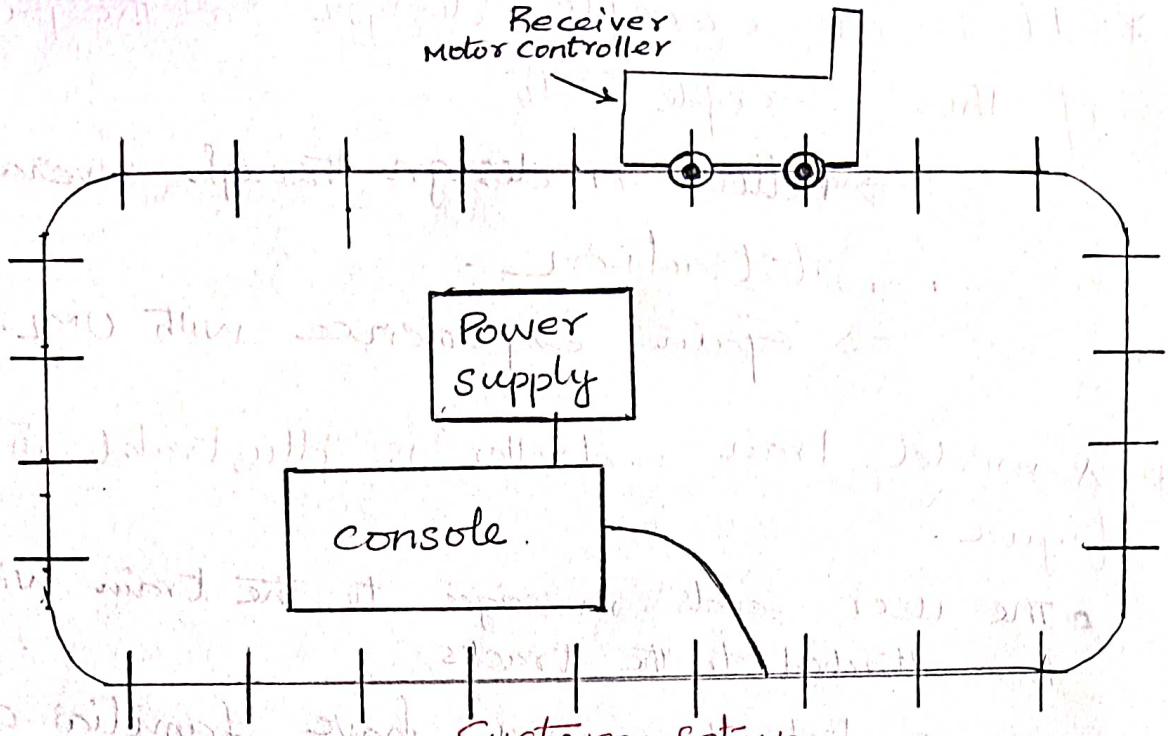
MODEL TRAIN CONTROLLER

* It is an example design and the purpose of this example is to
  - ⇒ Follow a design through several levels of abstraction.
  - ⇒ Gain experience with UML.

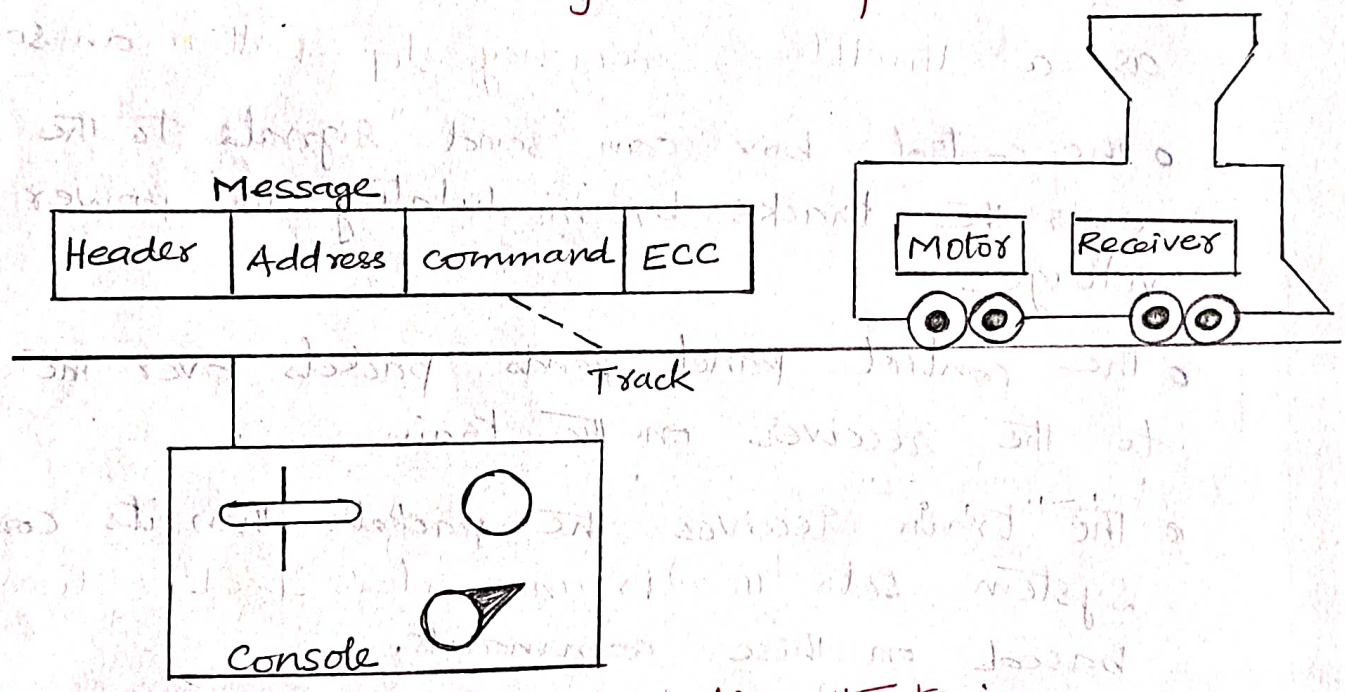
* A model train controller is illustrated in the following figure.
  - The user sends messages to the train with a control box attached to the tracks
  - The control box may have familiar controls such as a throttle, emergency stop button and so on.
  - The control box can send signals to the train over the tracks by modulating the power supply voltage.
  - The control panel sends packets over the tracks to the receiver on the train.
  - The train receives the packets then its control system sets the train motor's speed & direction based on those commands.
  - The packets includes Header, address, command & ECC (Error correction code)
  - This is a one-way communication system – the model train cannot send commands back to the user.

* This model start with the requirements for the train control system and then specifications.

A model train control system



System setup



Signaling the train

Requirements

* console can controll up to 8 trains on a single track
* Throttle has 63 different levels of speed in each direction.

* Inertia control adjusts responsiveness with at least 8 levels
* Emergency stop button
* Error detection scheme on messages.

Requirement Form

Name	Model train controller
purpose	Control speed of up to 8 model trains.
Inputs	Throttle, inertia setting, emergency stop train number
Outputs	Train control signals.
Functions	Set engine speed based upon inertia settings; respond to emergency stop
Performance	Can update train speed at least 10 times per second
Manufacturing cost	$50
Power	10 watts
Physical Size & Weight	approximate size of standard Keyboard & weight < 2 pounds.

DIGITAL COMMAND CONTROL [DCC]

⇨ DCC was created by the National Model Railroad Association to support practical digitally-controlled model trains.

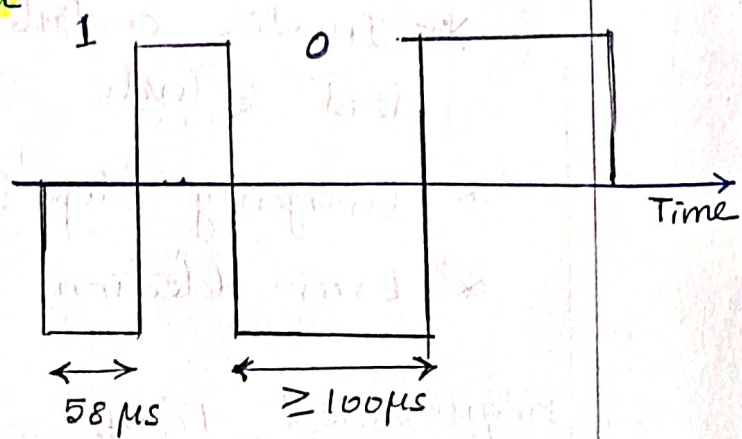
⇨ Hobbyists could mix & match components from several vendors.

⇨ DCC Documents

- Standard S-9.1, the DCC Electrical Standard, defines how bits are encoded on the rails for transmission.
- Standard S-9.2, the DCC communication standards, defines the packets that carry information.

DCC electrical Standard

* Voltage moves around the power supply voltage; adds no DC components
* 1 → 58 µs
  0 → ≥100 µs



Bit encoding in DCC

DCC Communication Standard

the basic packet format ; PSA (8D) +E

- P : Preamble which is a sequence of at least 10 one bits 11111111 1
- S : Packet start bit =0
- A : Address data byte ; it is an eight bits long
- 8S : Data byte start bit =0
- D : Data byte, which includes eight bits
  A data byte contains address, instruction, data or ECC.
- E ; Packet end bit. = 1

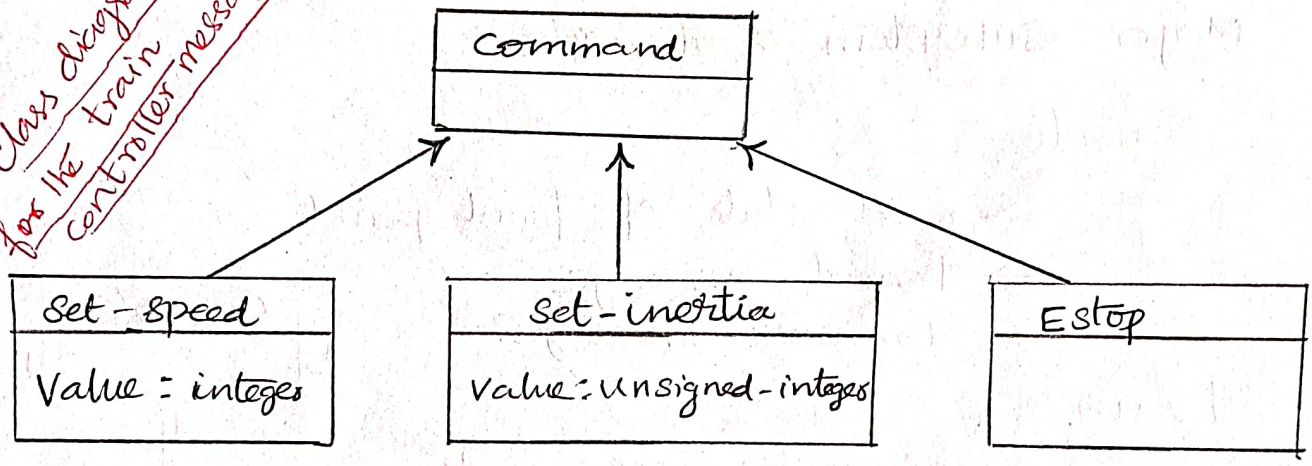
* A packet includes one or more data bytes
* A baseline packet is the minimum packet that must be accepted by all DCC implementation.
* A baseline packet has three data bytes
  (1) Address data bytes that gives the intended receiver of the packet
  (2) Instruction data byte provides a basic instruction
  (3) Error correction data byte is used to detect & correct transmission errors.
* The instruction data byte carrier several informations.
  Bits 0-3 ⇒ 4 bit speed value
  Bit 4 ⇒ additional speed bit & it is a LSB
  Bit 5 ⇒ direction 1 - forward & 0 - reverse
  Bit 7-8 ⇒ Set at 01 for provide speed & direction.
* ECC ⇒ Address bits ⊕ Instruction // The packets are seperated by 5 MS.

Conceptual Specification

* Before create a detailed specification, it is an intial and simplified specification.
- It gives good practice in specification and UML
→ Good idea in general to identify potential problems before investing too much effort in detail.

* In the model train contgol system, there are two major subsystems
1) Command unit
2) Train-board component

Class diagram for the train controller message



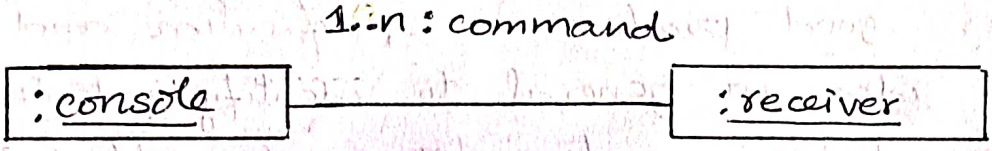
Command name	Parameter
Set-speed	Speed (+ve/-ve)
Set-inertia	inertia value (non-negative)
Estop	non.

Roles of message classes

⇨ Implemented message classes derived from message class
- Attributes & operations will be filled in for detailed specifications.

⇨ Implemented message classes specify message type by their class
- may have to add types as parameter to data structure in implementation.

Subsystem collaboration diagram shows relationship between console & receive (ignores role of track)



Subsystem structure modeling

→ Some classes define non-computer components denoted by *

→ choose important system at this point to show basic relationships.

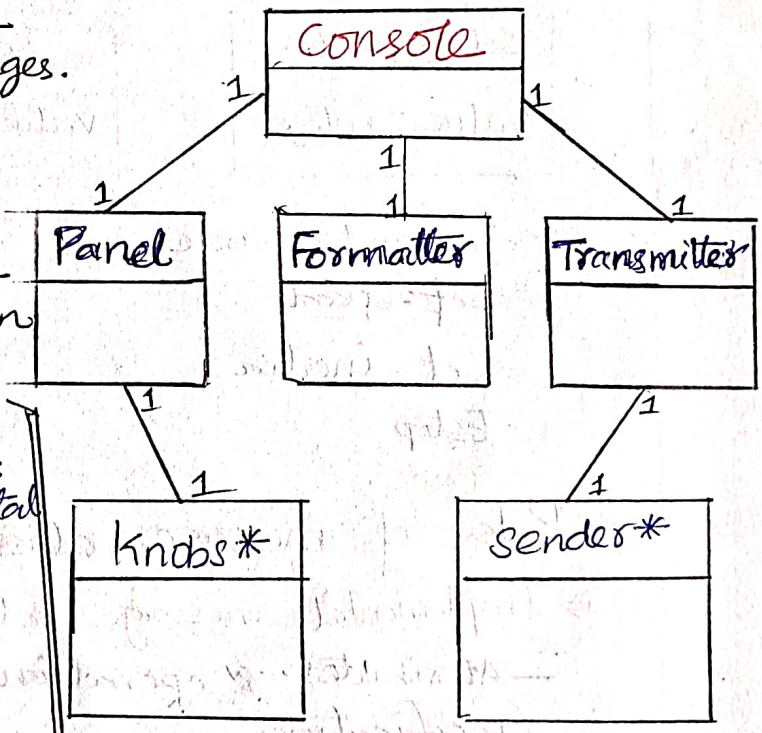
Major subsystem & its roles

Console

→ read state of front panel
→ Format message
→ transmit messages.

Train

→ receive message
→ interpret message
→ control the train



*=physical Object

A UML class diagram for the train controller showing the composition of the subsystem - Console.

Console class roles

Panel : describe the analog knobs and hardware to interface the digital parts of the system

Formatter : turns knob settings into bit (format) streams.

Transmitter : this class interfaces to analog electronics to send the message along the track.

Knobs* : analog knobs, buttons & levers

Sender : Analog electronics that send bits along the track.

-38-

Train Class roles

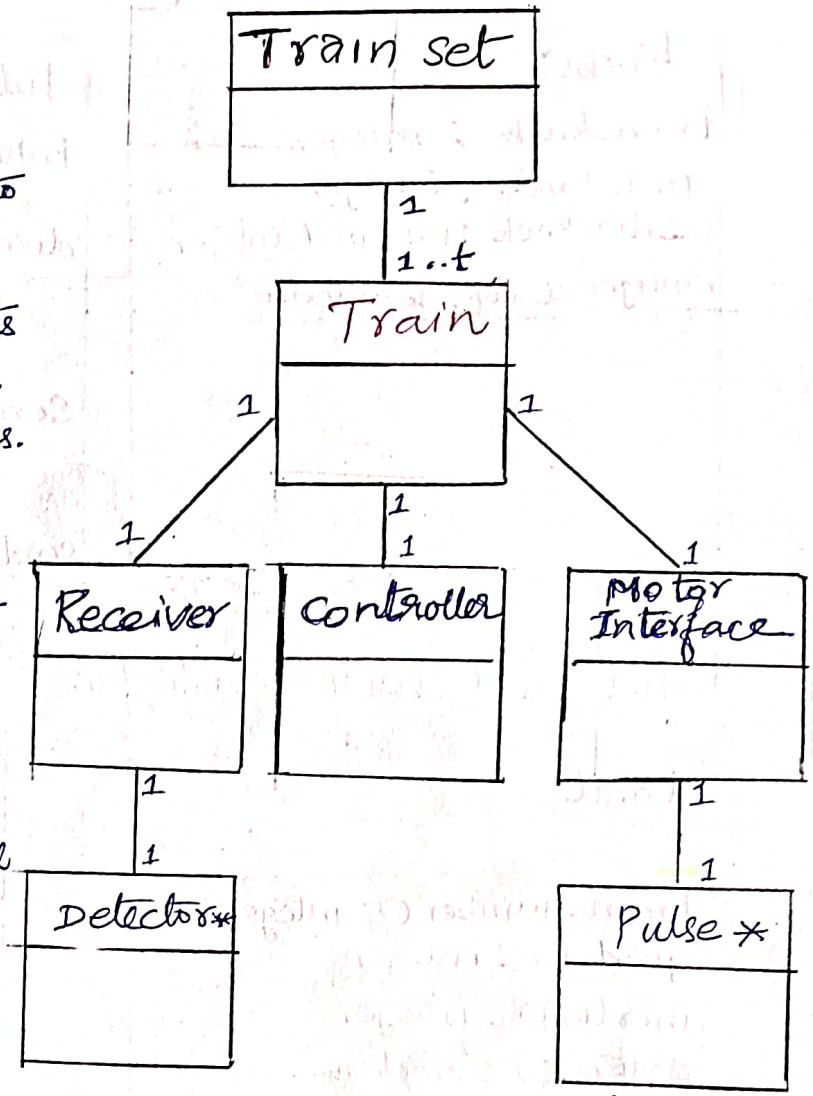
Receiver class knows how to turn the analog signals on the track into digital form.

Controller class interprets received commands & makes control decisions.

Motor Interface class generates signal required by the motor.

Detector* detects analog signals on the track & converts them into digital format

Pulser* turns digital commands into analog signals required to control the motor speed



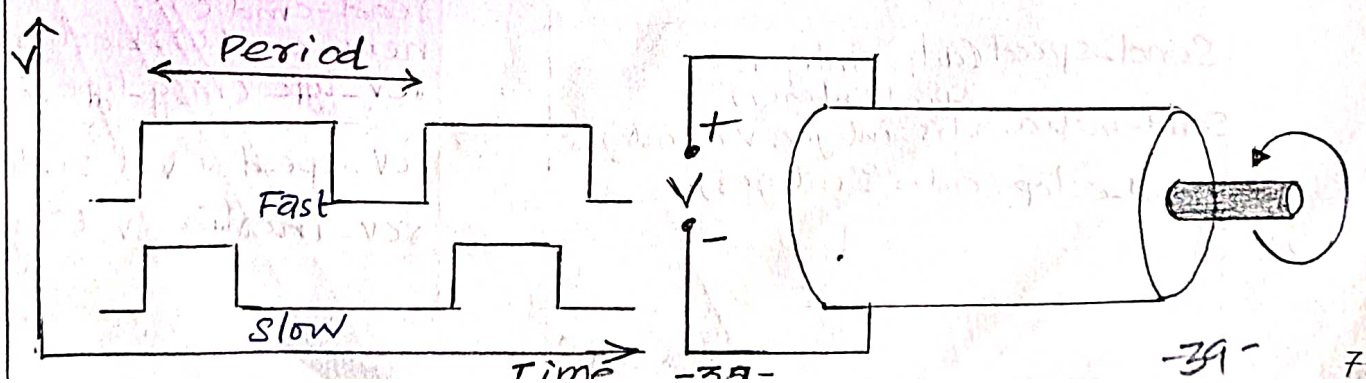
UML class diagram for the train subsystem.

DETAILED SPECIFICATION

⇒ We can now fill in the details of the conceptual specification:
- more classes
- behaviors

⇒ Sketching out the specification first help us to understand the basic relationship in the system.

Train Speed Control : Speed is controlled by PWM



Console physical Object classes

Knobs*
train_knob : integer Speed_knob : integer inertia_knob : unsigned integer emergency_stop: boolean

Pulser*
Pulse_width : unsigned_integer direction : boolean

Sender*
send-bit()

detector*
read-bit() : integer

Panel and motor interface classes

Panel
train_number() : integer speed() : integer inertia() : integer estop() : boolean new_setting()

motor_interface
speed : integer

Class description

* Panel class defines the controls.
  - new_setting() behaviour reads the controls.

* Motor-interface class defines the motor speed held as state

Transmitter & Receiver classes

Transmitter
Send-speed (adrs : integer, speed : integer) Send-inetia (adrs : integer, val : integer) send-estop (adrs : integer)

Receiver
current : command new : boolean
read-cmd() new-cmd() : boolean rcv_type (msg_type : command) rcv-speed (val : integer) rcv-inertia (val : integer

Class description

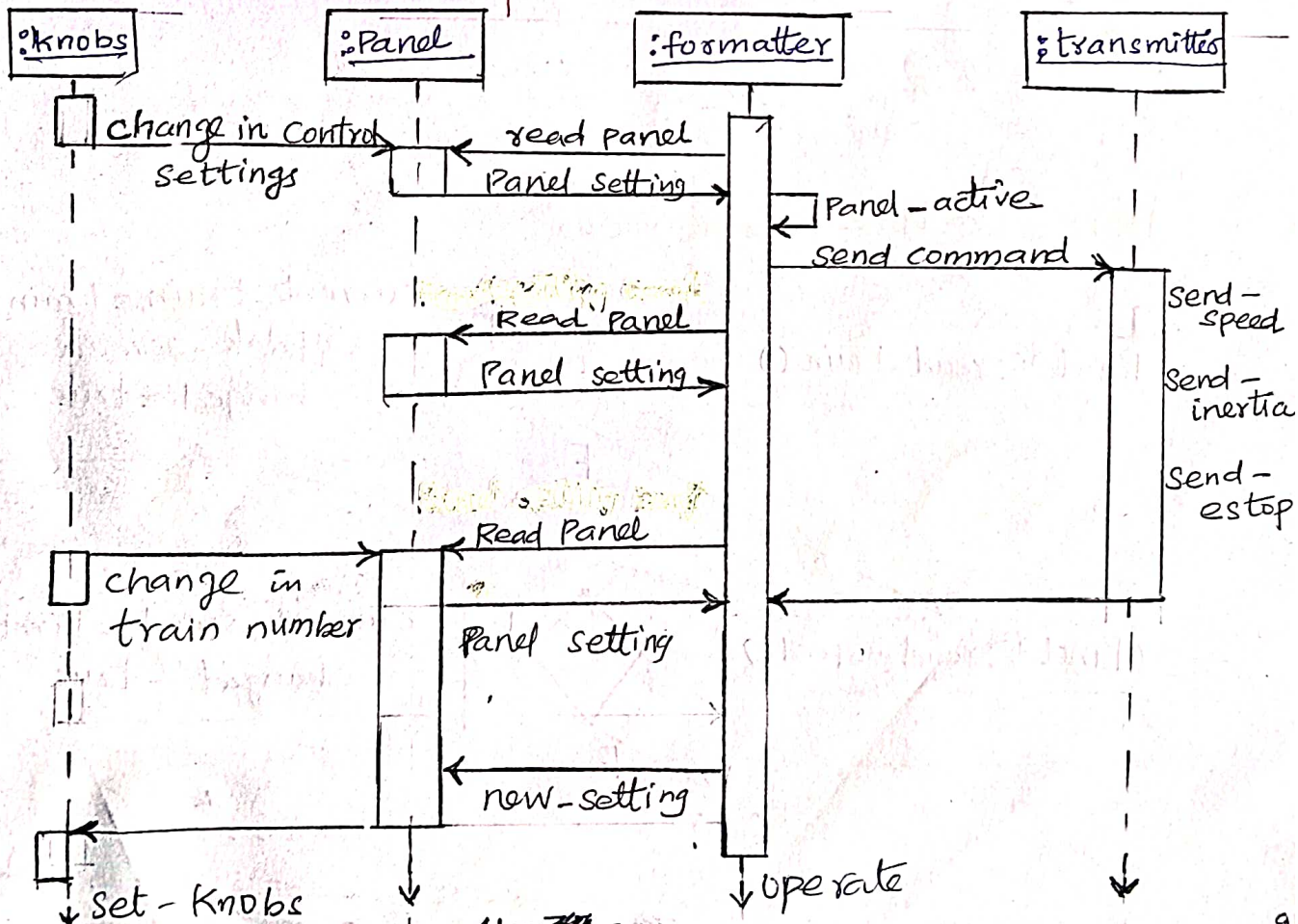
* Transmitter class has one behaviour for each type of message sent
* Receiver function provides methods to:
  - detect a new message;
  - determine its type;
  - read its parameters (estop has no parameters)

Formatter class

* Formatter class description
  - Formatter class holds state for each train setting for current train.
  - The operate () operation performs the basic formatting task.

formatter
current_train : integer current-speed[ntrains] : integer current_inertia[ntrains] : usigned integer current-estop[ntrains] : boolean
send-command () Panel_active () : boolean operate ()

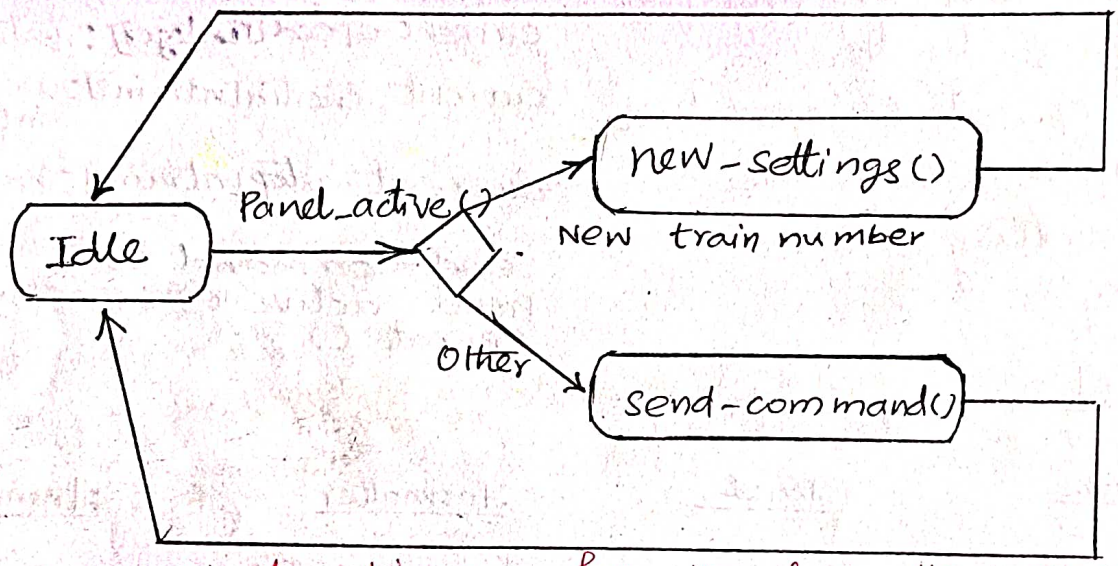
Control input sequence diagram



Control input cases

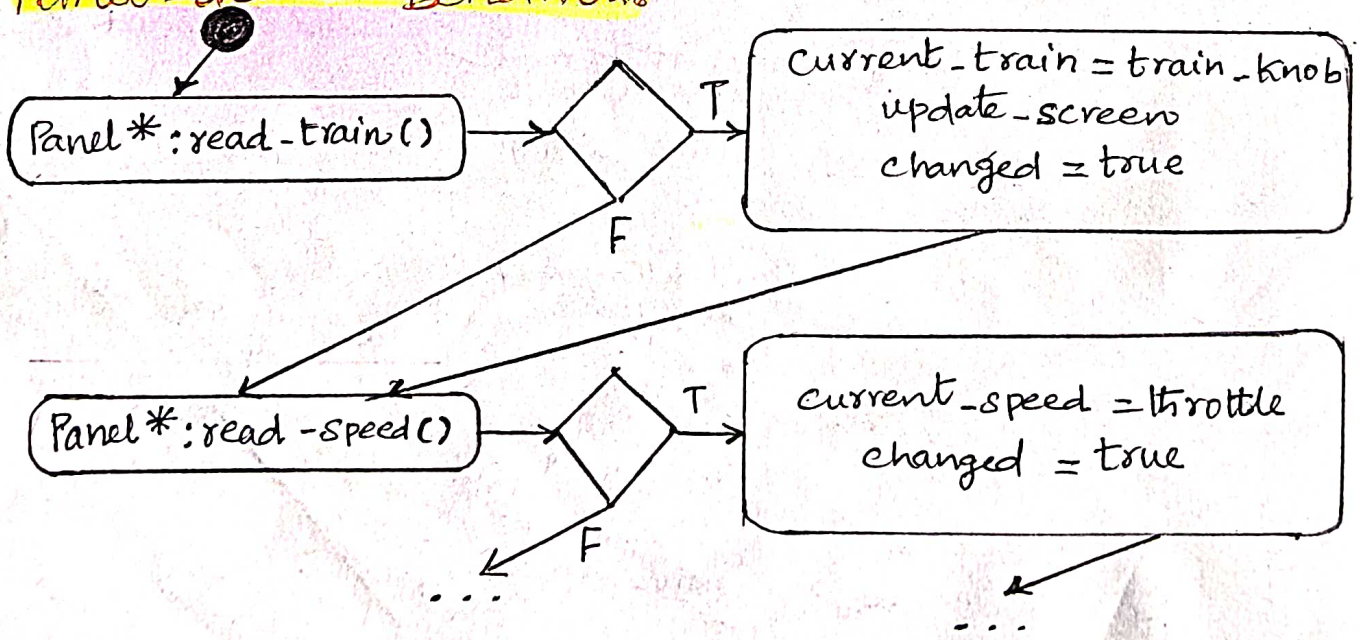
* use a soft panel to show current panel setting for each train
* Changing train number:
  - must change soft panel setting to reflect current train's speed, etc.
* controlling throttle / inertia / estop.
  - read panel, check for changes, perform command.

Formatter Operate Behaviour

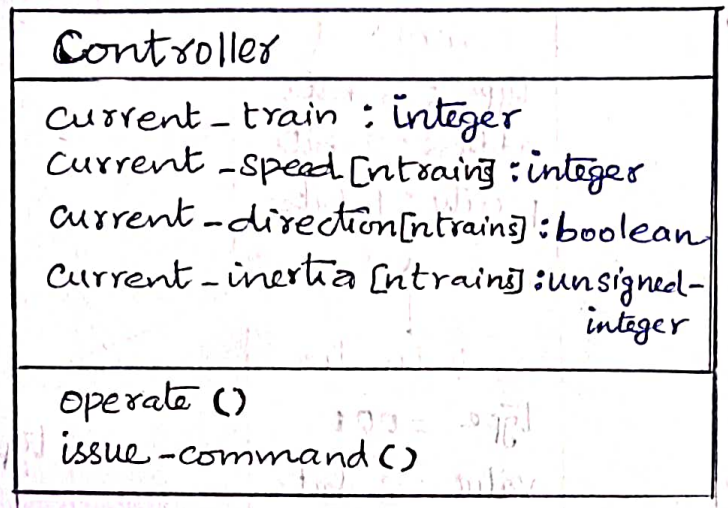


State diagram for the formatter operate behaviour.

Panel-active Behaviour



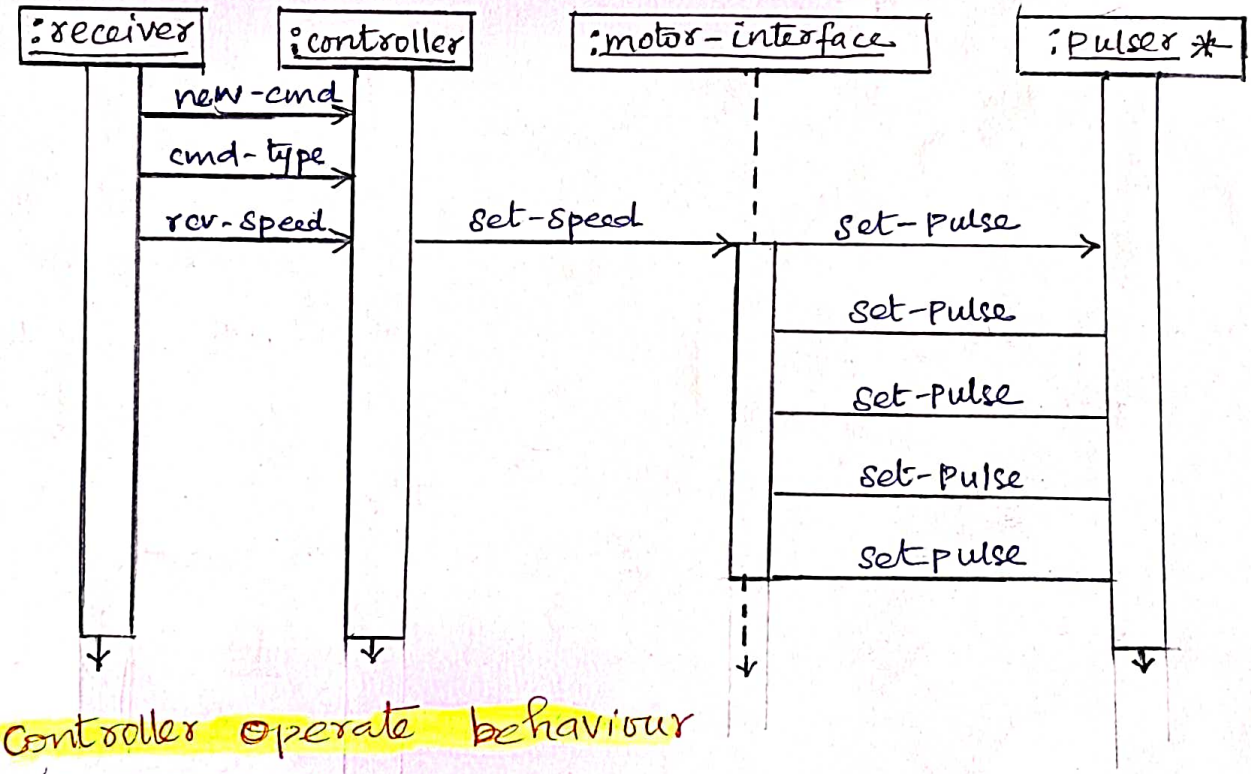
Controller class



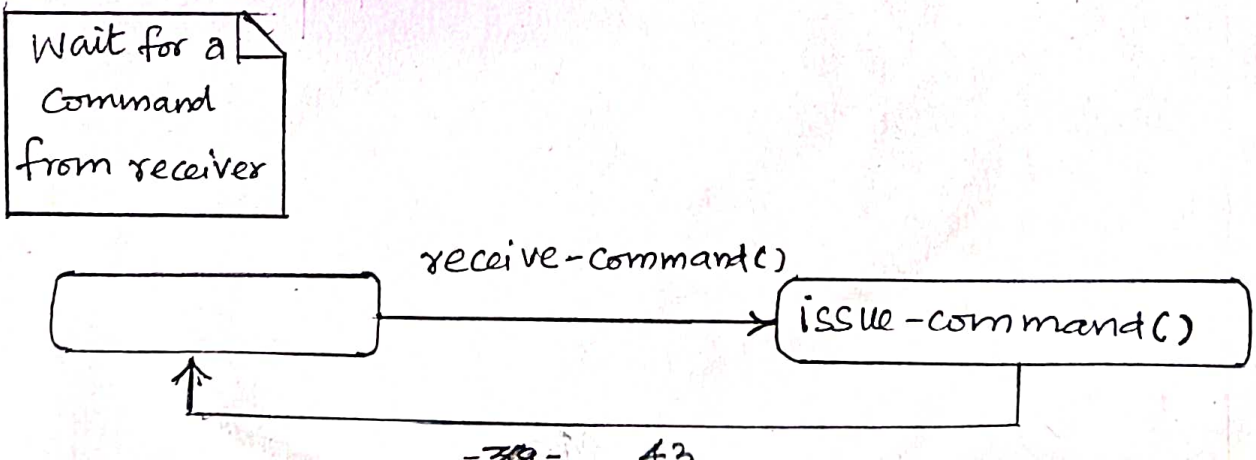
Setting the speed

* Don't want to change speed instantaneously.
* controller should change speed gradually by sending several commands.

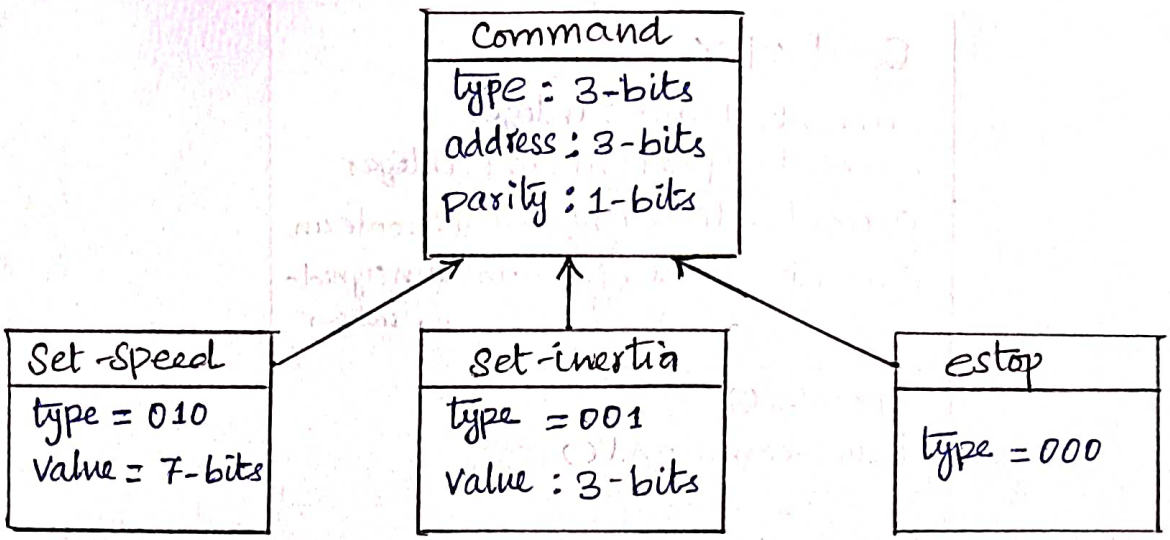
Sequence diagram for set-speed command



Controller operate behaviour



Refined Command classes



⇨ Seperate specification and programming
* Small mistakes are easier to fix in the specification
* Big mistakes in programming cost a lot of time.

⇨ It is not possible to seperate specification & architecture.

Design Methodologies

⇒ An effective design methodology should be defined in terms of design objectives and design choice, to be applied at each consecutive design step.

⇒ The design objectives and choice determine milestones to be achieved during a certain instance of a design process.

⇒ As embedded system are designed to perform dedicated functions, like real-time computing constraints.

⇒ In most cases, they are made of components that communicate with each other and the environment via sensors and actuators.

⇒ Challenges in the design of embedded systems

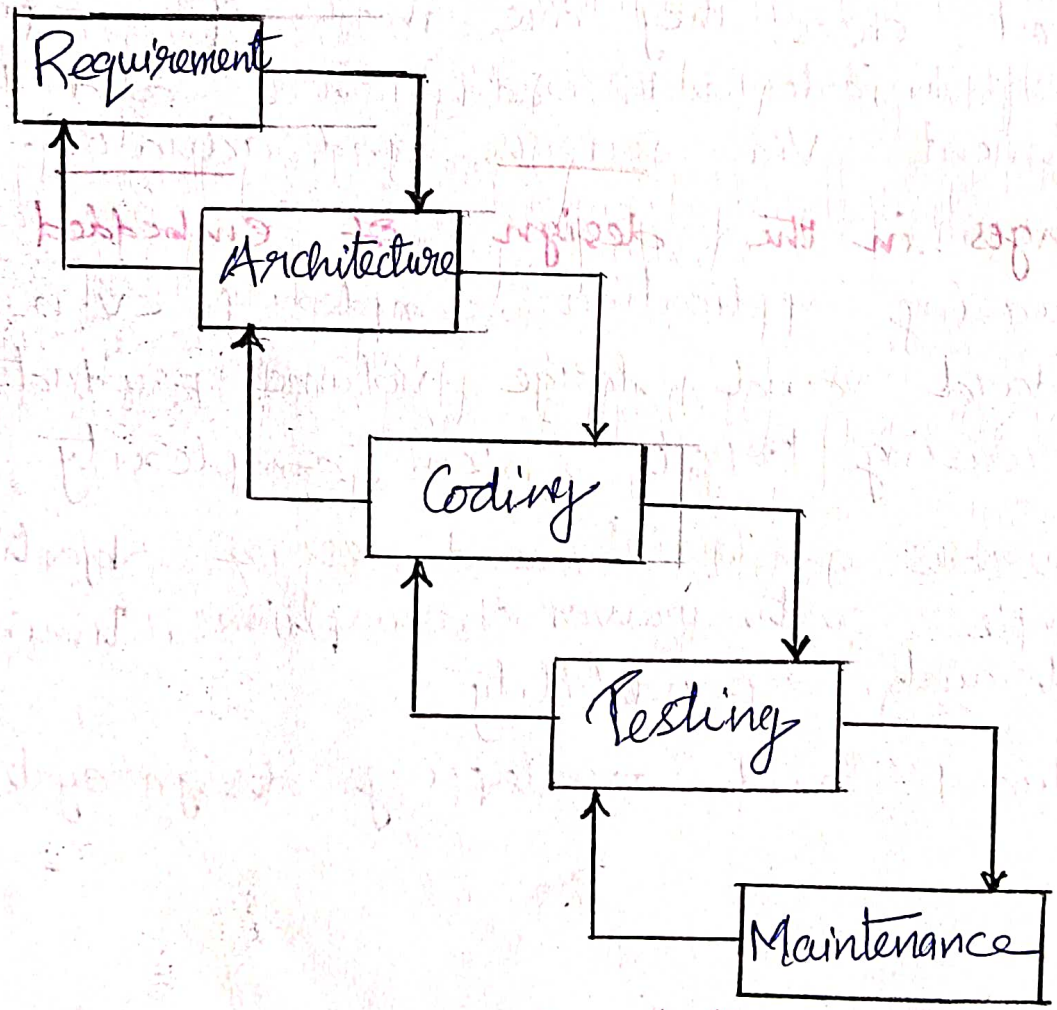
1. Increasing application complexity even in standard and large volume products.
2. Increasing target system complexity.
3. Numerous constraints and design objectives; example : cost, power consumptions, timing constraints, dependability.
4. Reduced and overlapping design cycles.

Design Flow

Design flow is a sequence of steps to be followed during a design. Some of the steps can be performed by tools and other steps can be performed by hand.

Waterfall Model:

The below figure shows the waterfall model introduced by Royce, the first model proposed for the software development process.



Waterfall Model.

In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The outcome of one phase acts as the input for the next phase sequentially.

Requirements: Defines needed information, function, behaviour, performance and interfaces.

Architecture: Data structures, software architecture, interface representations, algorithmic details.

Implementation: Source code, database, user documentation, testing.

Testing: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Maintenance: There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Advantages of Waterfall model:

1. Simple and easy to understand and use.
2. Provides structure to inexperienced staff.
3. Milestones are well understood.
4. Sets requirements stability
5. Good for management control (plan, staff, track)
6. Works well when quality is more important than cost or schedule.

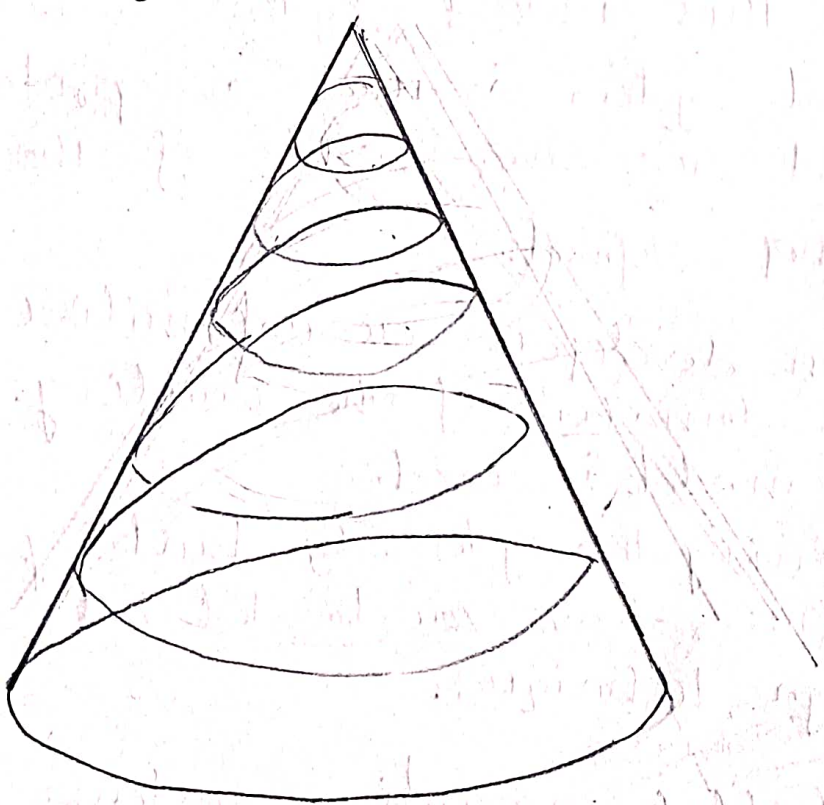
Disadvantages of waterfall model

1. Poor model for long and ongoing projects.
2. High amounts of risk and uncertainty.
3. It is difficult to measure progress within stages
4. Cannot accommodate changing requirements.

Spiral Model:

* The following figure shows spiral model of software design. Spiral model uses a cyclic approach to incrementally develop the software product while decreasing its degree of risk.
* This model uses planning risk analysis, engineering and evaluation phases.
* A software project repeatedly passes through these in cycles.

* The spiral model uses planning, risk analysis, engineering and evaluation phases.
* Projects split into smaller sub-projects: each iteration corresponds to a smaller projects.
* In spiral model, before each cycle to involve stake holders to decide on scope of cycle to arrive at plan.
* Following this risk analysis is carried out to arrive at list of risks. stake holders performs concurrent engineering to adress these risks through pro-typing and before the completion of current cycle.



* Software is produced in the engineering phase and tested at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project before the project continues to the next spiral.

* Spiral models are found very effective for large and mission-critical projects.

* In spiral model, entire project cost is very high since too many risk analysis and pro-typing are involved. Also, spiral model demands highly specific expertise to perform risk analysis.

Successive Refinement Design Methodology:

* The following figure shows successive refinement design methodology.

* In this method, system is built many times First system is used as prototype model and successive models of the system are further refined.

* When developers are unfamiliar with the application domain of the building the system, this model is used.

* Refining the system by building several complex systems allows you to test out architecture and design techniques.

* Embedded computing systems often involve the design of hardware/software project-

UNIT-II

ARM PROCESSOR. (Advanced RISC Machine)

→ The ARM is a 32-bit Reduced Instruction Set Computer (RISC) processor.

→ ARM processors are suitable for low power applications.

→ This has made them dominant in the mobile and embedded electronics market.

→ ARM architecture comes in several versions.

→ CISC ; Complex Instruction Set Computer
These machines provided a variety of instructions that may perform very complex tasks and it generally used a number of different instructions formats of varying lengths.

→ RISC : Reduced Instruction Set Computer
It is developed for high performance microprocessor & its instruction sets are executed in pipelined processors.

It can be used for any types of applications.

→ ARM 7 is a Von Neumann architecture & it supports two basic types of data.

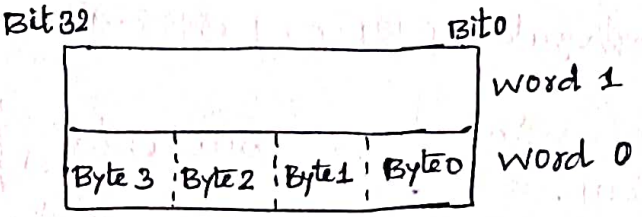
1). The standard ARM word is 32 bit long.

2] The word may be divided into four (8-bit) bytes.

→ ARM 7 allows addresses upto 32 bit long.
here 1 word = 32 bit
word 0 has location 0
word 1 has location 4
word 2 has location 8 and so on.

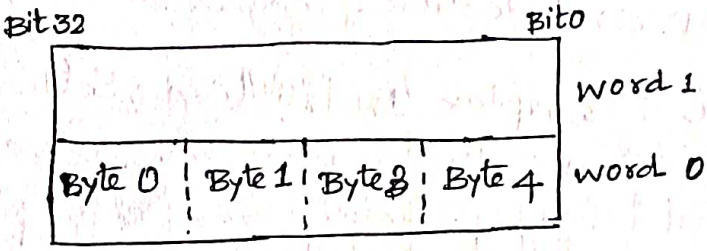
→ ARM processor can be configured at power to address the bytes in a word in either Little-endian or Big-endian mode.

→ In little-endian mode the lower order byte residing in the low order bits of the word.



Little-Endian
Intel x80

→ In big-endian mode the lowest order byte stored in the highest bits of the word.



Big-Endian
Motorola

→ Arithmatic & logical operations cannot be performed directly on memory locations.

→ ARM is a load store architecture and data operands must be loaded into the CPU and then store back to main memory to save the results.

→ ARM has CPSR (Current Program Status Register) in its programming model. This Register is set during every arithmatic, logical or shift operation.

→ When used in relation to the ARM

Byte means 8 bits - 1 byte
Halfword means 16 bits - 2 bytes
word means 32 bits - 4 bytes

→ Most ARM's implement two instruction sets

32-bit ARM Instruction set
16-bit Thum Instruction set

→ Jazell cores can also execute Java byte code.

→ The ARM has seven basic-modes of operations. Each mode has access to its own stack space and a different subset of register.

Some operations can only be carried out in a privileged mode.

-2-

ARM Processor Modes

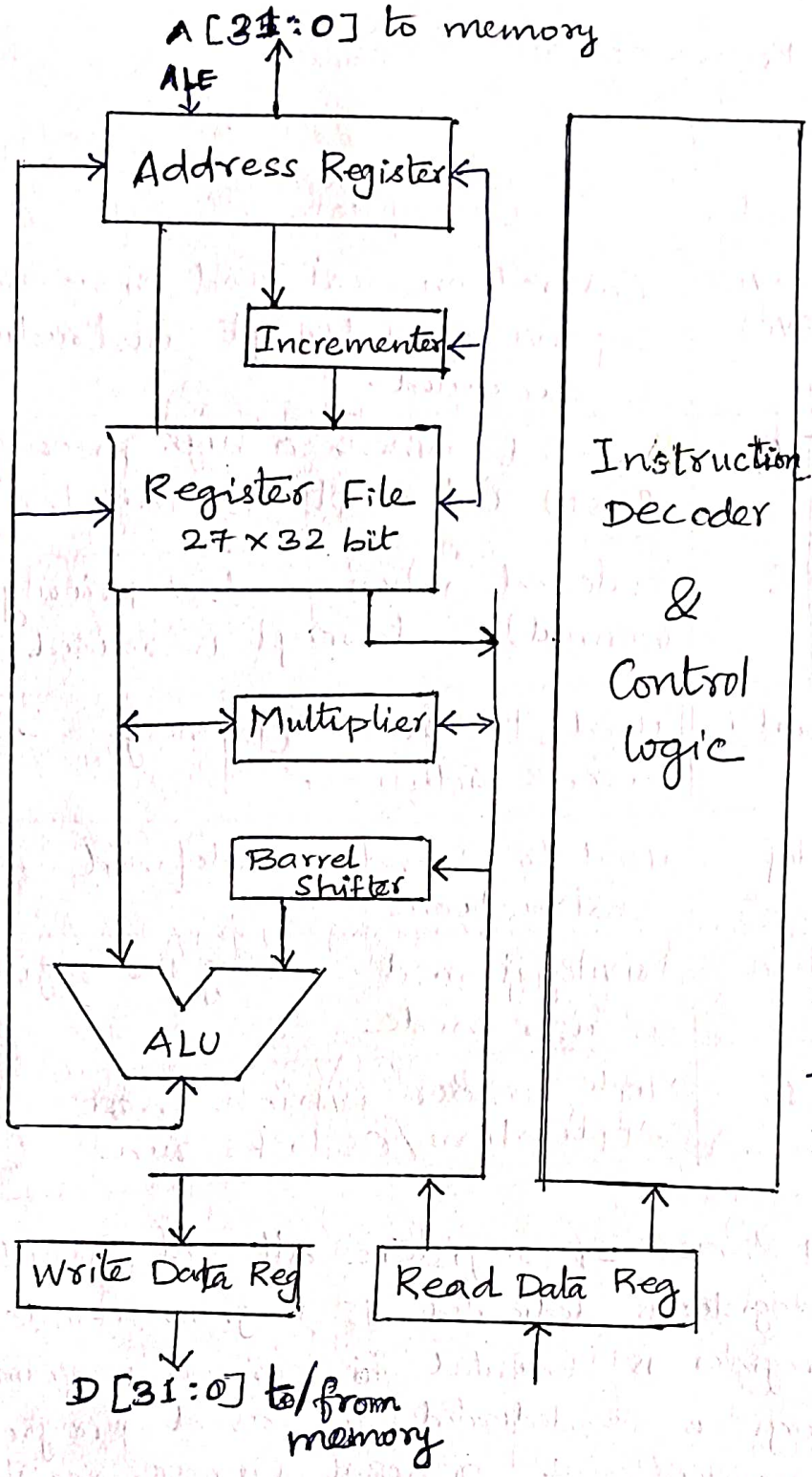
Mode	Description	Remarks
Supervisor (SVC)	Entered on rest and when a simultaneous Interrupt instruction is executed.	Privileged Modes
FIQ	Entered when a high priority (fast) interrupt is raised.	
IRQ	entered when a low priority (normal) interrupt is raised.	
Abort	used to handle memory access violations	
Undef	used to handle undefined instructions	
System	Privileged mode using the register as user mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Exception modes.

⇒ ARM has 37 registers all of which are 32-bits long
- 1 register is dedicated to Program counter.
- 1 register is dedicated to current program status register.
- 5 registers are dedicated to saved program status registers
- 30 registers are general purpose registers.

⇒ The current processor mode governs which of several banks is accessible. Each mode can access
- a particular set of r0-r12 registers
- a particular r13 (the stack pointer, SP) & r14 (the link reg. lr)
- the program counter, r15 (PC)
- the current program status register (CPSR)

⇒ Privileged modes (except system) can also access
- a particular SPSR (saved program status register)



* large uniform register file
* Load/store architecture
* Simple addressing mode
* Uniform & fixed-length instruction fields.
* High performance
* low code size
* low power consumption
* low silicon area
* 16 visible R0-R15
* 31 general-purpose 32-bit reg

→ ARM is an RISC architecture and thereby the operands are not directly fetched into the ALU in fact that they will first store in the register file and then to the ALU.

→ In the code datapath Barrel shifter doing an important role such that it can processes the data & barrel shifter can shift the data into right or left according to the arbitrary value assigned in the code.

ARM Processor Modes

	Mode	Description	Remarks
Exception modes.	Supervisor (SVC)	Entered on rest and when a simultaneous Interrupt instruction is executed.	Privileged Modes
	FIQ	Entered when a high priority (fast) interrupt is raised.	
	IRQ	entered when a low priority (normal) interrupt is raised.	
	Abort	used to handle memory access violations	
	Undef	used to handle undefined instructions	
	System	Privileged mode using the register as user mode	
	User	Mode under which most Applications / OS tasks run	Unprivileged mode

⇒ ARM has 37 registers all of which are 32-bits long
- 1 register is dedicated to Program counter.
- 1 register is dedicated to current program status register.
- 5 registers are dedicated to saved program status registers.
- 30 registers are general purpose registers.

⇒ The current processor mode governs which of several banks is accessible. Each mode can access
- a particular set of r0-r12 registers
- a particular r13 (the stack pointer, SP) & r14 (the link reg. lr)
- the progsam counter, r15 (PC)
- the current program status register (CPSR)

⇒ Privileged modes (except system) can also access
- a particular SPSR (saved program status register)

→ The barrel shifter differ from normal shift register in such a way that they can do all these shifts in a single cycle but shift registers can do the same depends on number of shift required.

→ The barral shifter are combination circuit as like shift registers.

→ Auto increment block can operate register independent of ALU.

→ The Auto incrementer block is followed by a incrementer bus which has access to both register bank (31*32 registers & 6 status registers).

→ The AI block can take the address from Address Register and can operate with the same without stay in queue for ALU to finish it's operation.

→ Fast Interrupt Response.

→ Excellent High level langnage support 'c' Language.

→ Simple & powerful Instruction set.

→ Load/Store multiple instructions.

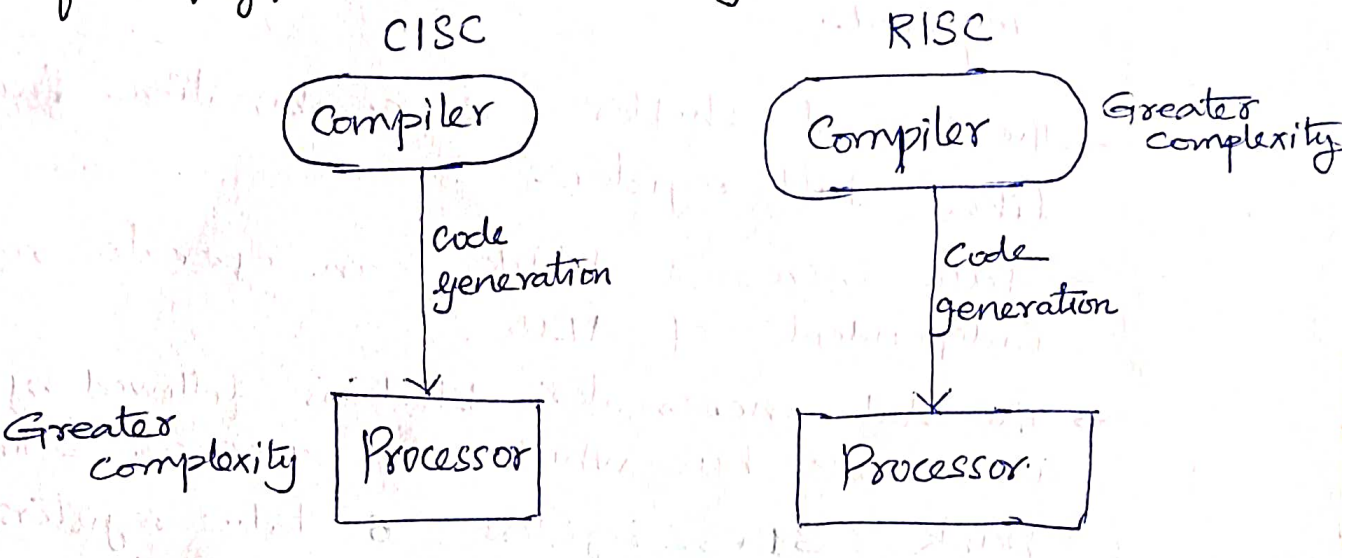
→ Conditional execution.

→ Auto Increment/Decrement addressing mode.

→ ARM7TDMI provides up to 120 Dhrystone MIPS and is known for its high code density & low power consumption, making it deal for Embedded devices.

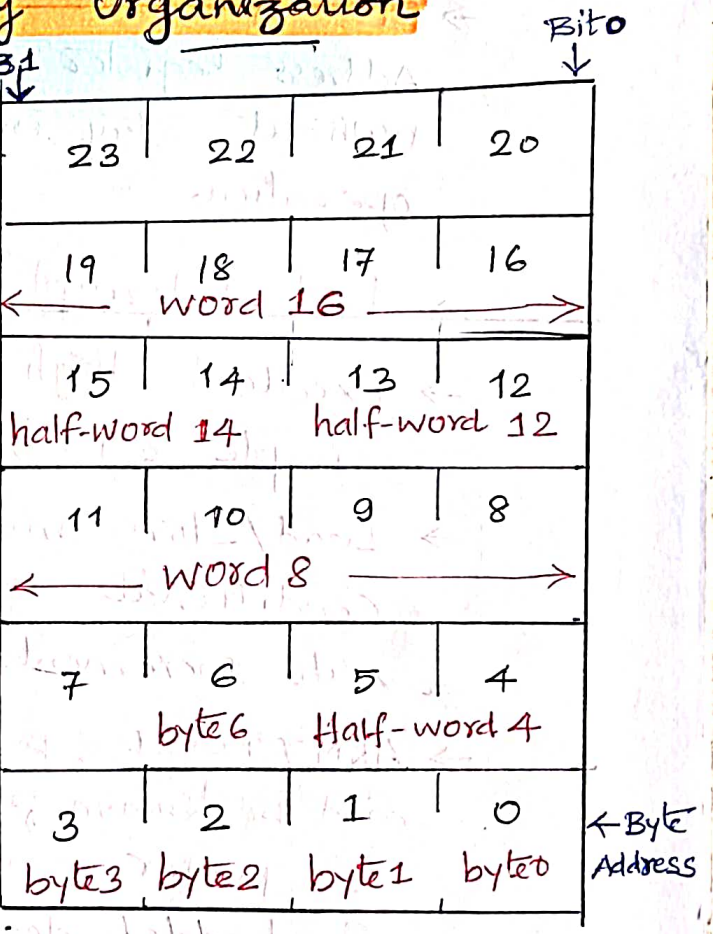
→ RISC machines have a large general-purpose register set. Any register can contain either data or address. Registers act as fast local memory store. In contrast, CISC processor have dedicated reg. for specific purpose.

→ RISC processor operates on data held in register banks. This will reduce multiple memory access and seperate load & store instructions are there for register bank-memory access.



ARM Memory Organization

→ The ARM7 is a Von-Neuman load/store architecture
→ only 32 bit data bus for both instruction & data
→ Memory is addressed as a 32 bit address space.
→ Data byte can be 8-bit bytes, 16-bit half words or 32 bit words and may be seen as a byte line folded into 4-byte words
→ Word will be alligned to a 4 byte boundaries.
→ Half-word must be alligned to 2 byte boundaries.



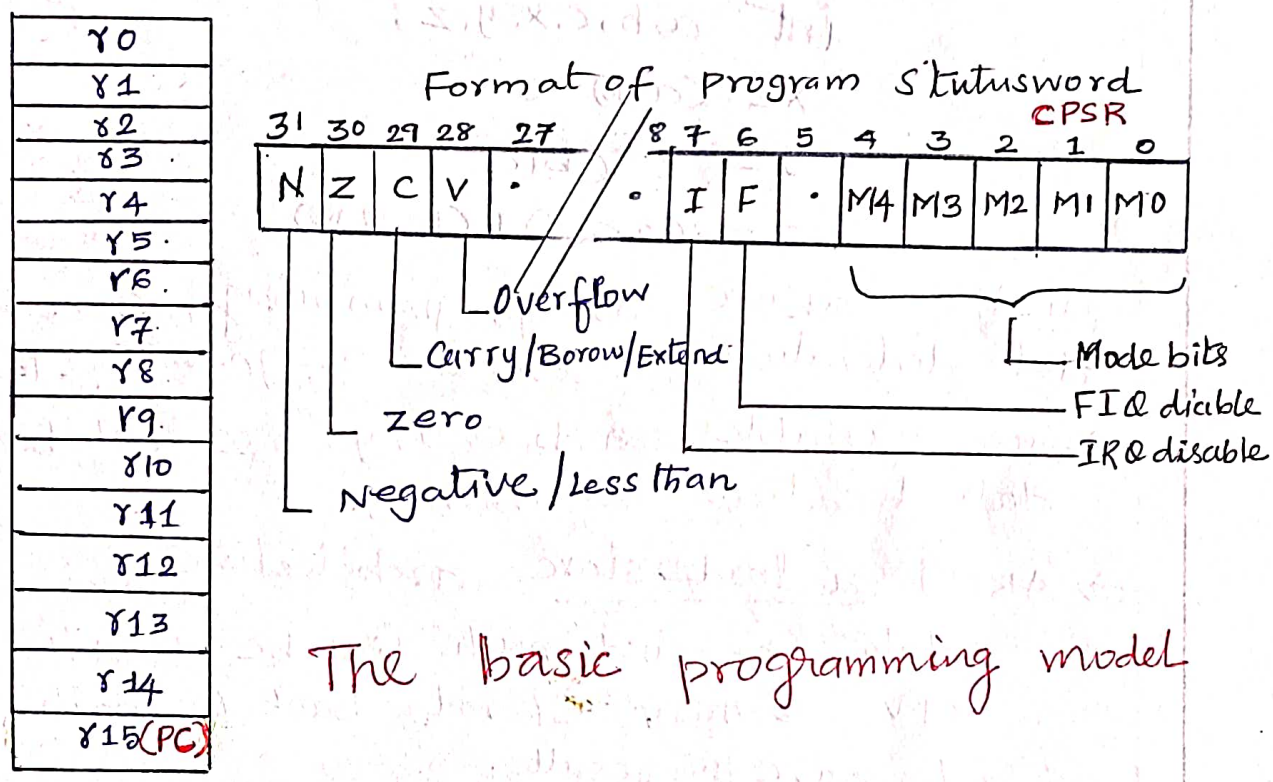
→ The memory controller supports all three access sizes.
→ Always ensure that memory controller supports all three access sizes.

⇨ The negative (N) bit is set when the result is negative in two's complement arithmatic

⇨ The zero (Z) bit is set when every bit of the result is zero.

⇨ The carry (C) bit is set where there is set a carry out of the operation.

⇨ The overflow (V) bit is set when an arithmatic operation result in an overflow.



The basic programming model

⇨ These bits can be used to check easily the result of an arithmatic operation.

⇨ However, if a chain of arithmatic or logical operations is performed & the intermediate state of the CPSR bits are important then they must be checked at each step since the next operation changes the CPSR value.

The basic form of a data instruction is simple

ADD r0, r1, r2 [r0 ← r1 + r2]

ADD r0, r1, #2. [r0 ← r1 + #2]

Data Operations

⇒ Arithmatic and logic operations in C are performed by variable.

⇒ Variables are implemented as memory locations.

⇒ Therefore, to be able to write instructions to perform C expressions & assignments, & consider built arithmatic & logical instructions as well as instructions for reading & writing memory.

```
int a, b, c, x, y, z;
x = (a+b) - c
y = a*(b+c)
z = (a<<2) | (b & 15)
```

in the above 'C' programming, 'C' code with data dederations & several assignment statements.

⇒ The variables a, b, c, x, y & z all become data locations in memory.

⇒ ARM is a load--store architecture

- data operand mush first be loaded into the CPU. & then stored back to main memory to save the results.

⇒ ARM has 16 general-purpose register $r0 \div r15$

⇒ r15 is program counter (PC)

⇒ PC always keep the track of next instruction to be executed.

⇒ The current Program status register (CPSR).

⇒ CPSR is set automatically during every arithmatic, logical, or shifting.

⇒ The top of the register holds arithmatic / logical operations Results.

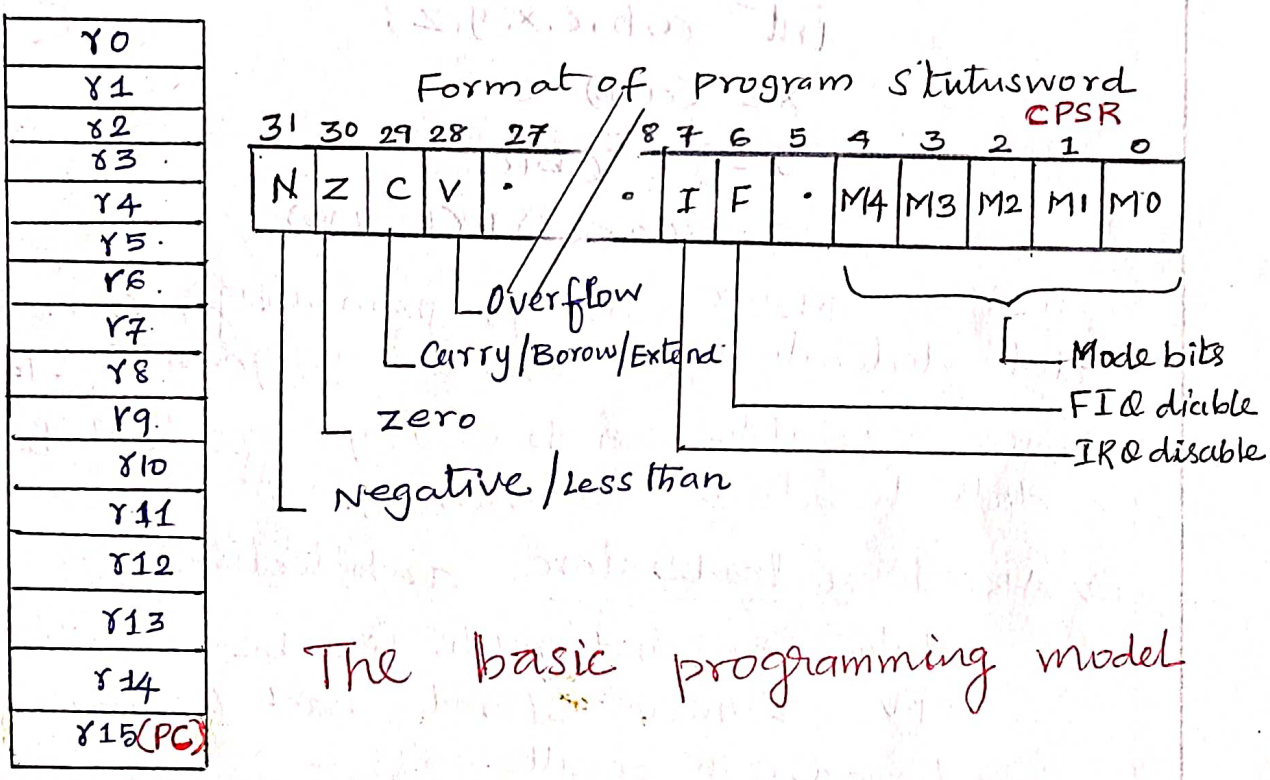
⇒ The result of that arithmatic / logical operation

⇨ The negative (N) bit is set when the result is negative in two's complement arithmatic

⇨ The Zero (Z) bit is set when every bit of the result is zero.

⇨ The carry (C) bit is set where there is set a carry out of the operation.

⇨ The overflow (V) bit is set when an arithmatic operation result in an overflow.



The basic programming model

⇨ These bits can be used to check easily the result of an arithmatic operation.

⇨ However, if a chain of arithmatic or logical operations is performed & the intermediate state of the CPSR bits are important then they must be checked at each step sine the next operation changes the CPSR value.

The basic form of a data instruction is simple

ADD r0, r1, r2 [r0 ← r1 + r2]

ADD r0, r1, #2. [r0 ← r1 + #2]

⇨ RSB performs a subtraction with the order of the two operands reversed

RSB r0, r1, r2 [r0 ← r2 - r1]

⇨ The bit-wise logical operations performs logical AND, OR, and XOR operations

⇨ BIC instruction stands for bit clear.

BIC r0, r1, r2 [r0 ← r1 and not r2] where a bit in the mask is 1 in second operand, the corresponding bit in the first source operand is cleared.

⇨ MUL instruction multiplies two values.

⇨ MLA instructions perform a multiply-accumulate operation, particularly used in matrix operations and signal processing

MLA r0, r1, r2, r3 [r0 ← r1 × r2 + r3]

⇨ Shift operations

LSL – logic shift Left (filling the Least-significant bits with zeros)
LSR – logic shift Right

⇨ The arithmatic shift left is equal to LSL
i.e ASL = LSL

⇨ ASP – it copies the sign bit

⇨ The rotate modifier always rotate right LSB → MSB

RRX modifier performs a 33-bit rotate, with the CPSR's 'C' bit being inserted above the sign bit of the word. This allows the Carry bit to be included in the rotation.

⇨ Comparision operation:

CMP r0, r1 [compute r0 - r1, sets the status bits NZCV & not result of subtraction

CMN uses an addition to set the status bits

⇨ TST performs a bit-wise AND on the operand.
TEQ performs a bit-wise Exclusive-OR operation.

⇨ MOV r0, r1 [r0 ← r1]
MVN instruction complements the operand bits during the move.

Instruction Sets

ADD	Add
ADC	Add with carry
SUB	subtract
SBC	Subtract with carry
RSB	Reverse Subtract
RSC	Reverse Subtract with carry
MUL	Multiply
MLA	Multiply & accumulate

CMP	compare
CMN	Negated compare
TST	Bit-wise test
TEQ	Bit-wise negated test

Logical

AND	Bit-wise AND
ORR	Bit-wise OR
EOR	Bit-wise - Exclusive-OR
BIC	Bit clear

Shift / rotate

LSL	Logical shift left (Zero fill)
LSR	logical shift right (zero fill)
ASL	Arithmatic shift Left
ASR	Arithmatic shift Right
ROR	Rotate Right
RRX	Rotate Right extended with C

MOV	Move
MVN	Move negated

⇨ The values are transferred between registers & memory using load & store instructions.
⇨ LDRB & STRB load and store bytes rather than whole words
⇨ LDRH & SDRH operate on half words
⇨ LDRSH & ~~SDRSH~~ extends the sign bit on loading.
⇨ The ARM uses indirect-addressing in load & store operations

⇨ LDR r0, [r1] : r1 hold the address, the data available in memory location mentioned in r1 is load into r0

STR r0, [r1] : store the contents of r0 in the memory location whose address is given in r1.

LDR r0, [r1 - r2] : r0 ← (r1 - r2)

LDR r0, [r1, #4] ; r0 ← (r1 + 4)

ADR r1, FOO ; r1 ← [0x100] ∵ 0x100 = FOO

Pseudo operation

ARM load-store instruction and Pseudo-Operations.

LDR	load
STR	store
LDRH	load half-word
STRH	store half-word
LDRSH	load half-word signed
LDRB	load byte
STRB	store byte
ADR	set register to address

⇨ The ARM also support seveal form of base-plus-offset addressing.

LDR r0, [r1, #16] ; r0 ← [r1 + 16]

⇨ This addressing mode has two other variations: auto-indexing & post-indexing. Auto-indexing updates the base register, such that

LDR r0, [r1, #16]! (base register ← r1; off set value ← #16)

first adds 16 to the value of r1 & then uses that new value as the address.

! operator causes the base register to be updated with the computed address so that it can be used later.

⇨ Post-indexing does not perform the offset calculation until after the fetch has been performed

LDR r0, [r1], #16 : r0 ← [r1] then [r1] + 16

In this case, the post-indexed mode fetches different value than the other two examples but ends up with the same final value for r1 as does auto-indexing.

Flow control

⇨ The B (branch) instruction is the basic mechanism in ARM for changing the flow of control.

⇨ The branch destination address is called as branch target

⇨ When branch occurs the PC is updated to the branch target.

⇨ The offset is in words, but the ARM is byte addressable, the offset is multiplied by four (shift left two bits) to form a byte address.

Thus, the instruction

B #100 [will add 400 to the current PC value]

⇨ The conditional braches always refers the condition codes as follows.

UNIT II EMBEDDED PROCESSORS & PERIPHERALS

The CPU BUS

⇨ The bus is the machanism by which the CPU communicates with memory and devices.

⇨ A bus is a collection of wires, protocol, memory and devices communicate.

⇨ The bus is to provide an interface to memory and I/O devices.

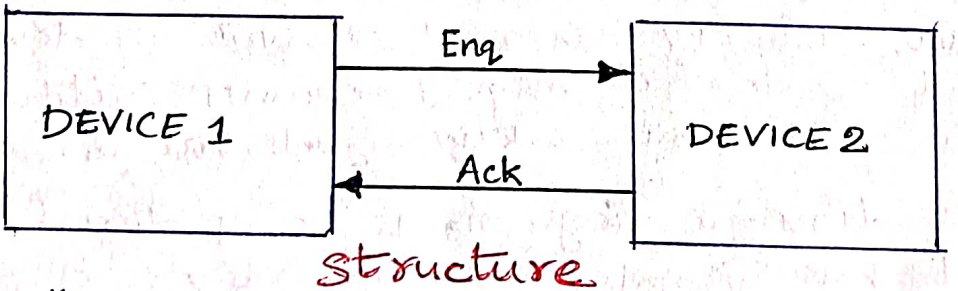
BUS PROTOCOLS

* The basic building block of most bus protocol is the four-cycle handshake
* The handshake ensures that when two devices want to communicate, one is ready to transmit & other is ready to receive.
* The two wires are dedicated to the handshake with enq (enquiry) and ack (acknowledge). Extra wires are used for data transmitting.

The four-cycles are described below:

1). Device 1 raises its output to signal an enquiry, which tells device 2 that it should get ready to listen for data

2) When device 2 is ready to receive, it raises its output to the signal an acknowledgment. At this point devices can transmit & receive



structure

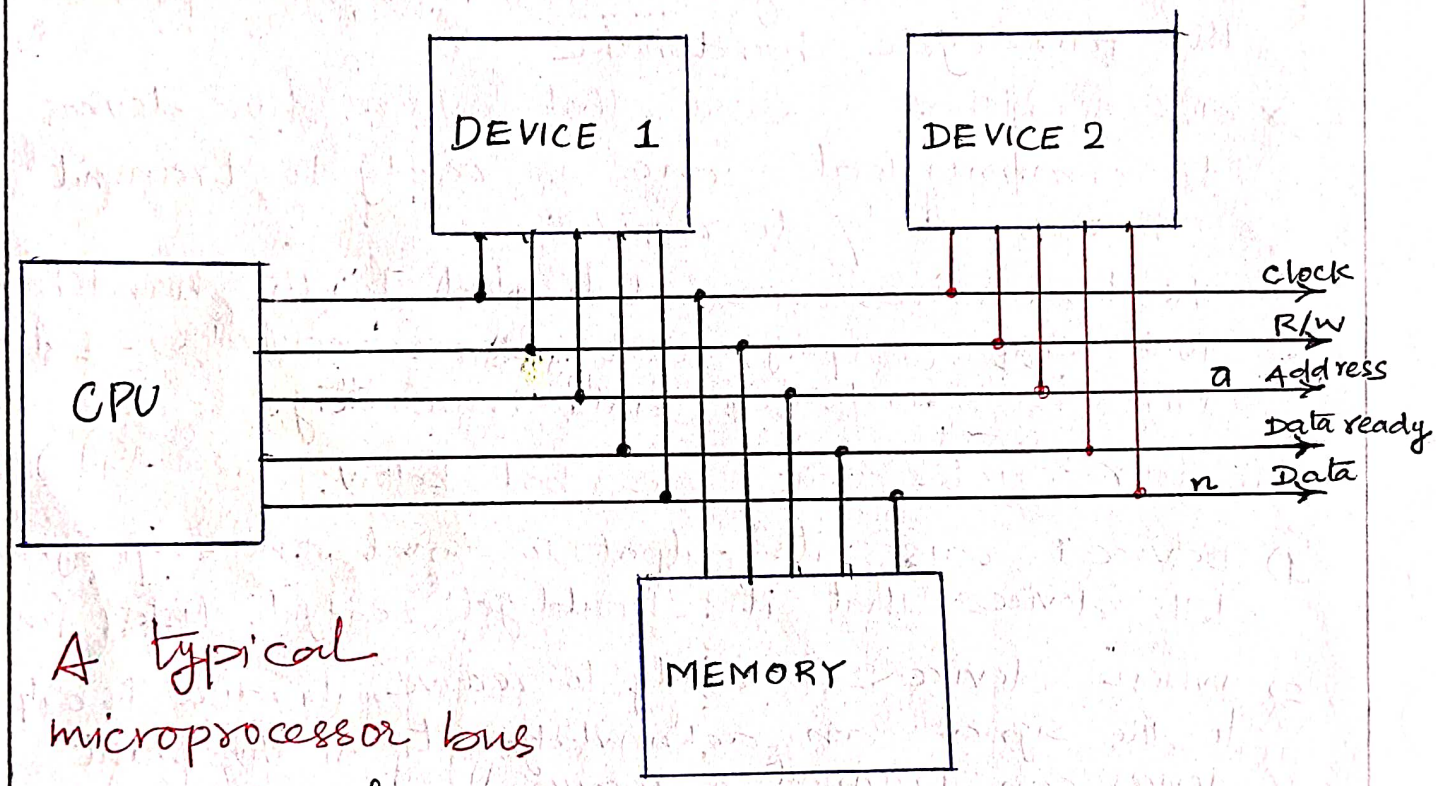
3) Once the data transfer is complete, device 2 lower its output, signaling that it has received the data.

4) After seeing the ack has been released, device 1 lower its o/p.

⇒ At the end of the handshake, both handshaking signals are low, just they were at the start of the handshake then the system is ready for another tranfer.

⇒ The below figure shows the structure of a typical bus that supports read & writes

* clock provides synchronization to the bus components.
* R/W is true when the bus is reading and false when the bus is writing.
* Address is true an a-bit bundle of signals that transmit the address for an access.
* Data is an n-bit bundle of signals that can carry data to or from the CPU &
* Data ready signals when the values on the data bundle are valid.



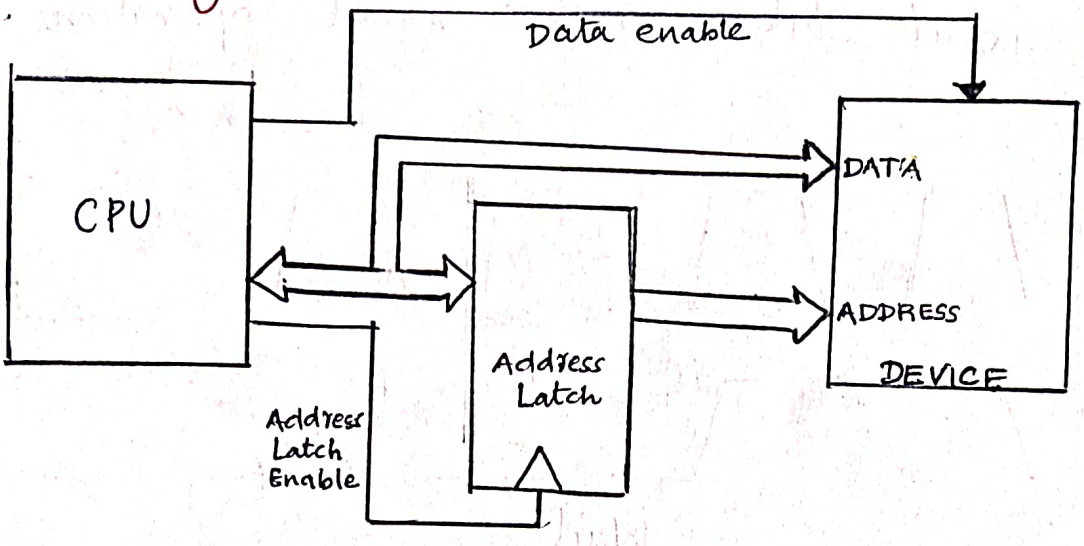
A typical microprocessor bus

⇒ All transfer on this basic bus are controller by the CPU - the CPU can read or write a device or memory but devices or memory cannot initiate a transfer.

⇒ The addres bus & R/W signals are unidirectional.

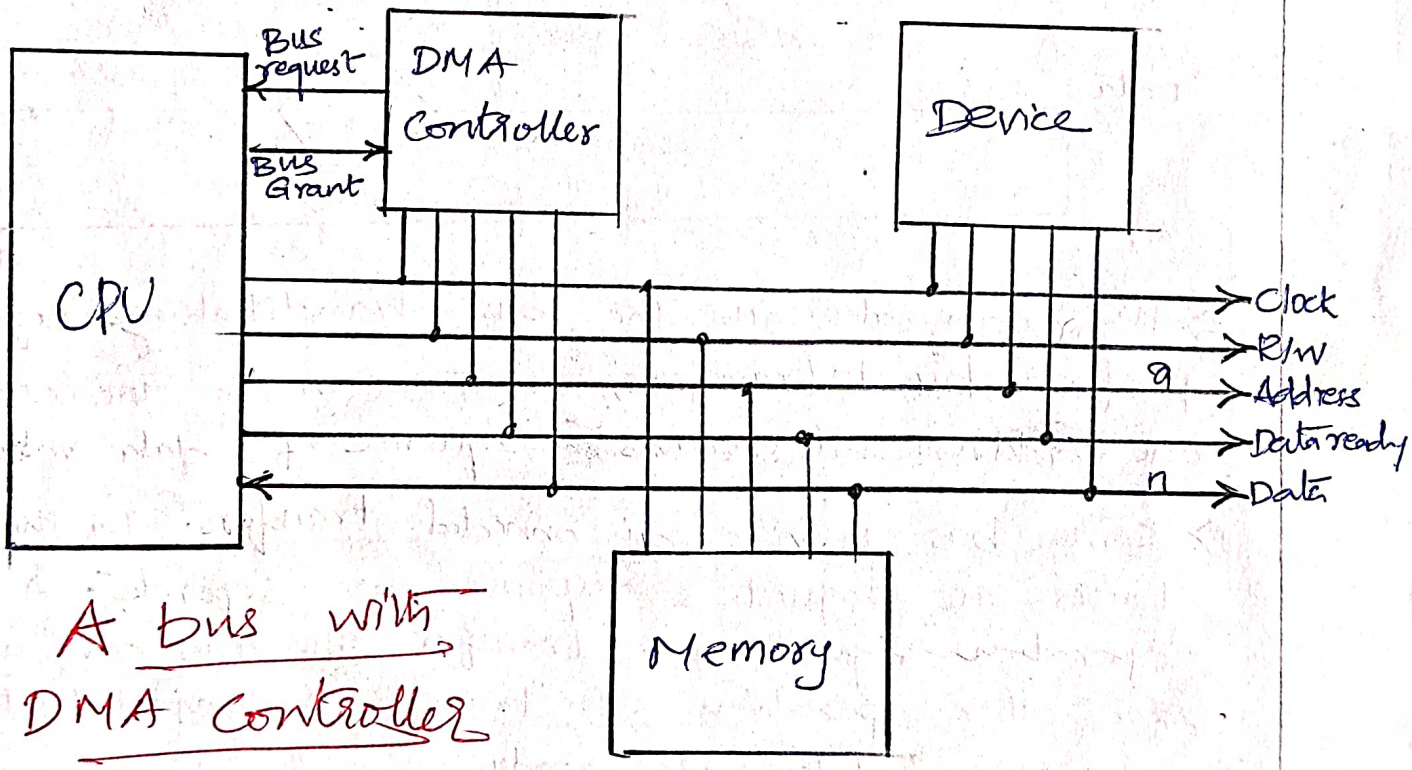
⇒ The timing diagram is a pictorial representation of bus signals, which clearly shows the individual signal state with respect to the clock signal.

Bus signals for multiplexing address & data



DMA : Direct Memory Access

For large data with high-speed transfer between memory and the I/o device without involving the CPU can be performed with the help of DMA controller.



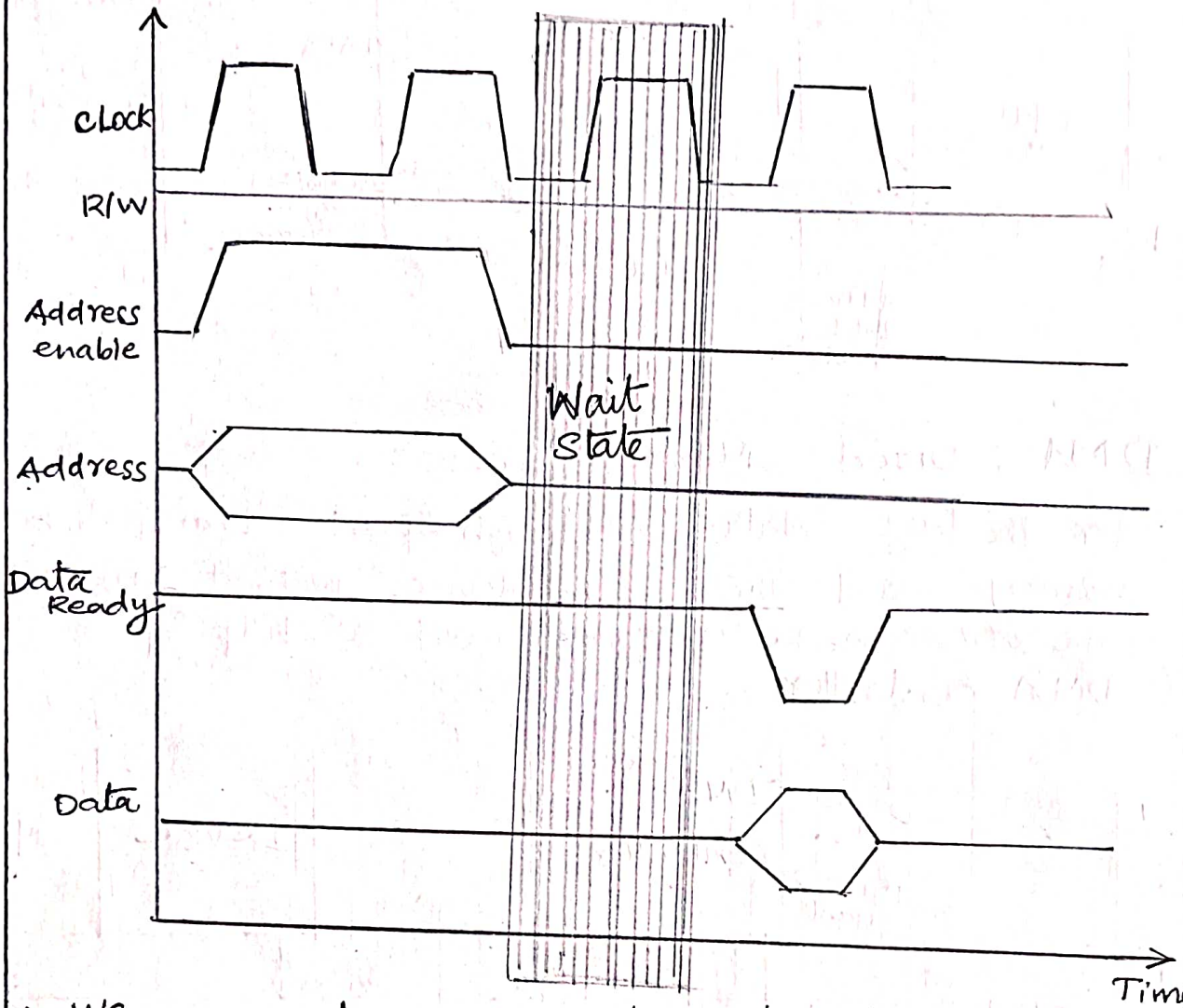
A bus with a DMA controller

⇒ DMA is a bus operation that allows reads & writes not controlled by the CPU but controlled by DMA controller which request the control of the bus from the CPU.

⇒ After gaining the control, the DMA controller performs read & write operation directly between devices & memory.

⇒ Bus request is used by the DMA controller for gaining the bus. Bus grant is used by the CPU to grant bus the DMA Controller

A wait state on the read operation



⇒ We can also use the bus handshaking signals to perform burst transfers, In this mode the CPU sends one address but receives a sequence of data values.

⇒ Some bus provids disconnected transfers. In these buses, the request & response are seperate. A first operation requests the transfer. The bus can then be used for other operations. The transfer is completed later, when the data are ready.

⇒ Some bus use multiplex address & data. Additional control lines are provided to tell whether the value on the address/data lines is an address or data

⇒ Usually, the address comes first followed by the data.

⇒ The address can be held (latch) in a register until the data arrive. so that both can be presented to the device at the same time.

System bus configurations

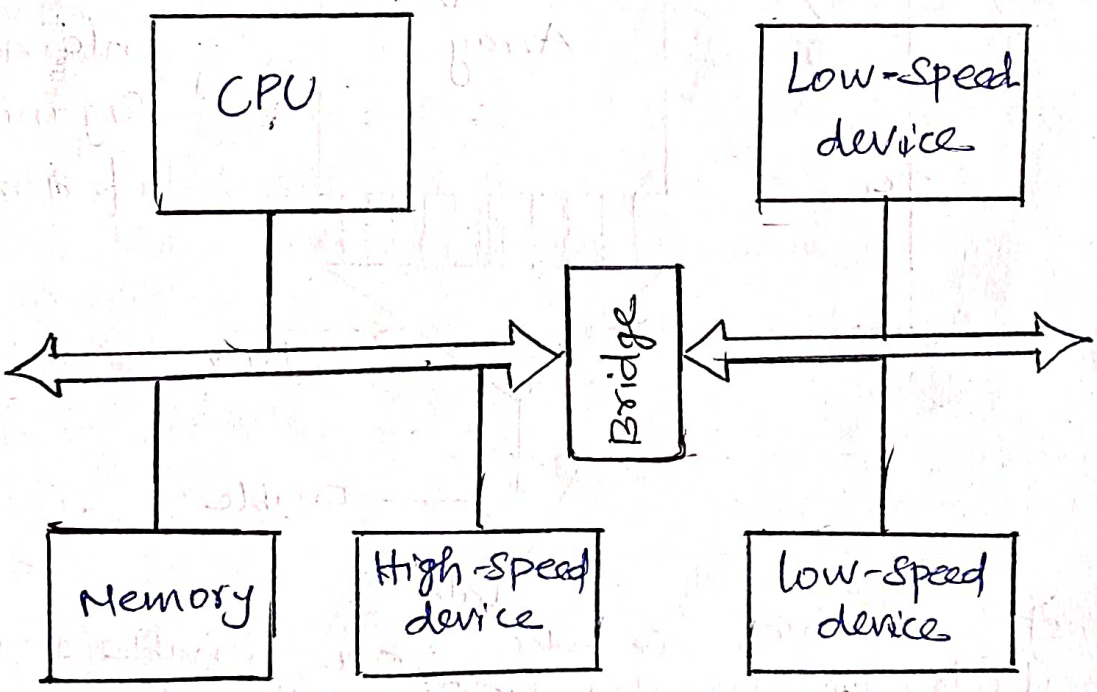
⇒ A microprocessor system often has more than one bus.

⇒ High-speed devices may be connected to a high-performance bus, while lower-speed devices are connected to a different bus.

⇒ A small block of logic known as **bridge** allows the buses to connect to each other

⇒ Advantages of bridges.

* High-speed buses may provide wider data connections
* A high-speed bus requires more expensive circuits & connections.. the cost of low-speed devices can be held down by using a low-speed low-cost bus.
* The bridge may allow the buses to operate independently, thereby providing some parallelism in I/O operations.



A Multiple Bus System

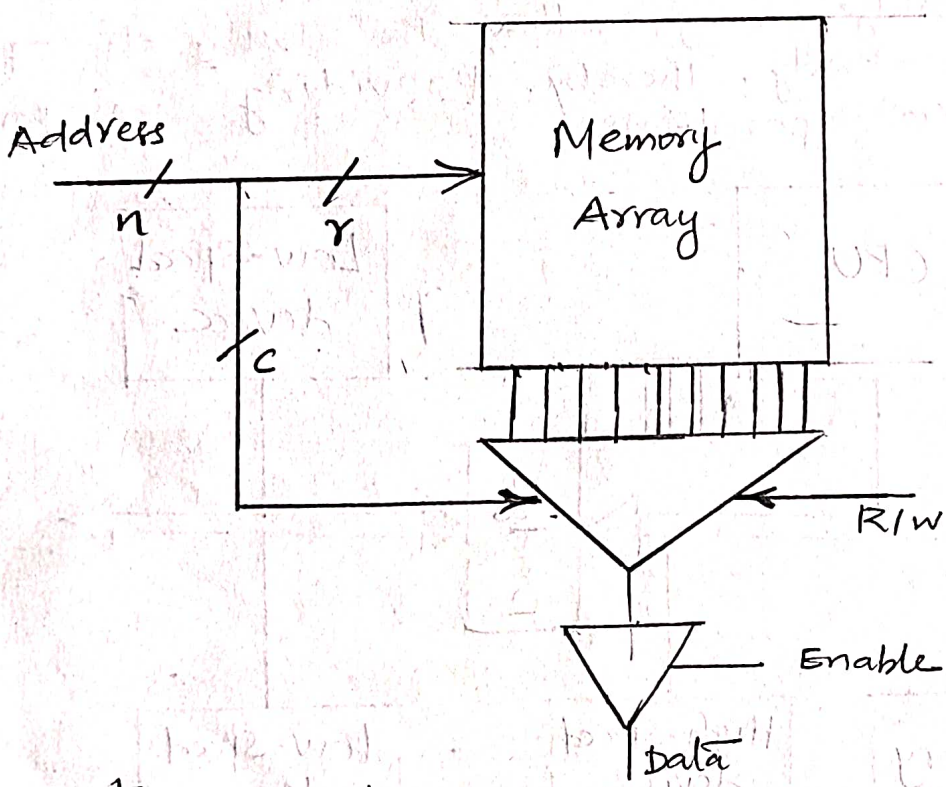
Memory Device Organization

⇒ The memory is characterized by its capacity. Such as 256 MB may be available in two version.

* As a 64M x 4-bit array, a single memory access obtained an 8-bit data item, with a maximum of 2^{16} different addresses.
* As a 32M x 8-bit array, a single memory access obtains a 1-bit data item with a maximum of 2^{13} different address.

⇒ The heigh/width ratio of a memory is known as its aspect ratio.

⇒ Internally, the data is stored in a two-dimentional array of memory cells. then its split as a row & a column address. (with $n = r + c$).



Internal Organization of a memory Device

⇒ Most memories includes an enable signal that controls the tri-stating of data onto the memory's pins.

⇒ A read/write signal (R/W) on read/write memories controls the direction of data transfer

Random-Access Memory (RAM)

⇒ RAM can be read and written.
⇒ The most bulk memory in modern systems is dynamic RAM (DRAM).
⇒ DRAM is very dense : it requires that its value be refreshed periodically since the value inside the memory cells decay over time. [every 10 ms refreshing needs]
⇒ The dominant form of DRAM is Synchronous DRAMs (SDRAM), which uses clocks to improve DRAM performance.

⇒ SDRAM use Row Address Select (RAS) & Column Address Select (CAS) to breaks the address into two parts, which select the proper row & column in the RAM array.
⇒ Signal transitions are relative to the SDRAM clock, which allows the internal SDRAM operations to be pipelined.
⇒ SDRAMs use a seperate refresh signal to control refresh signal to control refreshing.
⇒ Rather than refresh the entire memory at once. DRAMs refresh part of the memory at a time.
⇒ Refreshing slow down the speed of memory.
⇒ SDRAM supports burst modes that allows several sequential addresses to be accessed by sending one address.
⇒ SDRAM supports interleaved mode that exchanges pair of bytes.
⇒ Faster synchronous DRAM is known as Double-Data rate (DDR) SDRAM or DDR2 & DDR3 SDRAMS. are now in use.
⇒ DDRs simply use sophisticated circuit techniques to perform more operations per clock cycle.
⇒ SIMM (single in-line memory modules (or) DIMM (double in-line memory modules) are generally used in PCs
⇒ A SIMM or DIMM a small circuit board that fits into a standard memory socket. Memory chips are soldered to the circuit board to supply the desired memory.

Read-only Memory (ROM)

⇨ ROMs are preprogrammed with fixed data. They are very useful in embedded system since a great deal of the code, and some data, does not change over time.

⇨ ROMs are also less sensitive to radiation-induced errors.

⇨ The basic types are factory-programmed ROM (or Mask-programmed ROM) & Field-programmable ROM.

⇨ Factory-programmed ROM are ordered from the factory with particular programming.

⇨ Field-programmable ROMs can be programmed in the lab.

⇨ Flash memory is the dominant form of field programmable ROM & is electrically erasable.

⇨ Flash memory use standard system voltage for erasing & programming, allowing it to be reprogrammed inside a typical system.

⇨ It supports automatic distribution of upgrades.

⇨ The modern Flash memory allows memory to be erased in blocks.

⇨ A common application is to keep the boot-up code in a protected block but allow updates to other memory blocks on the device.

⇨ As the result, this form of flash is commonly known as boot-block flash.

I/O DEVICES

⇨ Some of the I/O devices are used as on-chip devices on micro-controllers; others are implemented seperately.

⇨ common I/o devices used in embedded computing system

* Timers and Counters
* Watchdog Timer
* A/D and D/A Converters
* Keyboard
* LEDs
* Displays
* Touchscreens

Timers and Counters

⇒ Both are built from adder logic with registers to hold the current value, with an increment input that adds one to the current register value.

⇒ A timer has its count connected to a periodic clock signal to measure time intervals.

⇒ A counter has its count input connected to an aperiodic signal in order to count the number of occurances of some external event.

⇒ Because the same logic can be used for either purpose, the device is called a counter/timer.

A watchdog timer

⇒ It is an I/O device that is used for internal purpose of a system.

⇒ The watchdog timer is connected into the CPU bus & also to the CPU's reset line.

⇒ The CPU's software is designed to periodically reset the watchdog timer, before the timer (still) ever reaches its time-out limit.

⇒ If the watchdog timer still does reach that limit, its time-out action is to reset the processor.

⇒ Otherwise CPU will hang-up, rather than diagnose the problem, the system is reset to get it operational as quickly as possible.

A/D & D/A converter

⇒ These devices are used to interface non-digital devices to embedded systems.

⇒ A/D conversion requires sampling the analog input before converting it to digital form.

⇒ A control signal causes the A/D converter to take a sample & digitize it.

⇒ There are two different types of A/D converter circuit
1) Constant-time converter &
2) Variable-time converter.

⇒ A typical A/D interface has analog input and two digital inputs. A data port allows A/D registers to be read and written, & a clock input tells when to start the next convertion.

⇒ D/A conversion is relatively simple. It's includes only the data value. The input is continuously converted to analog form.

Keyboard

➡ A keyboard is an array of switches, it includes some internal logic to interface with microprocessors.

➡ The major problem with mechanical switches is bouncing.

➡ The bouncing problem is rectified by hardware method and software method.

➡ A hardware debouncing circuit can be built using a one-shot timer & the software can also be used debounce the switch input.

➡ The more key switches are orgonished using encoded keyboard.

➡ An encoded keyboard use some code to represent which switch is currently being depressed.

➡ The important part of the encoded key is scanned array of switches

➡ The scanned key board array reads only one row of switches at a time.

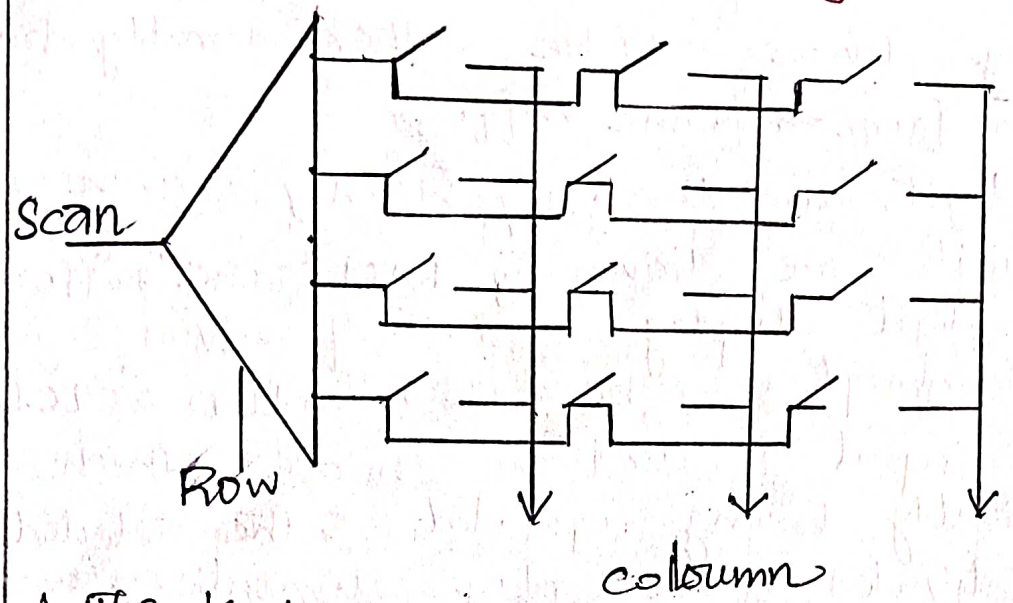
➡ The demultiplexer at the left side of the array selects the row to be read.

➡ When the key input is 1, that value is transmitted to one terminal of each key in the row

➡ If the switch in the column is depressed, the 1 is sensed at that column. Since only one switch in the column is activated, that value uniquely identifies a key.

➡ The row address & column output can be used for encoding, or circuit can be to give a different encoding.

A scanned key array

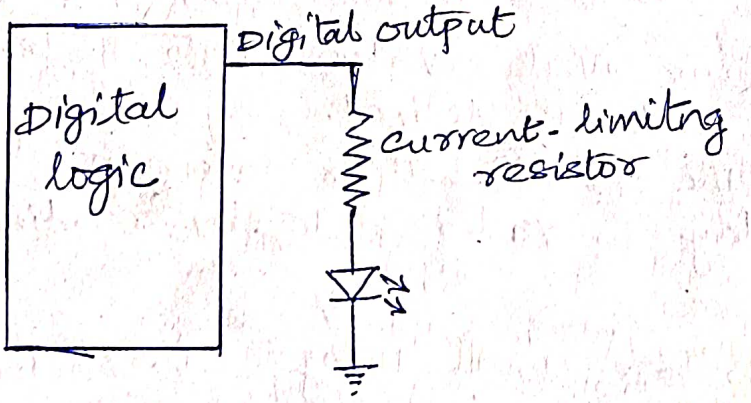


⇒ The keyboard microcontroller can be programmed to provide n-key rollover, so that rollover keys are sensed, put on a stack and transmitted in sequence as keys are released.

LEDs – (Light-Emitting Diodes)

⇒ It is a simple output device, a resistor is connected between the output pin & the LED to observe the voltage difference between the digital output voltage & the 0.7V drop across the LED.

⇒ When the digital output goes to 0, the LED voltage is in the device's off region and the LED is not on.



An LED connected to a digital output

Displays

⇨ A display device may be either directly driven or driven from a frame buffer.

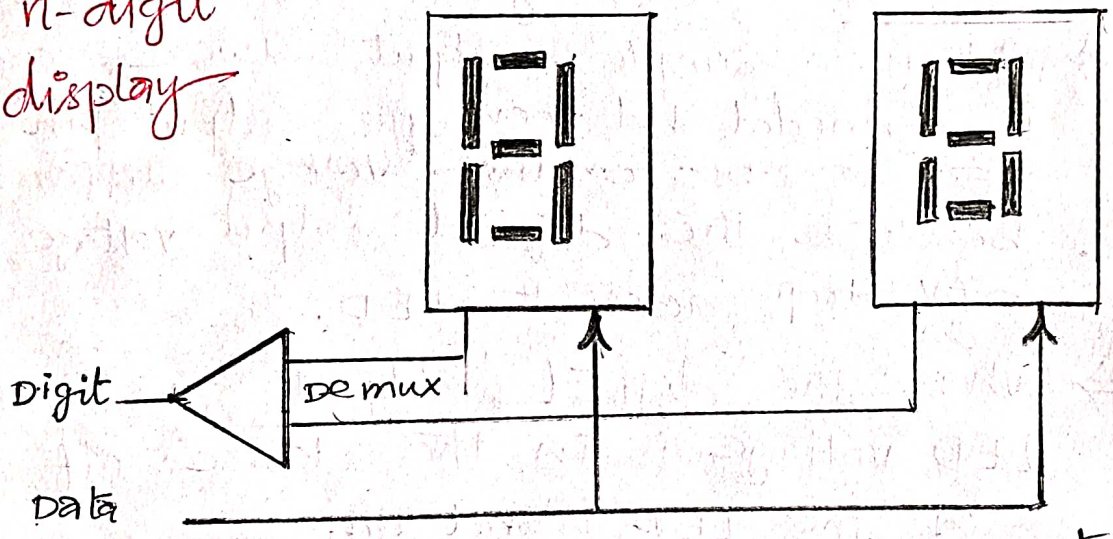
⇨ Small digits are driven by directly driven & large digits are driven by RAM frame buffer.

⇨ A single-digit display consists of seven segments; each segment may be either an LED or LCD element.

⇨ The digit input is used to choose which digit is currently being updated & the selected digit activates its display elements based on the current data value.

⇨ The display's driver is responsible for repeatedly scanning through the digits and presenting the current value of each to the display.

An n-digit display



⇨ A frame buffer is a RAM that is attached to the system bus.

⇨ The microprocessor writes values into the frame buffer in whatever order is desired.

⇨ The pixels in the frame buffer are generally written to the display in raster order by reading pixels sequentially.

⇨ Large displays are built using LCD. Each pixel in the display is formed by a single liquid crystal.

⇒ LCD displays present a very different interface to the system because the array of pixel LCDs can be randomly accessed.

⇒ Early LCD panels were called passive matrix because they relied on a 2-Dimentional grid of wires to address the pixels.

⇒ Modern LCD panels use an active matrix system that puts a transistor at each pixel to control access to the LCD.

⇒ Active matrix displays provide higher constrast and a higher-quality display.

Touchscreens

⇒ A touchscreen is an input device overlaid on an output device.

⇒ The touchscreen registers the position of a touch to its surface.

⇒ By overlaying this on a display, the user can react to information shown on the display.

⇒ The two most common types of touchscreens are resistive & capacitive.

⇒ A resistive touchscreen uses a 2-Dimentional volt-meter to sense position.

⇒ The touchscreen consists of two conductive sheets seperated by spacer bar

⇒ The top conductive sheet is flexible so that it can be pressed touch the bottom sheet

⇒ A voltage is applied across the sheet; its resistance causes a voltage gradient to apper across the sheet

⇒ The top sheet samples the conductive sheet's applied voltage at the contact position.

⇒ The analog & digital converter is used to measure the voltage & resulting position

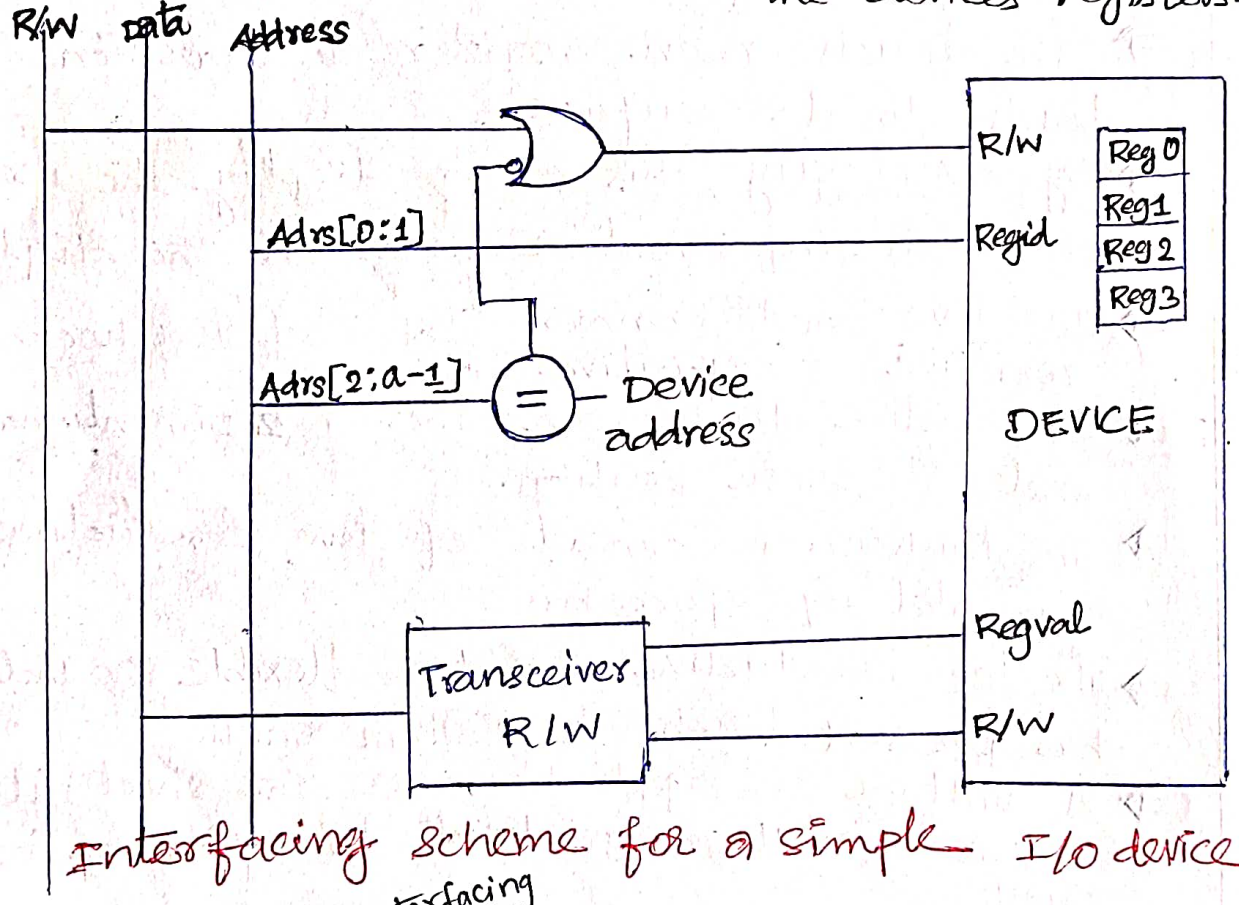
⇒ The touchscreen alternates between x and y position sensing by alternately applying horizontal & vertical voltage gradients.

Component Interfacing

⇨ The component interfacing is including memory & I/O devices with bus system.

⇨ Device Interfacing

- Some I/O devices are designed to interface directly to a particular bus, forming glueless interfacing.
- The glue logic is required when a device is connected to a bus for which it is not designed.
- An I/O device typically requires a much smaller range of addresses than a memory, so addresses must be decoded much more finely.
- Some additional logic is required to cause the bus to read and write the device's registers.



Interfacing scheme for a simple I/O device interfacing

⇨ I/O devices may be obtained in two ways

1) Simple I/O
2) Memory mapped I/O

⇒ The device has four register that can be read & written by presenting the register number on the regid pins. Also asserting R/W as required and reading or writing the value on the reg val pins.

⇒ To interface to the bus, the bottom two bits of the address are used to refer the registers within the device & the remaining bits are used to identify the device itself.

⇒ When the bus address can be set with switches & the bus address matches the device's.

⇒ The device's address can be set with switches to allow the address to be easily changed.

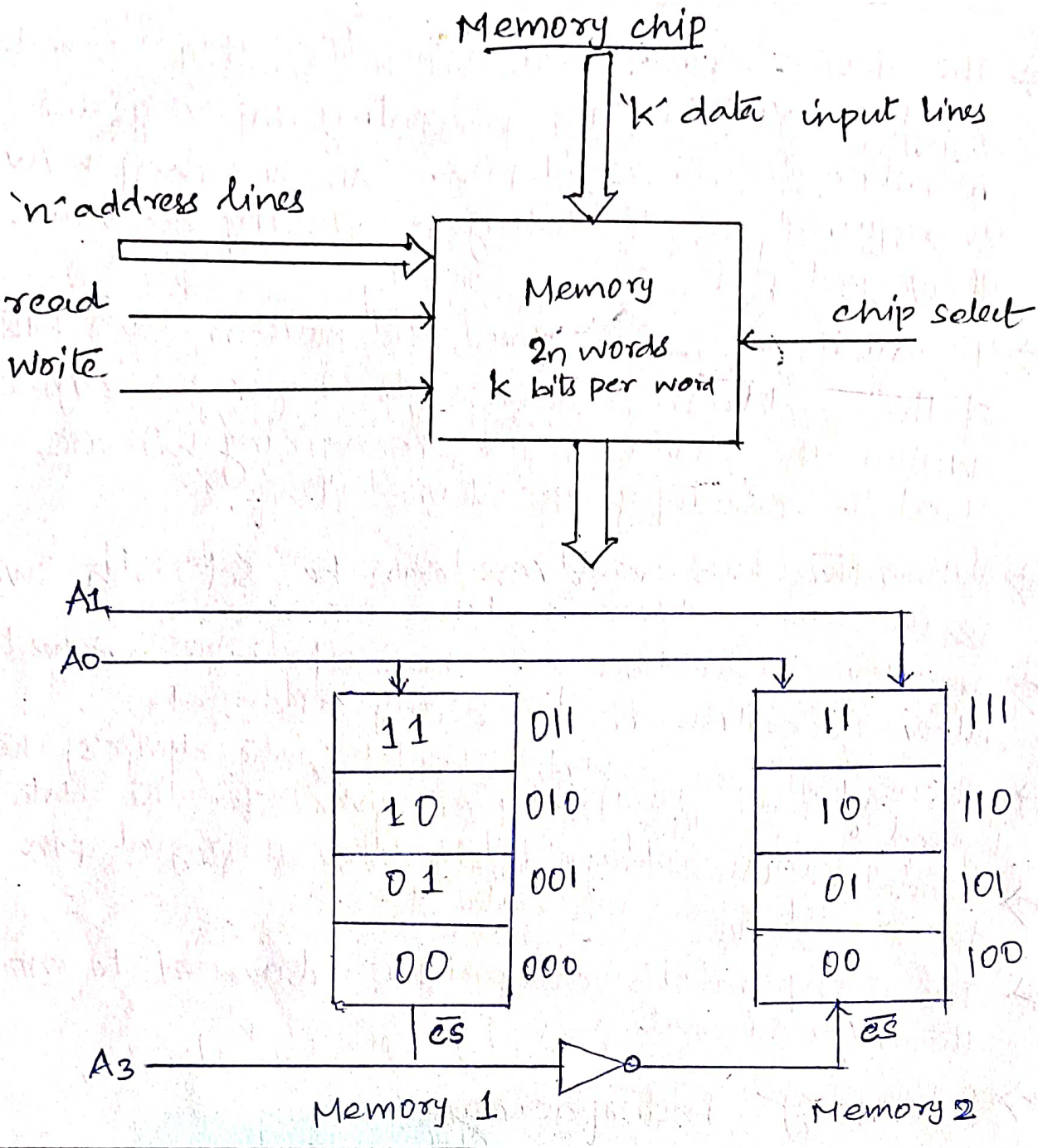
⇒ when the bus address matches the device's, the result is used to enable a transceiver for the data pins.

⇒ when the transceiver is disabled, the reg val pin are disconnected from the data bus.

⇒ The comparator's output is also used to modify the R/W signal.

⇒ Memory Interfacing

* If the memory size is exact then the interfacing is very simple.
* If we buy other than exact required size then we have to construct the memory to fit with the system.
* In case of multiple chips simple circuit like NOT gate will not work
* In this case normally decoder circuits take 3-to-8 decoder. This circuit is called address decoder. There are two types
  1) partially decoding
  2) fully decoding.



Designing with microprocessor

→ Architectures & Components
- Software
- Hardware

→ Some software is very hardware dependent.

→ Contains several elements in hardware platform. they are CPU
Bus
Memory
I/O devices : networking, sensors, actuators ect.

Software architecture

- Functional description must be broken into pieces
  - * division among people
  - * conceptual organization
  - * performance
  - * testability
  - * maintenance

Hardware and software architecture

- Hardware and software are intimately related.
- Software does not run without hardward.
- how much hardware you need is determined by the software requirements:
  - * speed
  - * memory

Evaluation boards

- Designed by CPU manufacturer or others.
- It includes CPU, memory, some I/O devices,
- It may include prototyping section.
- CPU manufacturer often gives out evaluation board netlist
- It can be used as starting point for your custom board design.

Adding logic to a board

→ Programmable logic devices (CPLDs)
- provide low/medium density logic.

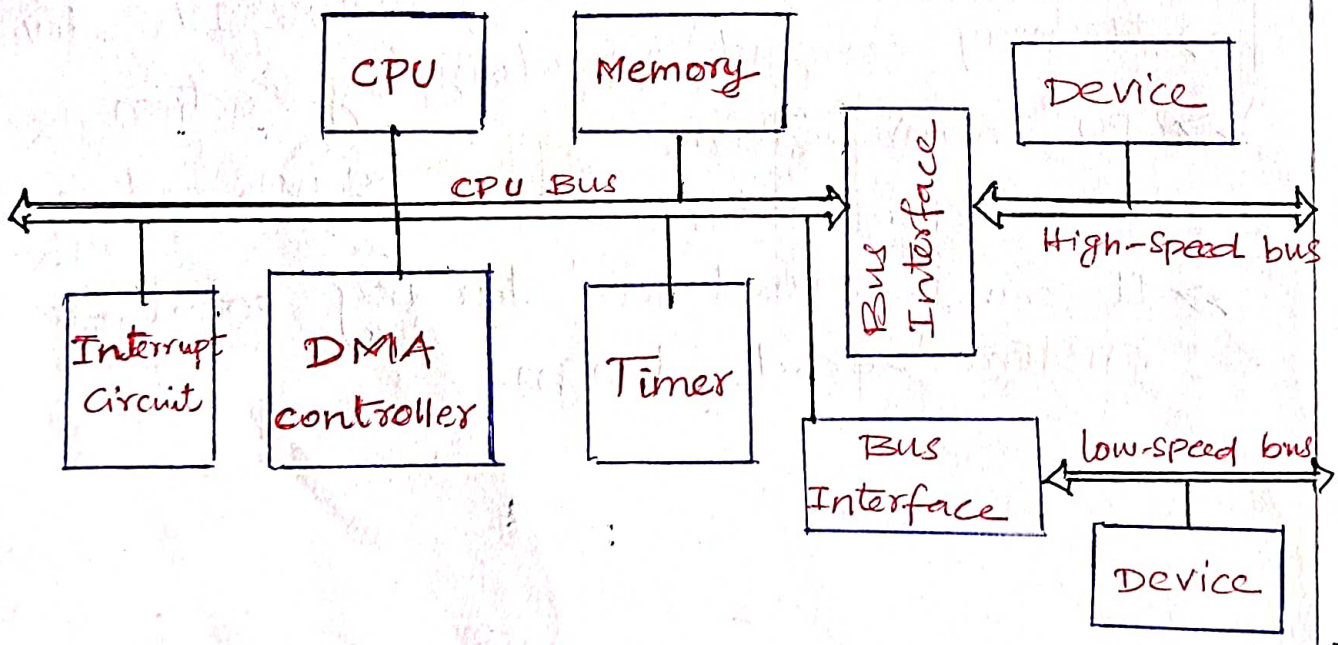
→ Field-programmable gate arrays (FPGAs)
- Provide more logic & multi-level logic.

→ Application-specific integrated circuits (ASICs)
- ASICs are manufactured for a single purpose.

→ If the PC is used as a platform, it takes the advantages
- cheap and easy to get
- rich and familiar software environment

and it takes the disadvantages
- requires a lot of hardware resources
- not well-adapted to real-time.

Typical PC hardware platform



Typical busses

⇒ PCI : standard for high-speed interfacing
- 33 or 66 MHz
- PCI Express

⇒ USB (Universal Serial Bus)
- Firewire
- relatively low-cost serial interface with high speed.

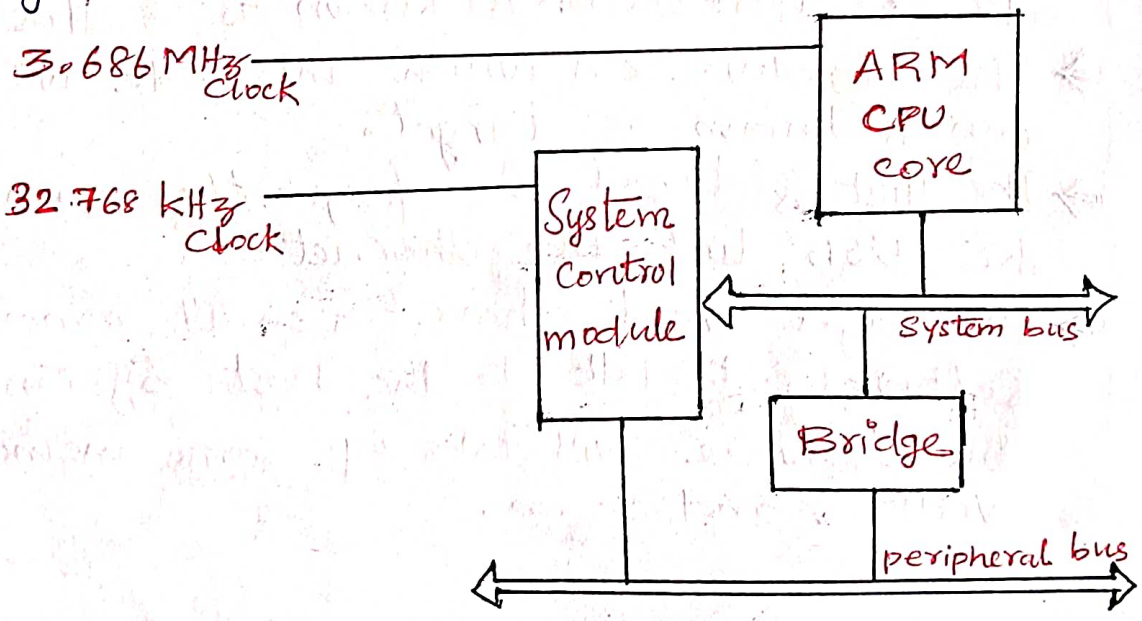
Software elements

⇒ IBM PC uses BIOS (Basic Input Output System) to implement low-level functions :
- Boot-up process
- Minimal device drivers

⇒ BIOS has become a generic term for the lowest-level system software.

Example : Strong ARM

⇒ StrongARM system includes
* CPU chip (3.686 MHz clock)
* system control module (32.768 kHz clock)



➔ The Strong ARM provides a number of functions besides the ARM CPU.

➔ The chip contains two on-chip buses:
   1) a high-speed system bus
   2) a lower-Speed peripheral bus.

➔ The system control module contains the following peripheral devices.
   * A real-time clock
   * An operating system timer
   * 28 general-purpose I/Os (GPIOs)
   * An interrupt controller
   * A power manager controller
   * A reset controller that handles resetting the processor

➔ The 32.768 KHz clock's frequency is choosen to be useful in timing real-time events.

➔ The lower clock rate and therefore lower power consumption.

DEVELOPMENT AND DEBUGGING

Development Environments

➔ A part of the software development on a PC or workstation is known as a **Host**.

➔ The hardware on which the code will finally run is known as **target**.

➔ The Host & target are frequently connected by the USB link like Ethernet.

➔ The target must have a small amount of software to talk to the host system.
The software will take up some memory, interrupt vectors and so on.

➡ The host should be able to do the following.
* load programs into the target,
* Start and Stop program execution on the target
* examine memory & CPU register.

A Cross-Compiler : It is a compiler that runs on one type of machine but generates code for another.

- After compilation, the executable code is downloaded to the embedded system by a serial link or burned in a PROM & plugged in.

➡ In many causes, a test bench program can be built to help debug in embedded code.

Debugging Techniques

challegs :
- target system may be hard to observe;
- target may be hard to control;
- may be hard to generate realistic inputs;
- Setup sequence maybe complex.

☆ The serial port found on most evalution bords is one of the most important hardware.

☆ Breakpoint is an another important debugging tool.

⇨ A break point allows the user to stop execution examine system state and change state.

⇨ Replace the breakpointed instruction with a subroutine call to the monitor program.

ARM breakpoints

0X400 MUL r4, r6, r6	0X400 MUL r4, r6, r6
0X404 ADD r2, r2, r4	0X404 ADD r2, r2, r4
0X408 ADD r0, r0, #1	0X408 ADD r0, r0, #1
0X40C B loop	0X40C BL bkpoint
uninstrumented code	code with breakpoint

⇨ Breakpoint handler actions

- save registers.
- Allow user to examine machine.
- Before running, restore system state.
  + safest way to execute the instruction is to replace it and execute in place.
  + Run another breakpoint after the replaced breakpoint to allow restoring the original breakpoint.

⇨ The microprocessor **in-circuit emulator** (ICE) is a specialized hardware tool that can help debug software in a working embedded system.

⇨ It allows you to stop execution, examine CPU state & modify the register.

⇨ The **logic analyzer** is an array of inexpensive oscilloscopes

⇨ The analyzer can sample many different signals simultaneously.

→ The logic analyzer records the values and then displays the result.

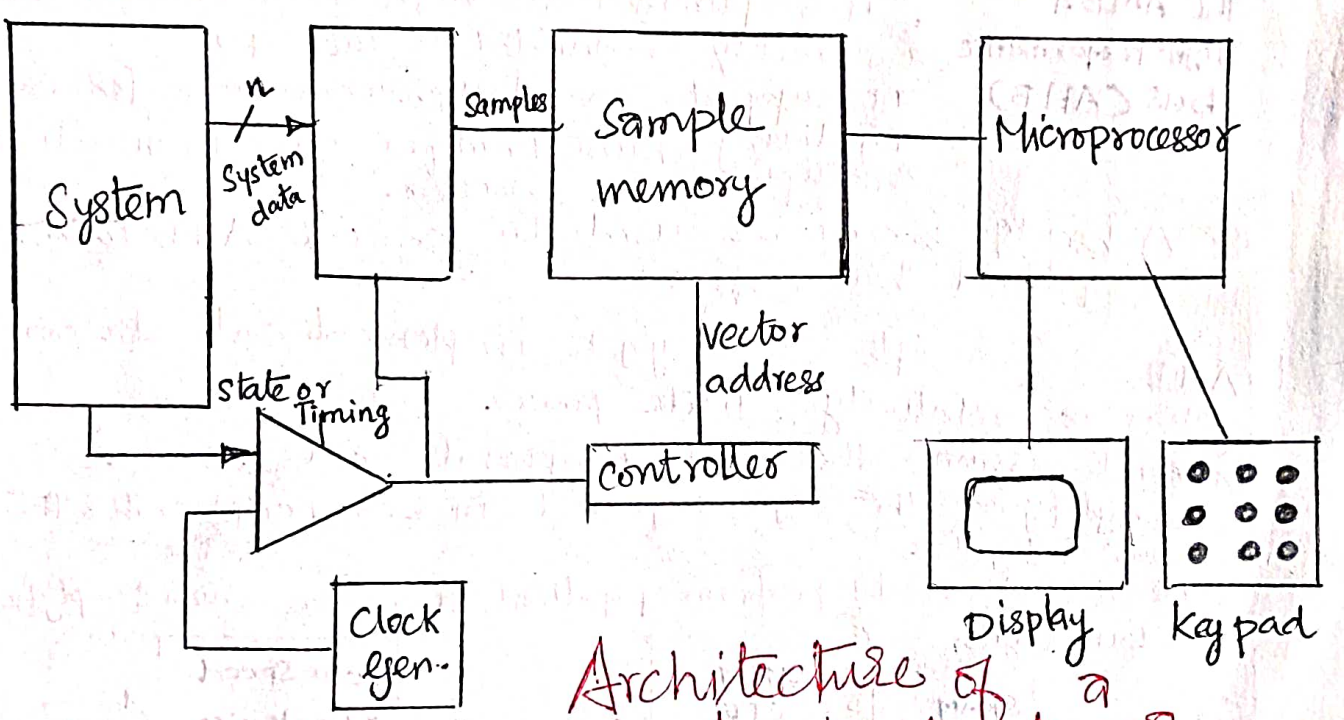
→ It can capture thousand of or even millions of samples of data.

→ It can acquire data in either of two modes that are called State & timing modes.

→ State mode uses the system's own clock to control sampling, so it samples each signal only once per clock cycle.

– Timing mode provides greater resolution in the signal for detecting glitches.

– state mode - used for sequentially oriented problems
– Timing mode - used for glitch-oriented debugging.



Architecture of a Logic Analyzer

⇒ logic analyzer gives a very good view of the externally visible signals. That information can be used for both functional and timing debugging.

Debugging challenges

→ logical errors in software can be hard to track down, but errors in real-time code can create problems that are even harder to diagnose

→ Bugs in drivers can cause non-deterministic behaviours in the fore-ground problem

→ Bugs may be timing-dependent.

ARM BUS

⇨ ARM has developed a seperate bus specification for single-chip systems

⇨ The AMBA bus supports CPUs, memories, & peripherals integrated in a system-on-silicon.

⇨ AMBA specification includes two buses.

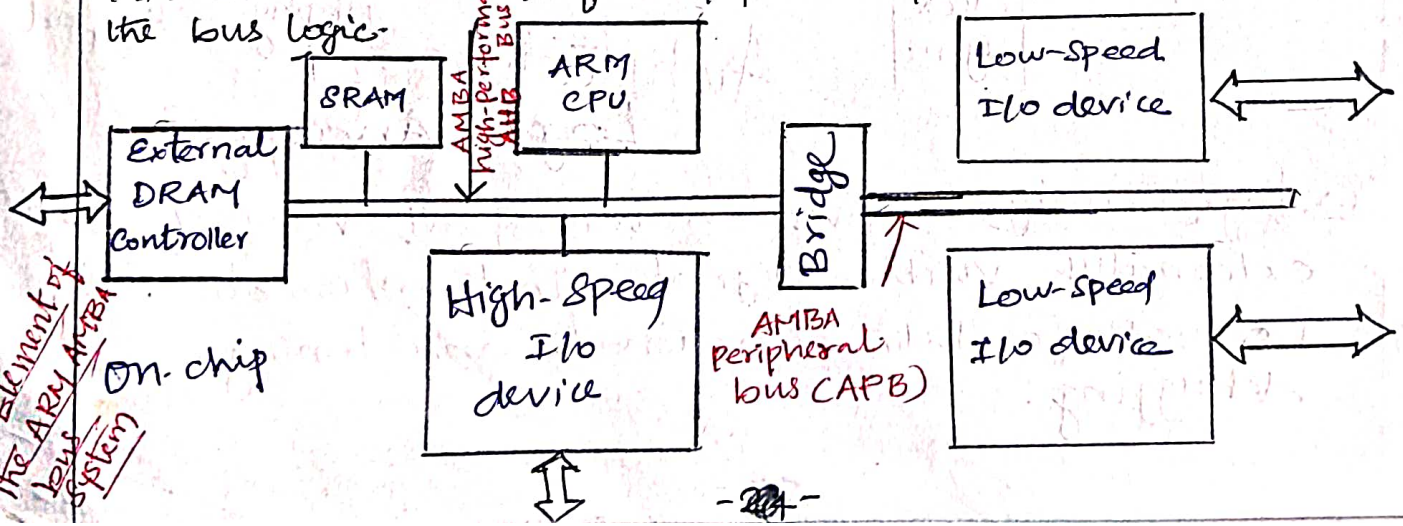
The AMBA High-Performance bus (AHB) ; It is optimized for high-speed transfer & it is directly connected to the CPU. It supports several High-performance features: pipelining, burst transfer, split transactions & multiple bus master.

⇨ A bridge is can be used to connect AHB to an AMBA peripheral bus (APB)

AHB Bus ; Simple & easy to implement : it also consumes relatively little power.

⇨ AHB assumes that all peripherals act as slave, simplifying the logic required in both peripherals & the bus controller.

⇨ It also does not perform pipelined operations, which simplyfies the bus logic.



* leading byte determines the outgoing port.
* MSB of each header byte distinguishes between host & switch packets.
* Variable payload length & 8-bit CRC.
* Switches; use blocking-cut-through routing
  Each contains two custom VLSI chips
  - Crossbar-switch
  - Dual interface
* Topology is arbitrary.
* Myrinet is flexible & fast.
* It also affordable systems with acceptable performance
* The data payload may be of variable length & the packet includes error correction codes.

Internets

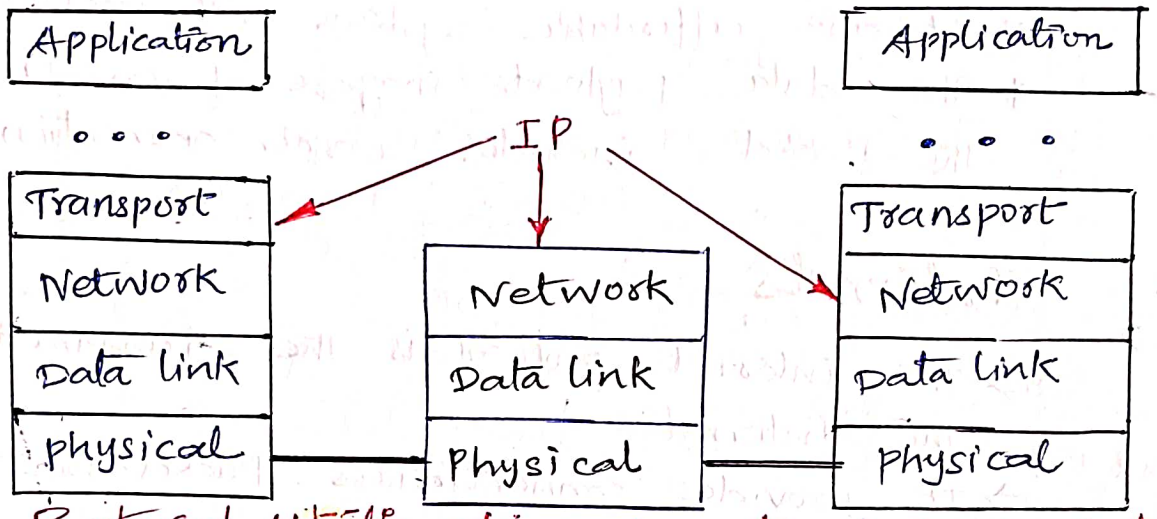
⇒ The internet protocol is the fundamental protocol on the Internet.
⇒ It provides connectionless, packet-based communication
⇒ Industrial automation uses a good application area for Internet-based embedded systems.
⇒ Internet protocol is very important in embedded computing.
⇒ The internet packet will travel over several different networks from source to destination.
⇒ The IP allows data to flow smoothly through these networks from one user to another.
⇒ IP works at network layer.
⇒ IP & individual networks are illustrated in figure present in next page
⇒ When node A wants to send data to node B, the application's data pass through several layers of the protocol stack to get into the Internet Protocol.

⇨ IP creates packets for routing to the destination, which are then send to the data link & physical layers.

⇨ A node that transmits data among different types of networks is known as a router.

⇨ In general, a packet may go to through sevel routers to get into its destination.

⇨ At the destination, the IP layer provides data to the transport layer & then application (layer).



Protocol utilization in Internet communication.

⇨ As the data pass through several layers of the protocol stack, the IP packet data are encapsulated in packet formats appropriate to each layer.

⇨ The basic format of an IP Packet is shown figure available in next page.

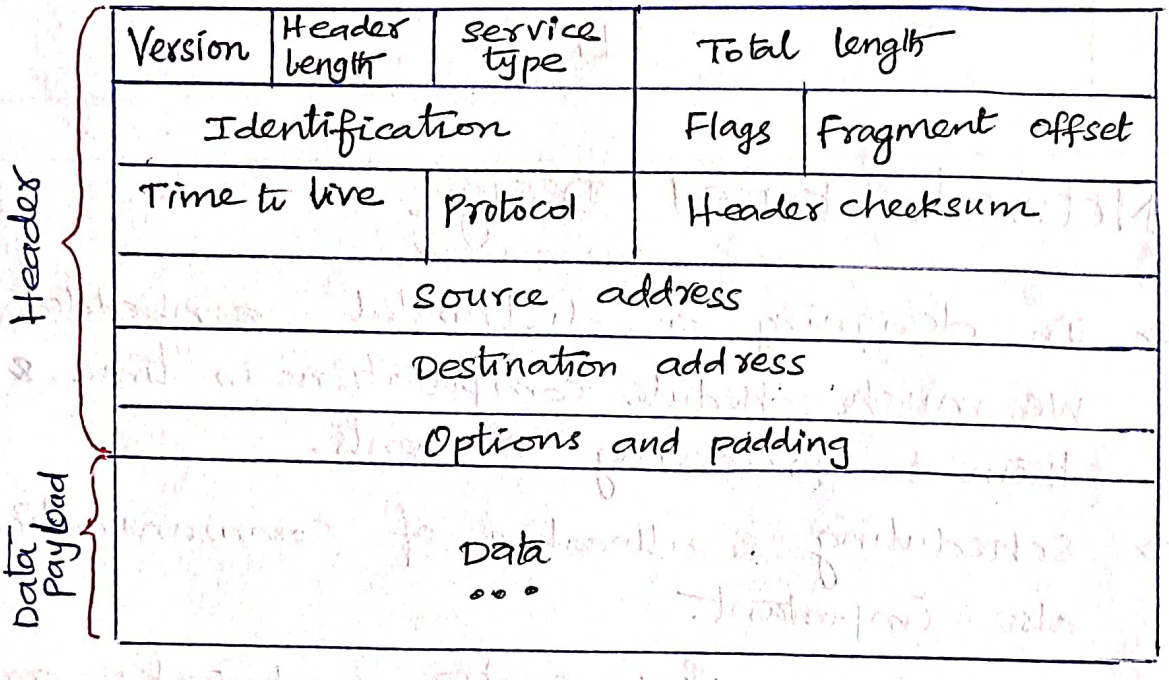
⇨ The header & data payload are both of variable length. The maximum total length of the header & data payload is 65,535 bytes.

⇨ The IP address is typically written in the form of xxx.xx.xx.xx The names by which users and applications typically referred to Internet nodes. such as foo.baz.com, are translated into IP addresses via calls to a Domain Name Server (DNS), one of the higher-level services built on top of IP.

⇒ IP works on network layer. so, it does not guarantee that a packet is delivered to its destination. Also packets may arrive out of order. This is referred to as best-effort routing.

⇒ Packets being routed along very different paths with different delays, real-time performance of IP can be hard to predict.

⇒ IP packet structure



⇒ The internet also provides higher-level services built on top of IP. Transmission control Protocol (TCP) is one such example.

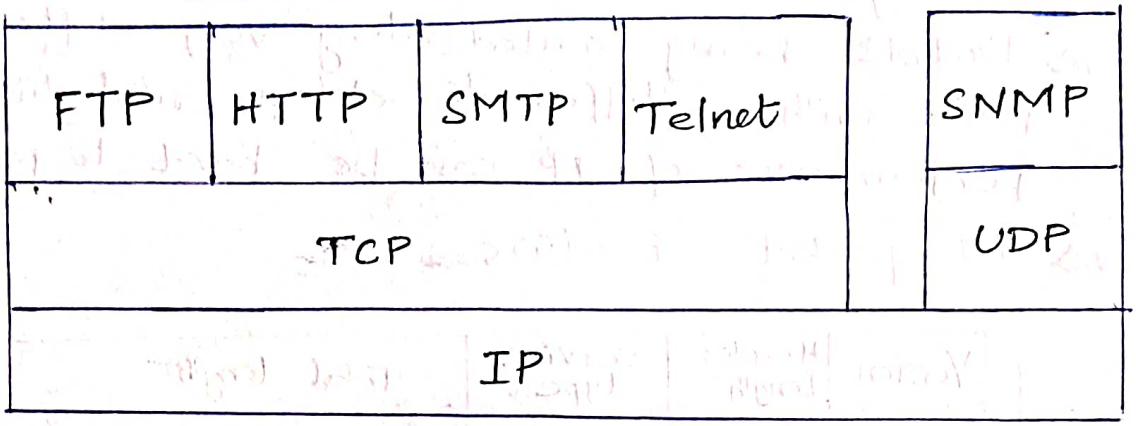
⇒ It provides connection-oriented service that ensure that data arrive in appropriate order.

⇒ The diagram in next page shows the relationship between IP & high-level Internet services.

⇒ Using IP as the foundation, TCP is used to provide File Transfer Protocol (FTP) for batch file transfers, Hypertext Text Transport protocol (HTTP) for World Wide Web services (WWW) Simple Mail Transfer Protocol (SMTP) for email. and Telnet for virtual terminals.

⇒ A separate transport protocol, User Datagram Protocol (UDP), is used as the basic for the network management services provided by the Simple Network Management Protocol (SNMP).

The Internet service stack



Network-Based Design

⇒ In designing a distributed embedded system, we must schedule computations in time & accelarate them to processing elements.

⇒ scheduling & allocation of communications are also important.

⇒ Many embedded system networks are designed for low cost and therefore reduce the communication speed. It may lead the bottleneck in system design.

Communication Analysis

⇒ To analyze the performance of network we must determine the delay incurred by transmitting messages.

⇒ Let us assume for the moment that messages are sent reliably, no retransmit a message & no contention can be modeled as $t_m = t_x + t_n + t_r$

t_x = transmitter-side overhead
t_n = network transmission time
t_r = Receiver side over head

Timers / counters:

Timers - to generate a time delay

Counters - to count events happening outside the microcontroller

In 8051 two counters / timers are there, T_0 and T_1. It is programmed to count internal clock pulses, acting as a timer or may be programmed to count external pulses acting as a counter.

Registers of timer 0:

16 bit - register divided into two 8 bit - registers (TL0) and (TH0)

Registers of timer 1:

16 bit - register divided int TL1 and TH1.

Control Registers:

Timer / counter action is controlled by

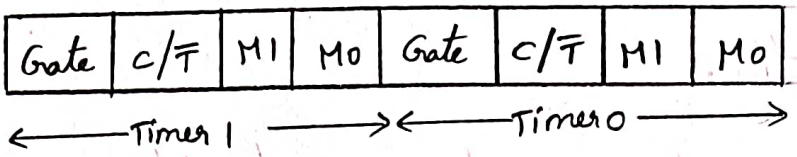
* TMOD register:

To set the various timer operation modes.

Dedicated solely to two timers and can be

considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers.

It is not bit addressable.



Gate :

OR gate enable bit that controls RUN/STOP of timer 1/0

Timer/counter enabled while TR 1/0 is set and $\overline{INT}$ 1/0 is high

C/$\overline{T}$: (clock/Timer)

Decide whether the timer is used as timer/counter.

C/$\overline{T}$ = 0 → timer

C/$\overline{T}$ = 1 → counter

M1:

Timer/counter operating mode select bit 1.

Set/cleared by program to select mode

M0 :

Timer/counter operating mode select bit 0.

Set/cleared by program to select mode

TCON Register:

Has control bits and flags for

timer in upper nibble

external interrupts in lower nibble

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1:

↳ Timer 1 overflow flag

↳ Set by hardware when timer/counter 1 overflows.

↳ cleared when processor vectors to execute ISR location 001B

TF0:

↳ Timer 0 overflow flag

↳ Set to enable timer/counter 0 overflow

↳ cleared → to execute ISR location 0001

IT1:

Interrupt 1 type control bit

Set → enable external interrupt 1 while falling edge signal

cleared → low-level signal on external interrupt 1.

IT0: Interrupt 0 type control bit

Set → enable interrupt 0 → falling edge

cleared → low signal on external interrupt 0.

TR1:

↳ Timer 1 run control unit

↳ Set by program to enable timer ~~to~~ count

↳ cleared by program to halt the timer.

TR0:

↳ Timer 0 run control flag

↳ Set → enable timer to count

↳ cleared → to halt timer

IE1:

↳ External Interrupt 1 edge flag

↳ Set → high to low edge signal is received

↳ cleared → ISR located at 0013

IE0:

External interrupt 0 edge flag

Set → high to low edge signal is received on $\overline{INT_0}$

cleared → ISR located at 0003

Timer mode of operation:

4 modes are there

Timer Mode 0:

Has 13 bit timer

THx register – 8 bit counter & TLx register – 5 bit counter.

Values of 000H to 1FFFH. I/p divided by 32 in TL so TH counts original frequency reduced by 384.

Timer Mode 1:

Has 16 bit timer → THx register of 8 bit counter and TLx register as 8 bit counter allows values of 0000 to FFFF

Pulse input divided to 256 in TLx so THx counts frequency reduced by 30720.

Timer Mode 2:

Has 8 bit timer allow values of 00H to FFH to be loaded into timer's register THX. After THx is loaded with the 8 bit value, the microcontroller give copy of it to TLx. Then timer starts.

It is an auto reload feature: TLx will count up from the number in overflow and be initialized again with contents of THx.

Timer Mode 3:

Timer 0 and 1 may be programmed to be mode 0, 1 and 2. But mode 3 is chosen for timer 0. placing timer 1 in mode 3 causes it to stop counting, the control bit TR1 and timer 1 flag TF1 are then used by timer 0.

Timer 0 in mode 3 becomes two completely seperate 8-bit counters. TH0 receives the timer clock under the control of TR1 only and sets the TF1 flag when it overflows.

UART

* Universal Asynchronous Receiver / Transmitter.

* Traditional Dobinition : converts parallel (8 bit) data to serial data and vice versa.

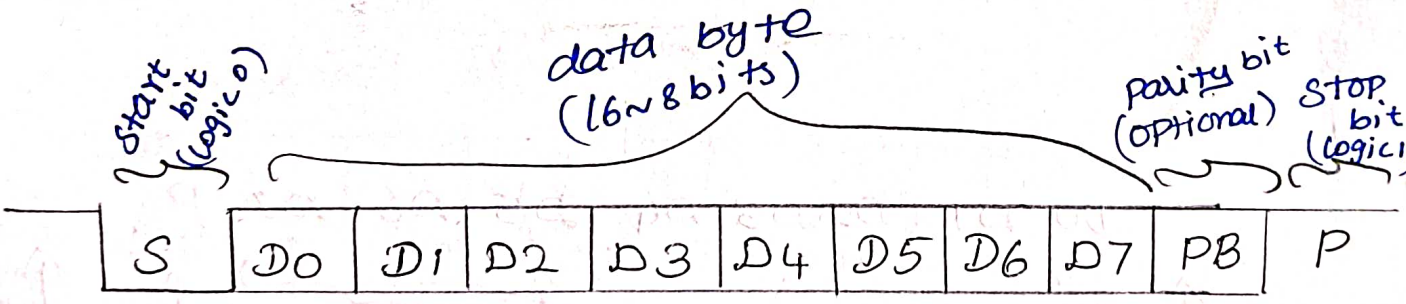
↳ UART is a simple half duplex, asynchronous serial Protocol.

↳ simple connections between two equivalent nodes.

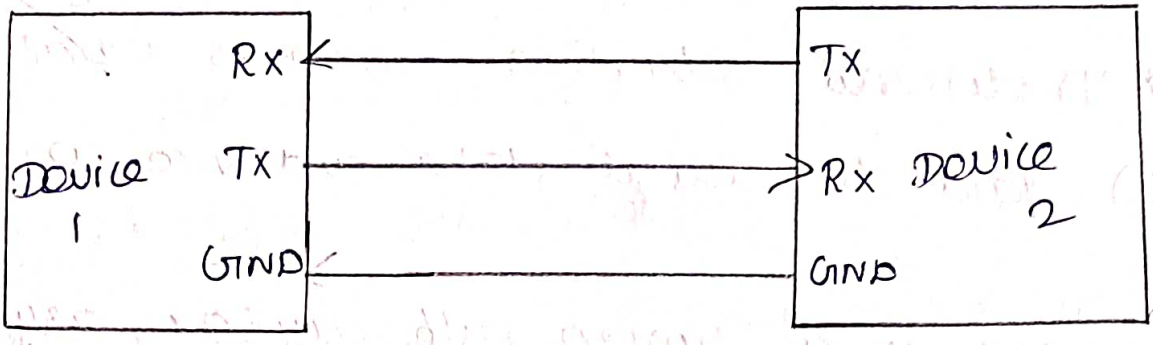
↳ Any node can initiate communication.

↳ Since connection is half duplex, the two lanes of communication are completely independent.

Packet format:



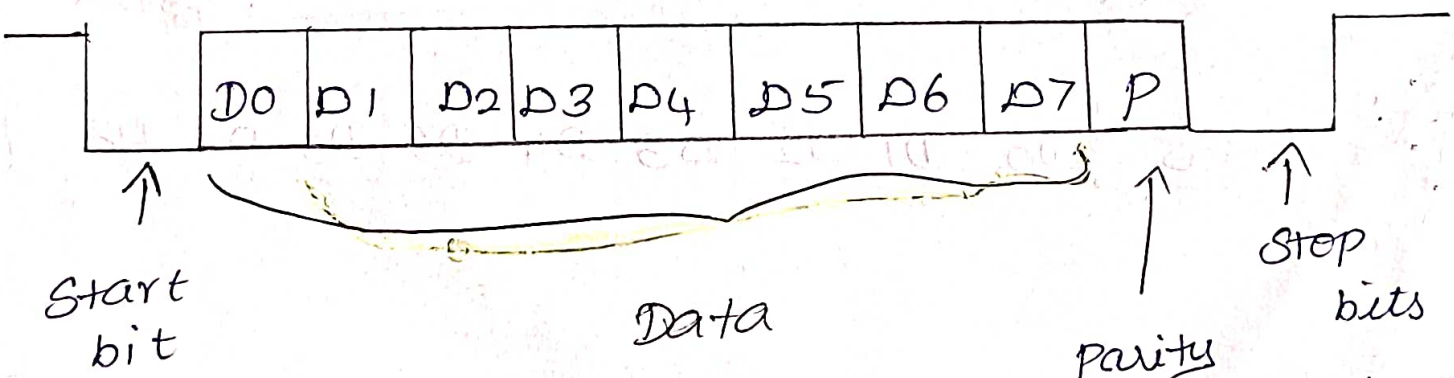
Connections for UART:



UART characteristics:

* The speed of communication (measured in bauds) is predetermined on both ends.
* A general rule of thumb is to use 9600 bauds for wired communication.
* UART implements error-detection in the form of parity bit.

Parity bit



* Parity bit is HIGH when number of 1's in the Data is odd.

* Respectively, it is LOW when number of 1's in the Data is even.

Baud Rate Generator:

* Used to generate the baud rates for both the transmitter and receiver.
* Not required for any other function including reads and writes.
* Crystal or External clock.
* 16 bit divisor Programmed in DLM / DLL registers.

$$\text{Baud Rate} = \frac{\text{clock frequency}}{(\text{Sampling Rate}) \times (\text{Divisor})}$$

Sampling Rate is 16.

Transmitter:

- Parallel to serial communication.
- Non FIFO mode
  - * Transmit holding Register (THR) and transmit shift register (TSR)
- FIFO mode
- 16x timing for bit shifting
- character framing
- Parity insertion
- TX FIFO interrupt and status.

Receiver:

- Serial to Parallel conversion.
- Non FIFO mode (RHR and RSR)
- FIFO mode (RX FIFO and RSR)
- 16x timing clock for mid bit sampling.
- Start bit detection and verification.
- Rx FIFO is 11 bits wide
  - * 8 data bits
  - * 3 error or error tags.

UNIT - III Embedded Programming

Components for Embedded Programs.

⇒ Embedded software uses three components.

1. State machine
2. Circular buffer
3. Queue

⇒ State machines are well suited to reactive system such as user interfaces, Circular buffers and queues are useful in digital signal processing.

State Machine:

* A state machine is any object that behaves different based on its history and current inputs.
* Many embedded systems consists of a collection of state at various levels of the electronics or software.
* In general, a state machine is any device that stores the status of something at a given time and can operate on input

to change the status and/or cause an action or output to take place for any given change.

In practice, however, state machines are used to develop and describe specific device or program interactions.

⇒ To summarize it, a state machine can be described as:

1. A set of states
2. An initial state or record of something stored somewhere
3. A set of input events.
4. A set of output events.
5. A set of actions or output events that maps the states and input to output.
6. A set of actions or outputs events that maps the states and inputs to states

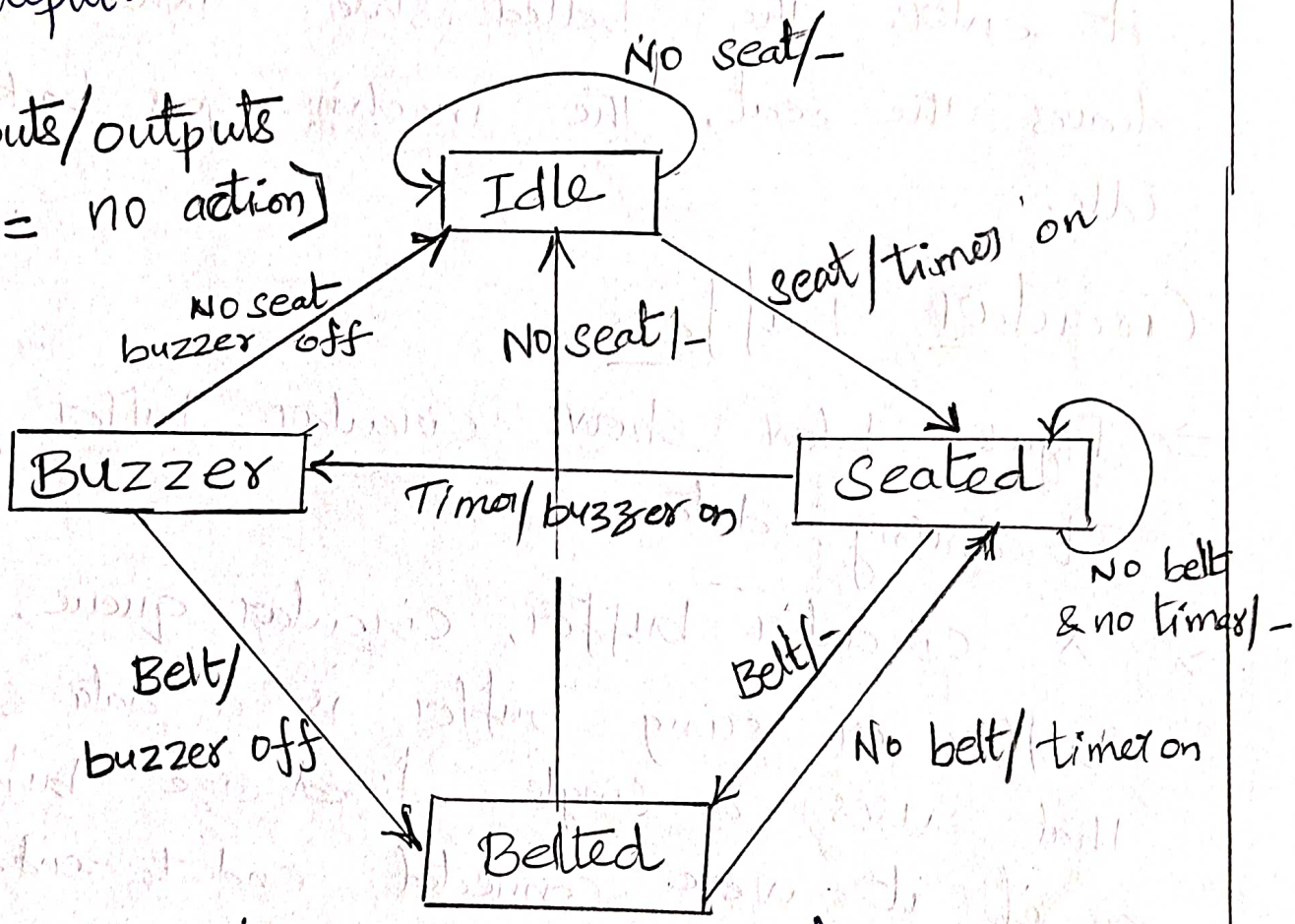
Finite State Automation (FSA), Finite state Machine (FSM) or state Transition Diagram (STD) is a formal method used in the specification and design of wide range of embedded and real time systems.

State Machine for seat belt controller.

⇒ Controllers job is to turn on a buzzer if a person sits in a seat and does not fasten the seat belt within a fixed amount of time.

⇒ This system has three inputs and one output.

Inputs/outputs
['-' = no action]



⇒ The inputs are a sensor for the seat to know when a person has sat down, a seat belt sensor that tells when the belt is fastened and a timer that goes off when the required time interval has elapsed. The output is the buzzer.

⇨ The idle state is in force when there is no person in the seat. When the person sits down, the machine goes into the seated state and turns on the timer.

⇨ If the timer goes off before the seat is fastend, the machine goes into the buzzer state. If the seat goes on first, it enters the belted state. When person leaves the seat, the machine goes back to idle.

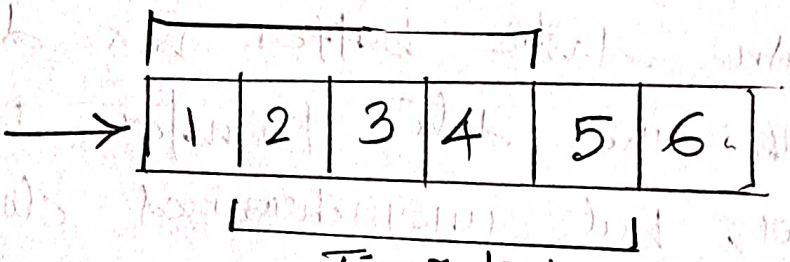
Circular Buffers

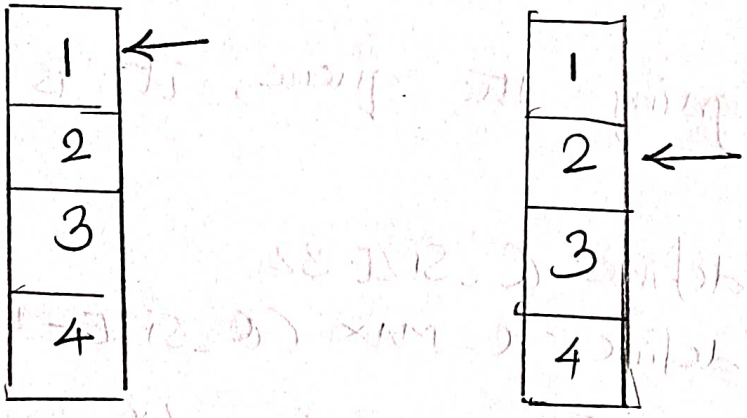
⇨ Figure below shows circular buffer for streaming data.

⇨ A circular buffer, circular queue, cyclic buffer or ring buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. This structure lends itself easily to buffering data streams.

⇨ The circular buffer behaviour is ideal for implementing any data structure that is statically allocated and behaves like FIFO

⇒ Circular buffer are a special type of buffer where the data is circulated around a buffer. In this way they are similar to a single buffer that moves the next data pointer to the start of the buffer to access the next data. In this way the address pointer circulates around the addresses.





Circular buffer

⇒ When a buffer underruns, it indicate that there is no more data in the buffer and that further processing should be stopped. This may indicate an error if the system is designed so that it would never run out of data

⇒ If it can happen in normal operation then the data underrun signal indicates a state and not an error. In both cases, a single signal is needed to recognise this point.

Queue

⇒ Queues are also used in signal processing and event processing.

⇒ Queues are used whenever data may arrive and depart at somewhat unpredictable times.

⇒ Queue is also referred to as an elastic buffer. An elastic buffer is a device that helps smooth the data transfer between two similar, but unsynchronized clock domains.

⇒ Linked list is used for building queue.

For designing the queue, it is declares as follows:

```
# define Q_SIZE 32
# define Q_MAX (Q_SIZE-1)
int q[Q_SIZE];      /* array for queue */
int head, tail      /* position of head and
                       tail in the queue */
```

Models of Programs

⇨ The basic concept is to enable the execution of an instruction whenever its required operands become available. Programs for data driven computations can be represented by data flow graphs.

⇨ Each instruction in a data flow computer is implemented as a template, which consists of the operator, operand receivers and result destinations.

⇨ Operands are marked on the incoming arcs and results are on outgoing arcs.

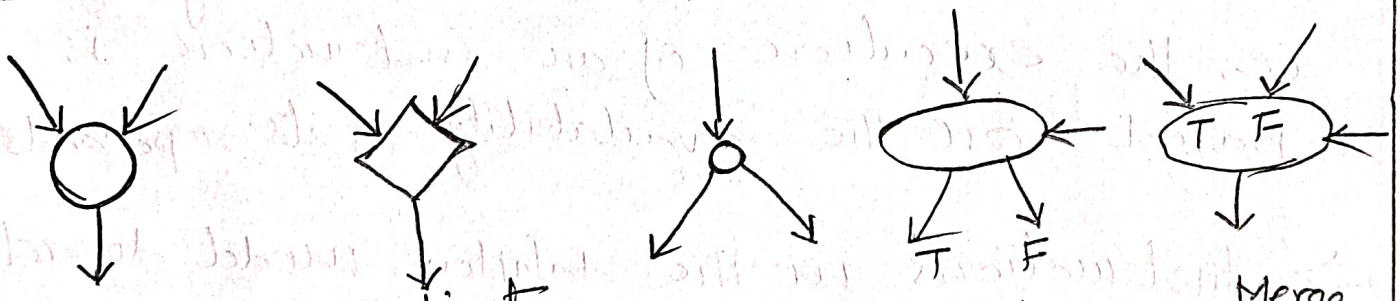
⇨ Dataflow model of execution is asynchronous, i.e., the execution of an instruction is based on the availability of its operands.

⇨ Instructions in the dataflow model do not impose any constraints on sequencing except the data dependencies in the program.

⇨ The dataflow model incurs more overhead in the execution of an instruction cycle compared to its control-flow counterpart due to its fine-grained approach to parallelism.

⇨ In dataflow machines each instruction is considered to be a seperate process. To facilitate data-driven execution each instruction that produces a value contains pointers to all its consumers. Since an instruction in such a dataflow program contains only references to other instructions. it can be viewed as a node in a graph.

⇨ Data flow program is represented as a directed graph. $G = G(N, A)$, where nodes in 'N' represent instructions and arcs in A represent data dependencies between the nodes. The operands are conveyed from one node to another in data packets called tokens.



Operator Predicate Copy Switch Merge

Basic primitives of the dataflow graph.

In dataflow computers, the machine level language is represented by dataflow graphs.

The above figures shows basic primitives of the dataflow graph.

- In dataflow machines each instruction is considered to be an instruction cycle compared to its "control flow" counterpart due to its fine-grained approach to parallelism.
- In dataflow machines each instruction is treated to be a seperate process. To facilitate data-driven execution each instruction that produces a value contains pointer to all its consumers.
- Dataflow program is represented as a directed graph, $G = \{G[N, A]$, where nodes in N represent instructions and arcs in 'A' represent data dependencies between the nodes. The operands are conveyed from one node to another in data packets called tokens.
- Dataflow graph exhibits two kinds of parallelism in instruction execution.

a) **Spatial parallelism**: Any two nodes can be potentially executed concurrently if there is no data dependence between them.

b) **Temporal parallelism**: This type of parallelism results from pipelining independent waves of computation through the graph.

- The dataflow graph is similar to a dependence graph used in intermediate representation of compilers.

Static Model

⇨ The static model allows at most one instance of a node to enabled for firing A dataflow actor can be executed only when all of the tokens are available on its input arcs can and no tokens exist on any of its output arcs.

Limitation of Static Model:

1. Consecutive iterations of a loop can only be pipelined.
2. Due to acknowledgement tokens, the token traffic is doubled.
3. lack of support for programming constructs that are essential to modern progeamming language.

Dynamic Model

It permits activation of several instances of a node at the same time during run-time. To distinguish between different instances of a node, a tage is associated with each token that identifies the context in which a particular token was generated.

- Advantage: Better performance as it allows multiple tokens on each thereby unfolding more parallelism
- Associative memory would be ideal.

Assembly linking and loading:

⇨ Assembly and linking are the last steps in the compilation process. They turn a list of instructions into an image of the program's bits in memory.

⇨ A compiler is a computer program that translates a program written in a high-level language to the machine language of a computer.

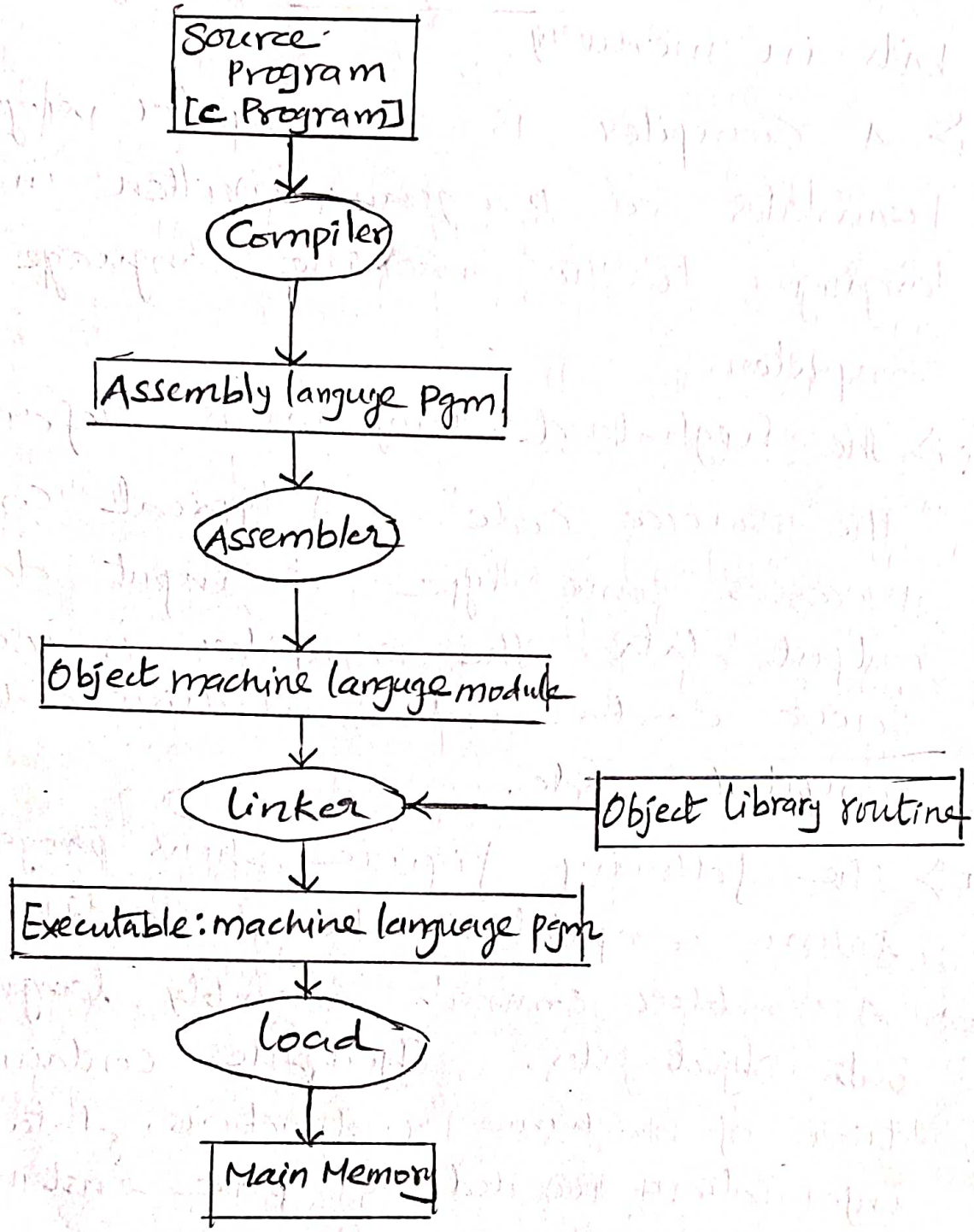
⇨ The high-level program is referred to as "the source code". A typical computer program processes some type of input data to produce output data. The compiler is used to translate source code into machine code or compiled code.

⇨ The following figure shows program generation from compilation through loading.

⇨ Assembler converts assembly language programs into object files. Object files contain a combination of machine instructions, data and information needed to place instructions properly in memory.

⇨ Linker merges the objects files produced by seperate compilation or assembly and creates an executable file-

loader: Part of the OS that brings an executable file residing on disk into memory and starts it running



Program generation from compilation through loading.

Assembler:

⇨ The assembler is responsible for translating the assembly language program into machine code. When the source code is essentially a simbolic representation for a numerical machine language is called an assembly language.

⇨ A pure assembly language is a language in which each statement produces exactly one machine instruction.

⇨ When translating assembly code into object code, the assembler must translate opcodes and format the bits in each instruction, and labels into addresses.

label: It is an identifier and optional field. Labels are used extensively in programs to reduce reliance upon programmers when data or code is located. The maximum length of label differs between assemblers. Some accept upto 32 characters long, other only four characters.

⇨ The name of each symbol and its address is stored in a symbol table that is built during the first pass. The symbol table is built by scanning from the first instruction to the last.

⇨ Symbol table contains name and address field. Symbol table is built by the analysis phase. It also contains flag to indicate errors.

⇨ During scanning the current location in memory is kept in a Program location counter (PLC).

⇨ Memory allocation mean the fixing the address of the assembly language statement. Suppose we want to fix the memory address of M, then also fix the address of remaining instructions.

⇨ location counter is data structure used to implement the memory allocation. location counter [LC] is always made to contain the address of the next memory word in the target program.

⇨ SYMTAB includes the name and value for each label in the source program, togethe with flags to indicate error conditions. It also contains information about the data area or instruction labeled.

⇨ During Pass 01: labels are entered into SYMTAB as they are encountered in the source program along with their assigned address.

⇨ During Pass 02, symbols used as operand as a hash table for efficiency of insertion and retrieval. Entries are rarely deleted from this table.

⇨ It is possible for both passes of the assembler to read the original source program as input. Information such as location counter values and error flag statement can be communicated between the two passes.

⇨ For this reason, Pass I usually writes an intermediate file that contains each source statement together with its assigned address, error indications etc. This file is used as the input to Pass 2.

Linking

⇨ This program might make use of other programs, or libraries of programs that are linked together into a single program and the interconnecting references are resolved.

⇨ Linker is a program which combines the target program with the code of other programs and library routines.

⇨ During the process of linking the object module is created. This object module contains the target code and information about other programs and library routines that are required to call during the program execution.

⇨ The output of translator is a program called object module. The linker processes these object modules binds with necessary library routines and prepares a ready to execute program. Such a program is called binary program.

⇨ The binary program also contains some necessary information about allocation and relocation. The loader then loads this program into memory for execution purpose.

⇨ The place in the file where a label is defined is known as an entry point. The place in the file where the label is used is called an external reference. The main job of the loader is to resolve external reference based on available entry points.

Compilation Techniques

- Compilation combines translation and optimization. The high-level language program is translated into the lower-level form of instructions; optimization try to generate better instruction sequences than would be possible if the brute force technique of independently translating source than would be possible. if the brute force technique of independently translating source code statements were used.

1. **Lexical analysis:** The lexical analysis is also called scanning. It is the phase of compilation in which the complete source code is scanned and your source program is broken up into group of string called token.

2. **Syntax analysis:** The syntax analysis is also called parsing. In this phase the tokens generated by the lexical analyser are grouped together to form a hierarchical structure. The syntax analysis determines the structure of the source string by grouping the token together.

3. Semantic analysis: Once the syntax is checked in the syntax analyser phase the next phase i.e. the semantic analysis determines the meaning of the source string. For example meaning of source string means matching of paranthesis in the expression, or matching of if .... else statement or performing arithmatic operations of the expressions that that are type compatible, or checking the scope of operation.

4. Intermediate code generation:
The intermediate code is a kind of code which is easy to generate and this code can be easily converted to target code. This code is in variety of forms such as three address code, quardruple, triple, posix.

5. Code optimization: The code optimization phase attempts to improve the intermediate code. This is necessary to have a faster executing code or less consumption of memory.

6. Code generation: In code generation phase the target code gets generated. The intermediate code instructions are translated into sequence of machine instruction.

Program Level Performance Analysis:

⇨ Execution time of a program often varies with the input data values because those values select different execution paths in the program.

⇨ Cache has a major effect on program performance.

⇨ Execution times may vary even at the instruction level.

⇨ Floating-point operations are the most sensitive to data values, but the normal integer execution pipeline can also introduce data-dependent variations.

- Program performance is measured in following ways:

1. Simulator of CPU supplied by manufacture.
2. Timer connected to the microprocessor bus can be used to measure performance of executing sections of code.
3. A logic analyzer can be connected to the microprocessor bus to measure the start and stop times of a code segment.

Elements of Program Performance.

- Program execution time is given as

Execution time = Program path + Instruction timing.

- The path is the sequence of instructions executed by the program. This instruction timing is determined based on the sequence of instructions traced by the program path, which takes into account data dependencies.
- Program execution times depend on several factors:

1. **Input data values:** Different values, different execution paths.
2. **Cache behavior:** Also dependent on input values.
3. **Instruction level:** Floating-point operations and pipelining effects.

⇒ Program paths offer insight into a program's dynamic behavior that is difficult to achieve any other way. Unlike simpler measures such as program profiles, which aggregate information to reduce the cost of collecting or storing data, paths capture some of the usually invisible dynamic sequencing of statements.

⇒ Examination of programs path has unveiled a striking degree of path locality, which the Computer/compiler communities have potentially exploited to increase program performance.

Software Performance Optimization

loop Optimizations.

- Optimizing loops particularly important in compilation, since loop account for much of the executions times of many programs.
- The code optimization can be significantly done in loop of the program. Specially inner loop is a place where program spends large amount of time.
- Hence if number of instructions are less inner loop then the running time of the program will get decreased to a large extent. Hence loop optimization is a technique in which code optimization performed on inner loops.
- Three methods are used for loop optimizations: **code motion, instruction variable elemination. and strength reduction.**

1. **code motion**: It is a technique which moves the code outside the loop. Hence is the name. If there is a technique which moves the code. If there lies some expression in the loop whose result remains unchanged even after executing the loop for several times, then such an expression should be placed just before the loop

- Here before the loop means at the entry of the loop.

```
While (i <= Max-1)
  {
    sum = sum + a[i];
  }
```

Before optimization

```
n = Max-1
While (i <= n)
{
    sum = sum + a[i];
}
```

After optimization.

2. Instruction Variables

* An induction variable is a variable in a loop, whose value is a function of the loop iteration number $V = f(i)$
* A variable x is called an induction variable of loop L if the value of variable gets changed every time. It is either decremented or incremented by some constant.

For example:

```
B1
i := i + 1
t1 := 4j
t2 := a[t1]
if t2 < 10 goto B1
```

* In above code the values of i & t_1 are in locked state. That is when value of i gets incremented by 1 then t_1 gets incremented by 4. Hence i and t_4 are induction variables.

3. Strength reduction:

The strength of certain operators is higher than others. For instance strength of * is higher than +. In strength reduction technique the higher strength the higher strength operators can be replaced by lower strength operations.

Program Level Energy and Power Analysis and optimization :

* Power consumption is a particularly important design metric for battery-powered systems because the battery has a very limited lifetime.
* Power consumed by the CPU is a major part of the total power consumption of a computer system and thus has been the main target of power consumption analysis.
* How long the device needs to run and whether the batteries can be recharged, need to be thought out head of time. In some systems, replacing a battery in a device can be a big expense.
* There are several methods to conserve power in an embedded system, including clock control, power-sensitive processors, low-voltage ICs and circuit shutdown.
* By measuring the current drawn by the processor as it repeatedly executes distinct instructions or distinct instruction sequences, it is possible to obtain most of the information that is required to evaluate the power consumption.

* Power is modeled as a base cost for each instruction plus the inter-instruction overheads that depend on neighboring instructions.

* The base cost of an instruction can be considered as the cost associated with the basic processing needed to execute the instruction

* However, when sequences of instructions are considered, certain inter-instruction effects come into play, which are not reflected in the cost computed ~~&~~ solely from base cost.

* 1. Circuit state: Switching activity depends on the current inputs and previous circuit state.

2. Resource constraints: Resource constraints in the CPU can lead to stalls e.g. pipeline stalls and write buffer stalls.

3 Cache misses: Another inter-instruction effects is the effect of cache misses.

- As the instruction cache size increases, the energy cost of the software on the CPU declines, but the instruction cache comes to dominate the energy consumption.

* If the cache is too small, the program runs slowly and the system consumes a lot of power due to the high cost of main memory accesses.

* If the cache is too large, the power consumption is high without a corresponding payoff in performance. At intermediate values the execution time and power consumption are both good.

Methods for improving energy consumption:

1. Try to use registers efficiently
2. Analyze cache behaviour to find major cache conflicts;
3. Make use of page mode accesses in the memory system whenever possible.

* Some additional observations about energy optimization as follows:

1. Moderate loop unrolling eliminates some loop control overhead.
2. Software pipelining reduces pipeline stalls, thereby reducing the average energy per instruction.
3. Eliminating recursive procedure calls where possible saves power by getting rid of function call overhead.

Analysis and Optimization of program size:

- Data provide an excellent opportunity for minimizing size because the data are most highly dependent on programming style.
- In data dominated applications, such as image or speech signal processing applications. summing up the sizes of all the arrays is the most straightforward way to get an upper bound of the memory requirement.
- In the data dependency relations in the code are used to find the number of array elements produced or consumed by each assignment, from which a memory trace of upper and lower bounding rectangle as a function of time is found.
- Care should be taken while designing buffer size. Data can sometimes be packed. Such as by storing several flags in a single word and extracting them by using bit-level operations.

* A very low-level technique for minimizing data is to reuse values. Data buffers can often be reused at several different points in the program.
* Minimizing the size of the instruction text of a program requires a mix of high-level program transformations and careful instruction selection.
* Encapsulating functions in subroutines can reduce program size when done carefully.

Program Validation and Testing:

* Testing is an organized process to verify the behavior, performance, and reliability of a device or system against designed specifications.
* Debugging is the process of removing defects (bugs) in the design phase to ensure that the synthesized design, when manufactured will behave as expected. Testing is a manufacturing step to ensure that the manufactured device is defect free.

⊗ Embedded software development uses specialized compilers and development software that offer means for debugging. Developers build application software on more powerful computers and eventually test the application in the target processing environment.

o Testing methods are of two types.

1. Black-box testing: This method generates tests without looking at the internal structure of the program.

2. White box testing: This method generate tests looking based on the program structure. This method also called as clear-box testing.

Black Box Testing: It is also called functional testing. It is testing that ignores the internal mechanism of a system or component and focuses solely on the outputs generates in response to selected inputs and execution conditions.

- With black box testing, the software tester does not have access to the source code itself. The code is considered to be a "big black box" to the tester who can't see inside the box.
- Black-box is based on requirements and functionality, not code.
- Random tests from one category of black-box test. Random values are generated with a given distribution.
- The expected values are computed independently of the system, and then the test inputs are applied. A large number of tests must be applied for the result to be statistically significant, but the tests are easy to generate.
- Using black box testing techniques, testers examine the high-level design and the customer requirements specification to plan the test cases to ensure the code does what it is intended to do.
- Functional testing involves ensuring that the functionality specified in the requirement specification works.

* System testing involves putting the new program in many different environments to ensure the program works in typical customer environments with various versions and types of operating systems and/or applications.

Advantages:

1. Tests the final behavior of the software.
2. Can be written independent of software design.
3. Can be used to test different implementation with minimal changes.

Disadvantages:

1. Doesn't necessarily know the boundary cases.
2. Can be difficult to cover all portions of software implementation.

White Box Testing:

⇨ Often called "structural" testing.

⇨ Knowing the internal working of a product, test that all internal operations are performed according to specifications and all internal components have been exercised.

UNIT-IV REAL TIME SYSTEMS

Structure of Real Time System

⇒ Real time system is an example of a general purpose operating system. This system is used when the requirements are inelastic for data flow or operations of processors, so, it is often used as a control device in dedicated applications.

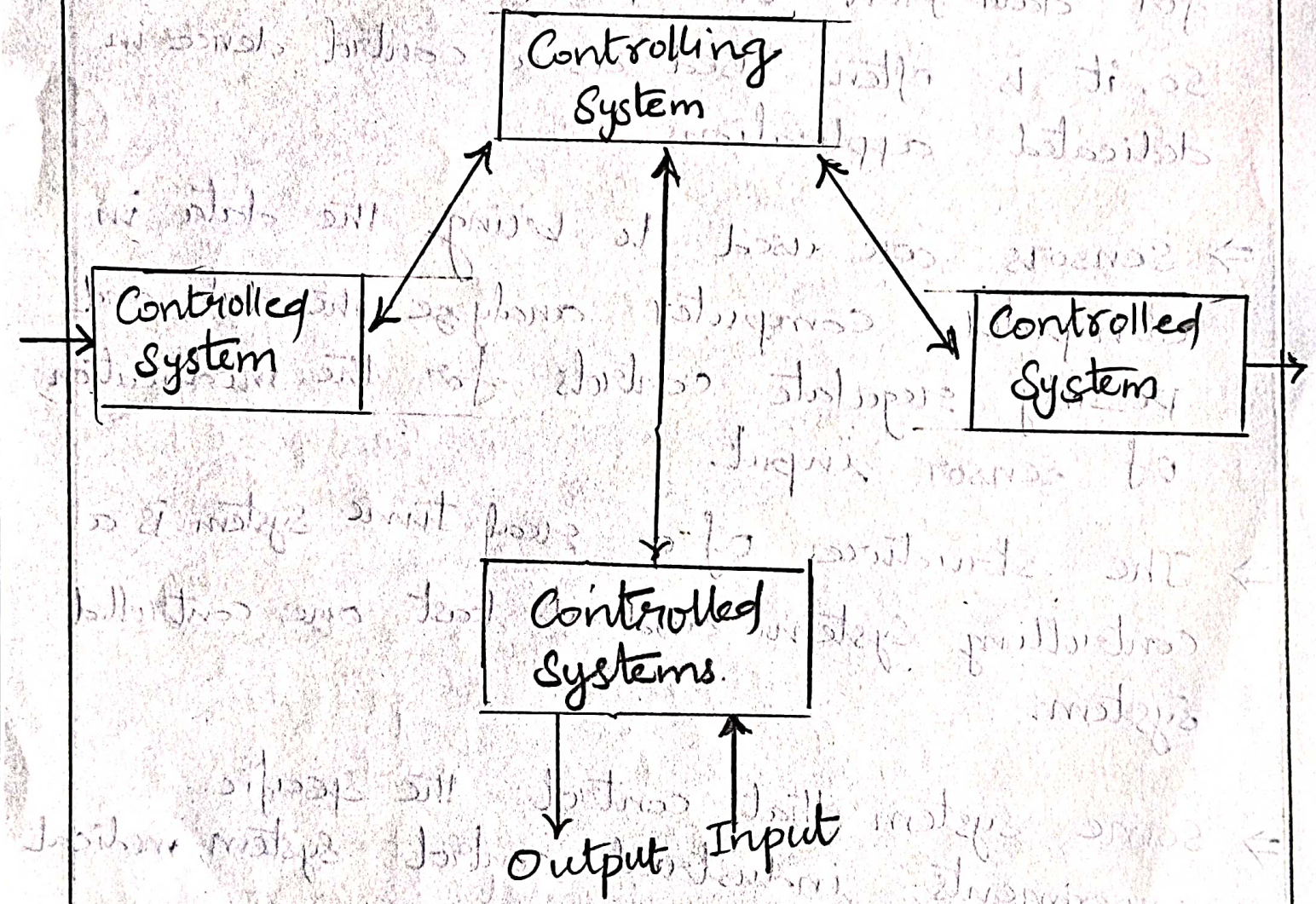
⇒ Sensors are used to bring the data in computer. Computer analyze the data and possibly regulate controls for the modification of sensor input.

⇒ The structure of a real time system is a controlling system and at least one controlled system.

⇒ Some system that control the specific experiments, industrial control system, medical imaging system and some display systems are different forms of real time system.

⇒ It also contains some systems like automobile engines, fuel injection systems, weapon systems and home appliances controller.

⇒ In real time system, there is fixed time constriants. Processing is required to be done within defined contraints otherwise the system will fail.



⇒ The functionality of the real time system is considered to be correct only if it returns the correct results with in any time constrains.

⇒ Control a device using actuator, based on sampled sensor data. It also control loop compares measured value and reference value. Reference input, accuracy of measurements depends on correct control law computation.

⇒ Time between measurements of $y(t)$, $x(t)$ is the sampling period, T. small T gives better approximates analogue control, but large T needs less processor time: if T is too large, oscillation will result as the system fails to keep up with changes in the input.

⇒ Analysis of a control system involves the determination of the system response. This can be carried out experimentally, or by estimating the response on the basis of a system model. Then, it is the task of control system design to achieve a desired overall system response by modifying the controller

⇒ The controller will have its control strategy upon the disturbed signal and might drive the plant into an undesire state. In many designs, the sampler is preceded by an anti-alias filter to avoid the effect of aliasing.

Hard Real Time System:

⇒ A hard real-time system is one where the response time is specified as an absolute value. This time is normally dictated by the environment.

⇒ A system is called a hard real-time if tasks always must finish execution before their deadlines or if message always can be delivered within a specified time interval.

⇒ Hard-real time is often associated with safety critical applications.

⇒ Missing a deadline may be catastrophic. Critical deadline is called hard deadline.

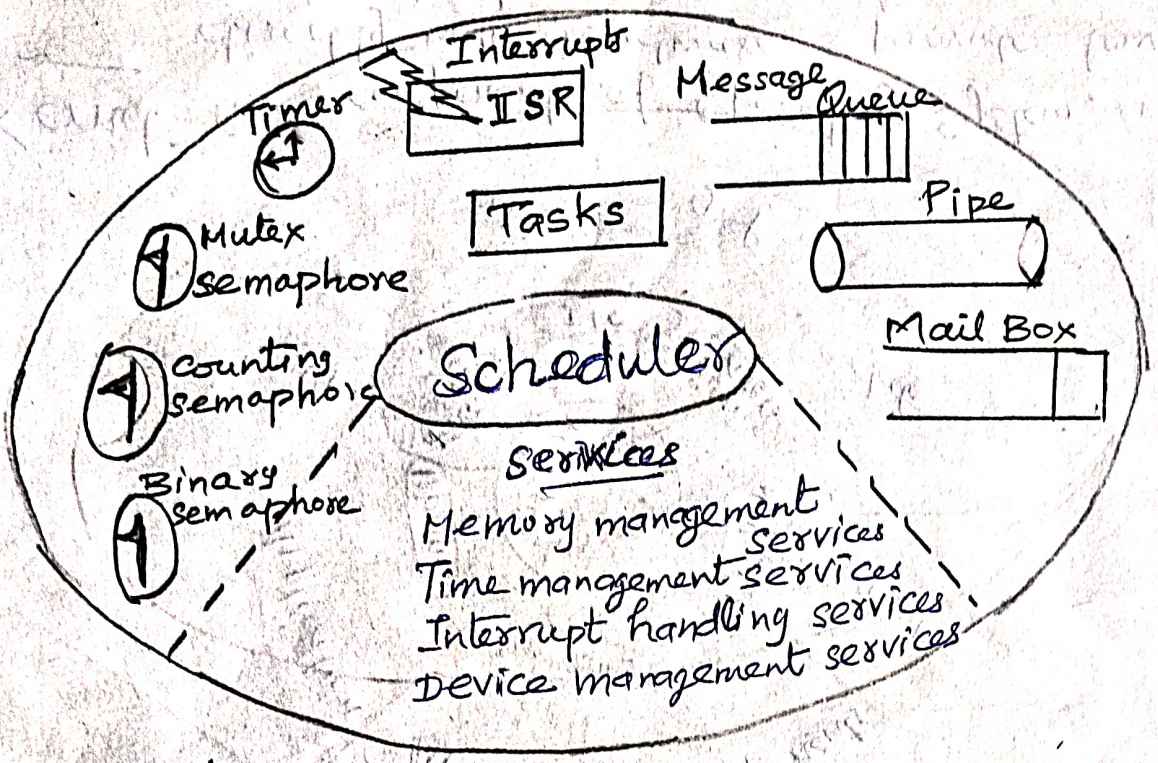
Soft Real Time System:

⇒ A soft real-time system is one where the response time is normally specified as an average value. The time is normally dictated by the business or market.

⇒ A single computation arriving late is not significant to the operation of the system, through many late arrivals might be.

⇒ Soft real time means that only the precedence and sequence for the task-operations are defined, interrupt latencies and context switching latencies are small but there can be few deviations between expected latencies of the task & observed time constraints and a few deadline misses are accepted.

KERNEL



Kernal

The various objects of kernel are Tasks, Task scheduler, Interrupt service Routines, Semaphores, Mutexes, Mail Boxes, Message Queues, Pipes, Timers etc,

Scheduler

The schedular is the heart of every kernel. A scheduler provides the algorithms needed to determine which task executes when.

Schedulable Entities:

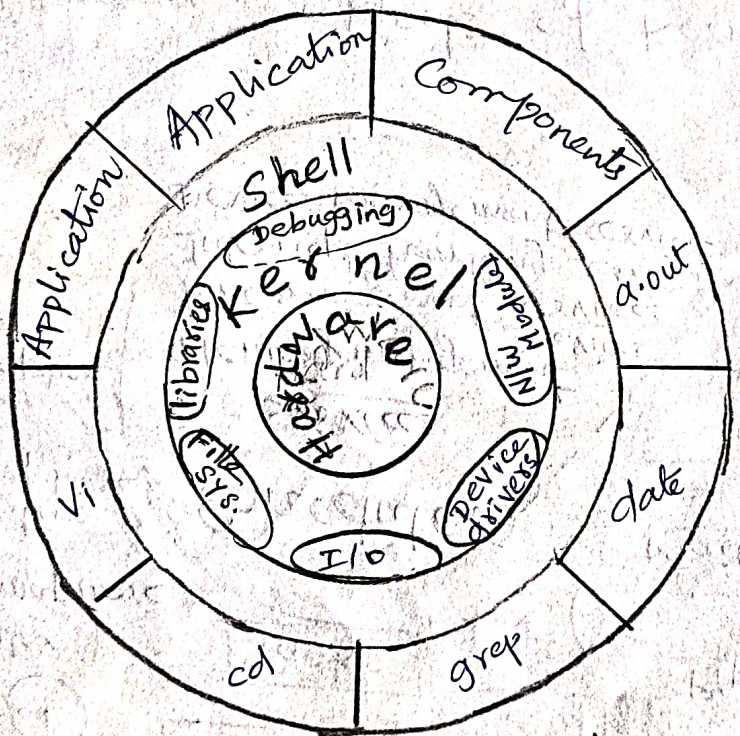
It is a kernel object that can be completed for execution time on a system based on predefined scheduling algorithm. Tasks and Process are example of schedulable entities found in most kernel.

RTOS Architecture

A real-time OS is a program that schedules execution in a timly manner, manages system resources, provides a consistant foundation for developing application code.

RTOS is a multi-tasking OS intended for real time applications to produce high and guaranteed throughput on disired time.

Hardware: It consists of all peripherals, Processor & Memory



Kernel: Core components of Operating System, interact directly with hardware & provide low level services to layer components

shell: An interface to kernel, hiding complexity of kernel's function from users. Takes commands from user & executes kernel's functions.

UNIT-IV REAL-TIME SYSTEM

REAL-TIME CHARACTERISTICS

Real-Time System is one whose logical correctness is based on both correctness of the outputs and their timeliness

⇨ Real-time systems are those systems in which the overal correctness of the system depends on both the functional correctness and the timing correctness.

⇨ Real-time system also have a substantial knowledge of the system it controls and the applications running on it.

⇨ Deadline dependent.

⇨ Predictability is important.

⇨ Deadline – a time within which the task should be completed.

⇨ Hard Real-Time System – system fails if deadline window is missed
Ex: aircraft control

⇨ Soft Real-Time System – system will be undesirable for performace reason if deadline window is missed.
Ex: multimedia applications

⇨ Firm deadline – Missing a deadline makes the task useless (similar to hard RT), however the deadline may be missed occasionally (similar to soft deadline).

⇨ Most systems: combination of both hard & soft deadlines

⇨ Generalization: cost function associated with missing each deadline.

Characteristics of RT systems:

⇨ Real-time systems are often Embedded systems.

⇨ They often require concurrent processing of multiple inputs.
- concurrent task must be created & managed in order to fulfill the functions of the system.

⇨ Task scheduling is one of the important aspects of managing concurrency
- Since tasks will compete for the same resources (such as processors)

⇨ Real-time systems need to respond to synchronous events (i.e. periodic events) as well as asynchronous events (i.e. aperiodic events).

⇨ Real-time systems often requires high Reliability & safty requirements.

⇨ Environmental factors such as temperature, shock, vibration, size limits & weight limits usually have an impact on the system hardware & software requirements.

⇨ Fault-tolerant requirements & Exception handling have special consideration due to the high reliability & Critical timing requirements.

⇨ Interfacing requirements: The devices which are typically interfaced to a RTS.

⇨ Guaranteed response times
- We need to able to predict with confidence the worse case response times for system: efficiency is important but predictability is essential.

⇨ Job; unit of work that is scheduled & executed by the system
- Computation of a FFT [Fast Foureir Transform]
- Transmission of a data packet

⇨ Task; a set of related jobs which jointly provide some system function.

⇨ Release time;
- The instant of time at which the job becomes available for execution
- Job have no release time if all the jobs are released when the system begins execution

⇨ Response time;
- The length of time from the release time of the job to the instant when it completes

⇨ Relative deadline:
- The maximum allowable response time of a job

⇨ Deadline or Absolute Deadline;
- The instant of time by which its execution is required to be completed
- Equal to the release time plus the relative deadline.
- A job has no deadline if its deadline is at infinity.

• The set of rules that determines the order in which tasks are executed is called a scheduling algorithm.

- A schedule is feasible if all tasks can be completed according to the timing constraints
- A set of tasks is schedulable if there exists at least one algorithm that can produce a feasible schedule.

⇨ A Periodic task is executed repeatly at regular time intervals and each invocation is called a job or instance
- often time-driven

⇨ A Aperiodic task is executed to response to external events and to respond, it executes aperiodic jobs whose release time are not know a priori.
- Often event-driven
- Off-line guarantee of aperiodic tasks must make proper assumption on the environments; that is, by assuming a maximum arrival rate for each event (i.e. minimum interarrival time)
- Aperiodic tasks characterized by a minimum interarrival time are called Sporadic tasks.

Types of Scheduling

⇨ Preemptive: the running tasks can be interrupted to assign the processor to another task.

⇨ Non-preemtive; A task, once started, is executed until completion.

⇨ Static; the scheduling decisions are based on fixed parameters and assigned to tasks before their activation

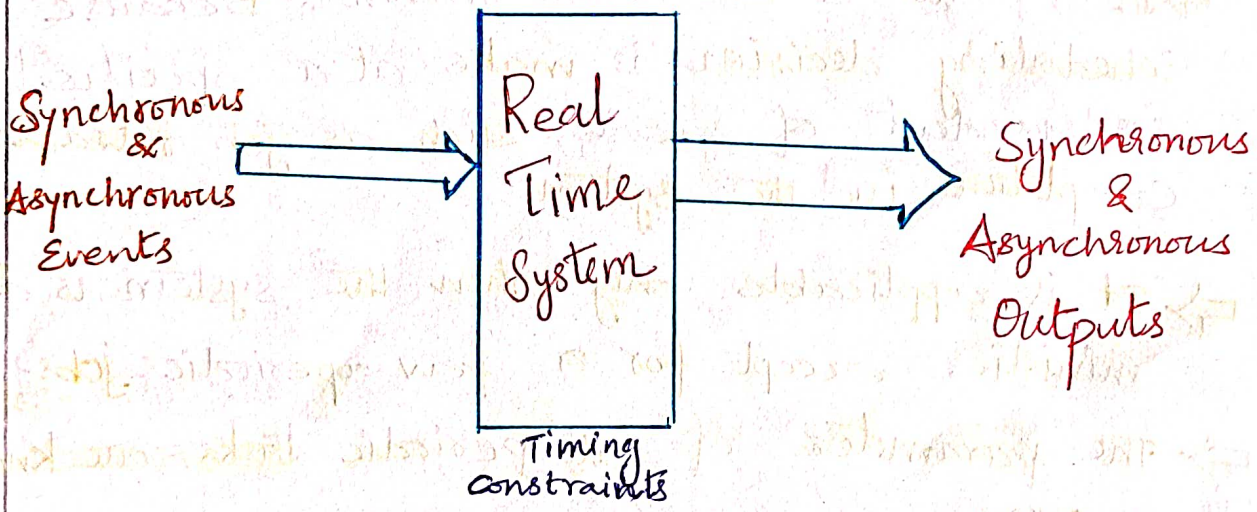
⇨ Dynamic; the scheduling decisions are based on dynamic parameters that may change during system evolution.

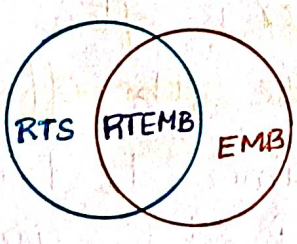
⇒ Off-line :- a scheduling algorithm is executed on the entire task set before actual task activation
- The schedule may be stored in a table and later executed by a dispatcher.

⇒ On-line : scheduling decisions are taken at runtime every time a new task enter the system or when a running task terminates.

⇒ Optimal : an algorithm is optimal if it minimizes some given cost function defined over the task set.

⇒ Heuristic : an algorithm is heuristic if it tends towards but does not guarantee to find the optimal schedule.





Railway monitoring & scheduling : RTS

Cell phone : EMB

Heart pacemaker : RTEMB

Approaches to Real-Time scheduling

* Clock-Driven Approach
* Weighted Round-Robine Approach
* Priority-Driven Approach
* Dynamic versus Static Systems
* Effective Release Times & Deadlines
* Earliest Deadline First (EDF) Algorithm
* Least Slack Time First (LST) Algorithm
* Validation of schedules

CLOCK-DRIVEN APPROACH

⇨ It is also called time-driven. Because each scheduling decision is make at a specific time, independent of events, such as job releases or completions in the system.

⇨ It is applicable only when the system is deterministic, except for a few aperiodics jobs.

⇨ The parameters of all periodic tasks are known a priori.

⇨ Therefore, the scheduler is often constructed by a static schedule of the jobs off-line.

- use a hardware timer to trigger the scheduling decision. The timer is set to expire periodically without intervention of the scheduler.

⇨ When the system is initialized, the scheduler selects & schedules the jobs that will execute until the next scheduling decision time & then blocks itself waiting for the expiration of the timer.

⇨ When the timer expires, the scheduler awakes & repeats these actions.

Round-robin approach

⇒ Generally used for scheduling time-shared applications.

⇒ The jobs are scheduled in a round-robin system so that every job joins a first-in-first-out (FIFO) queue

⇒ The job at head of the queue executes first at most one time slice. Otherwise it is pre-empted and placed at the end of the queue to wait for its next turn.

⇒ When there are 'n' ready jobs in the queue, each job gets one time slice in the total time (n), in every round.

⇒ Each job gets $\frac{1}{n}^{th}$ share of the processor when there are 'n' jobs ready for execution. Hence it is known as processor-sharing algorithm.

Weighted round-robin approach

⇒ The weighed round-robin algorithm has been used for scheduling real-time traffic in high speed switched networks.

⇒ Rather than giving all the read jobs equal shares of the processor, different jobs may be given different weights.

⇒ weight = the time slice allocated to the job.

⇒ A job with weight wt get same time slice in every round.

⇨ The length of the round equals the sum of the weights of all the ready jobs.

⇨ By adjusting the weights of jobs we can speed-up or slow-down the progress of each job towards its completion.

⇨ By giving each job a fraction of the processor, a round robin scheduler delays the completion of every job.

⇨ If it is used to schedule priority based jobs, the response time ~~is not suitable for scheduling~~ of a chain of jobs can be very large.

⇨ For this reason, the weighted round-robin approach is not suitable for scheduling such jobs.

⇨ But for Unix pipe, weighted round-robin scheduling may be a reasonable approach, since a job & its successors can execute concurrently in a pipelined fashion.

Example

- For example consider two sets of jobs

$J_1 = \{J_{1,1}, J_{1,2}\}$ & $J_2 = \{J_{2,1}, J_{2,2}\}$

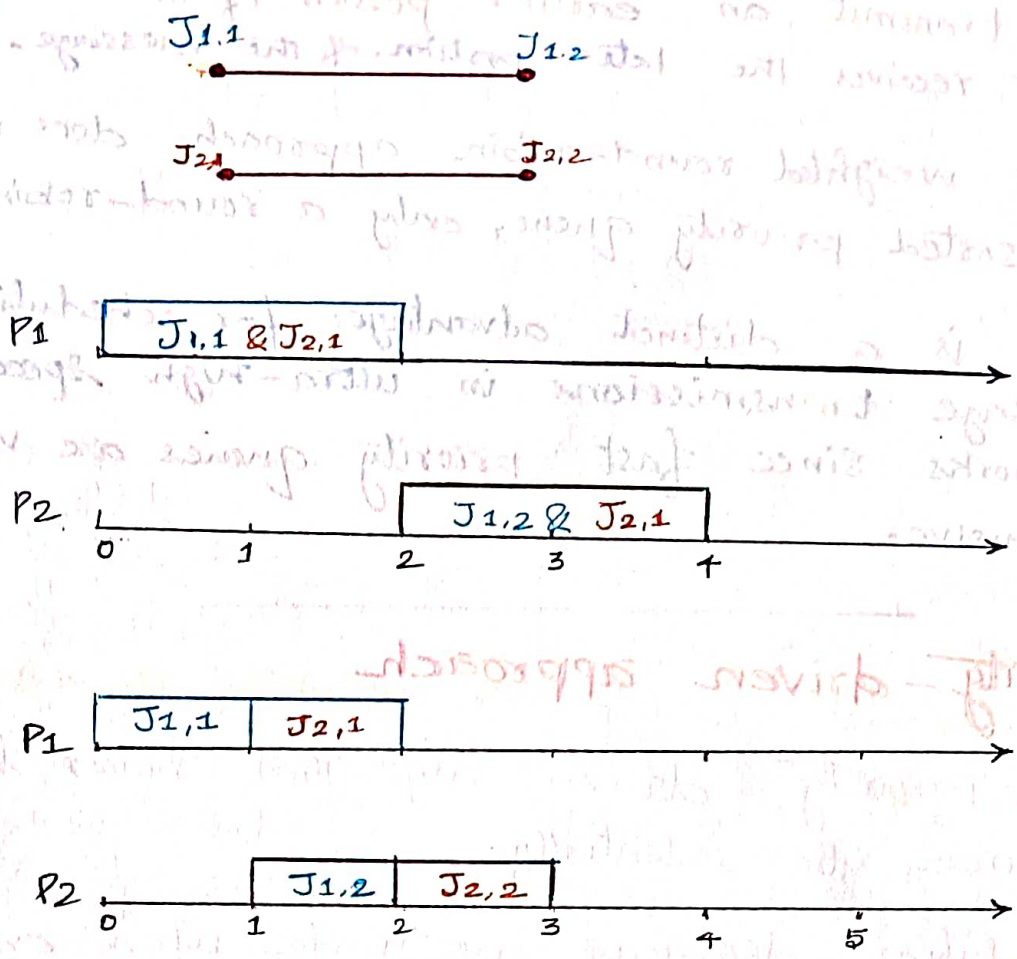
- The release times of all jobs are 0
- The execution times of all jobs are 1

$J_{1,1}$, $J_{2,1}$ executes on processor P_1

$J_{1,2}$, $J_{2,2}$ executes on processor P_2

- Suppose that $J_{1,1}$ is the predecessor of $J_{1,2}$ & $J_{2,1}$ is the predecessor of $J_{2,2}$

Example Illustration of Round-robin scheduling of precedence-constrained jobs.



* Figure (a) shows that both sets of complete approximately at time 4 if the jobs are scheduled in a weighted round-robin manner.
* In contast, if the jobs on each processor are executed one after another, one of the chains can complete at time 2, while the other can complete at time 3.
* Suppose that the result of the first job in each set is piped to the second job in the set. The latter can execute after each one or a few time slices of the former complete.
* Then the round-robin approache is better because both sets can complete a few time slices after time 2.

* In a switched network a downstream switch can begin to transmit an earlier portion of the message as soon as it receives the later portion of the message.

* The weighted round-robin approach does not require a sorted priority queue, only a round-robin queue.

* This is a distinct advantage for scheduling message transmissions in ultra-high speed networks since fast priority queues are very expensive.

Priority-driven approach

⇒ The priority driven algorithm never leave any resource idle intentially.

⇒ Scheduling decisions are made when events of jobs occur (Ex: releases & completion of jobs). Hence priority driven algorithms are event-driven one.

& other commonly used terms of this approache are greedy scheduling, list scheduling & work-conserving scheduling.

⇒ It is greedy because it takes best decisions.

⇒ So that it makes some jobs wait even when they are ready to execute and the resources they require are available.

⇒ priorities are assigned to jobs, and jobs ready for execution are placed in one or more queues.

⇒ At any scheduling decision time, the jobs with the highest priorites are scheduled and executed on the available processors.

⇨ Hence a priority-driven scheduling algorithm is defined largely by the list of priorities it assigns to jobs

⇨ The priority list & other rules such as whether preemption is allowed, defined the scheduling algorithm completely.

⇨ Most non-real-time scheduling algorithms are priority-driven.

Examples:

- FIFO (first-in-first-out) and LIFO (last-in-first-out) algorithms which assign priorities to jobs based on their release times.
- SETE (Shortest-execution-time-first) and LETF (Longest-execution-time-first) algorithms which assign priorities based on job execution times.

⇨ Because we can directly change the priorities of jobs, even round-robin scheduling can be thought of as priority driven.

⇨ The priority of the executing job has executed for the time slice.

⇨ The task graph shown here is a classical precedance graph: all its edges represent precedence constraints.

⇨ The number next to the name of each job is its execution time.

⇨ J_5 is released at time 4, all other jobs are released at time 0.

⇨ The two processors P_1 & P_2 are used to schedule the jobs & they are communicated through shared memory

⇨ Hence the costs of communication among jobs are negligible no matter where they are executed.

⇨ The schedulers of the processors keep one common priority queue of ready jobs.

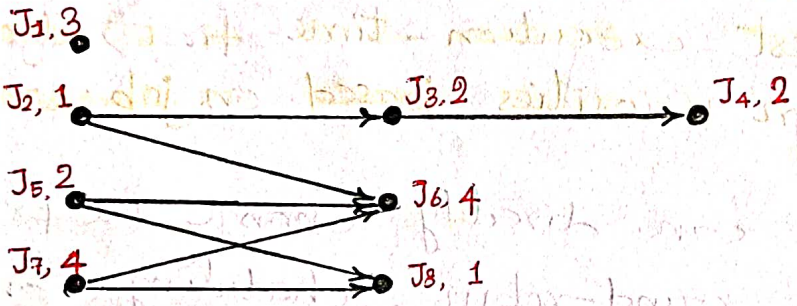
⇨ The priority list is given next to the graph.

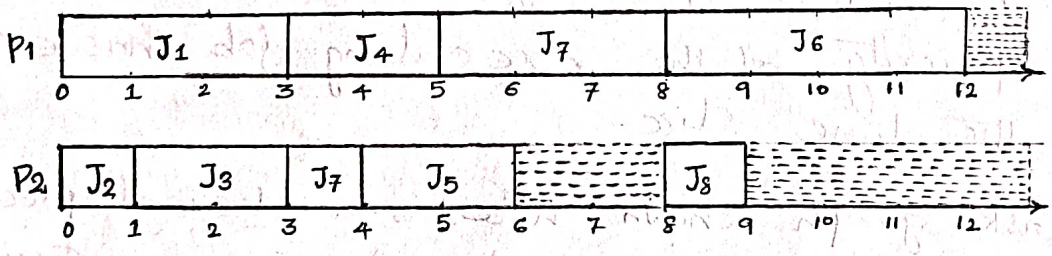
⇨ 'Ji' has a higher priority than Jk if $i < k$. All the jobs are are preemptable.

⇨ scheduling decisions are made whenever some jobs becomes ready for execution or some jobs completes.

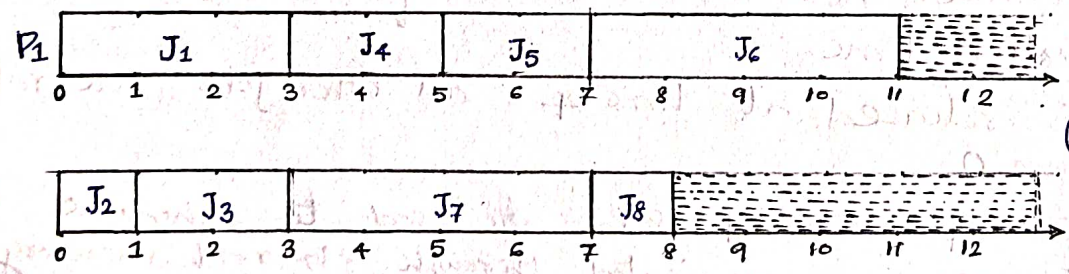
⇨ Figure (a) below shows the schedule of the jobs on the two processors generated by the priority-driven algorithm following this priority assignment.

⇨ At time 0, jobs J_1, J_2 & J_7 are ready for execution. Since J_1 & J_2 have higher priority than J_7, they are ahead of J_7 in the queue & hence are scheduled.





(a) Preemptive



(b) Non-preemptive

Example of priority-driven scheduling

⇨ They are the only jobs in the common priority queue at this time.

⇨ Since J_1 & J_2 have higher priorities than J_7, they are ahead of J_7 in the queue and hence they are scheduled.

⇨ The processors continue to execute the jobs scheduled on them except when the following decisions are made.

* At time 1, J_2 completes and hence J_3 becomes ready. J_3 is placed in the priority queue ahead of J_7 and is scheduled on P_2, the processor freed by J_2.
* At time 3, both J_1 & J_3 complete. J_5 is still not released. J_4 & J_7 are scheduled.
* At time 4, J_5 is released. Now there are there ready jobs. J_7 has the lowest priority among them. consequently, it is preempted. J_4 & J_5 have the processors.
* At time 5, J_4 completes. J_7 resumes on processor P_1
* At time 6, J_5 completes. Because J_7 is not yet completed, both J_6 & J_8 are not ready for execution. Consequently, processor P_2 becomes idle.
* J_7 finally completes at time 8. J_6 & J_8 can be scheduled.

⇨ Figure (b) shows a non-preemptive schedule according to the same priority assignment.

⇨ Before 4, this schedule is the same as the preemptive schedule.

⇨ However, at time 4 when J_5 is released, both processors are busy. It has to wait until J_4 completes (at time 5) before it can begin execution.

⇨ It turns out that for this system this postponement of the higher priority job benefits the set of jobs as a whole.

⇨ The entire set completes 1 unit of time earlier according to the non-preemptive schedule.

Estimating Program Run Time:

- A real-time program is defined as a program for which the correctness of operation depends on the logical results of the computation and the time at which the results are produced.
- In general, there are three types of programming **Sequential, Multi-tasking & Real-time.**

⇨ Estimating program run time depends on the following factors:

1. Source Code: Source code that is carefully tuned and optimized takes less time to execute.
2. Compiler: It maps source level code into a machine level program.
3. Machine architecture: Executing program may require much interaction between the processor and the memory and I/O devices.
4. Operating System: OS determines such issues as task scheduling and memory management. Both have major impact on memory management.

Analysis of Source Code:

⇨ Cosider the following code;

```
a := b x c;
b := d + e;
d := e - f;
```

⇨ This is straight line code. The total execution time is given by

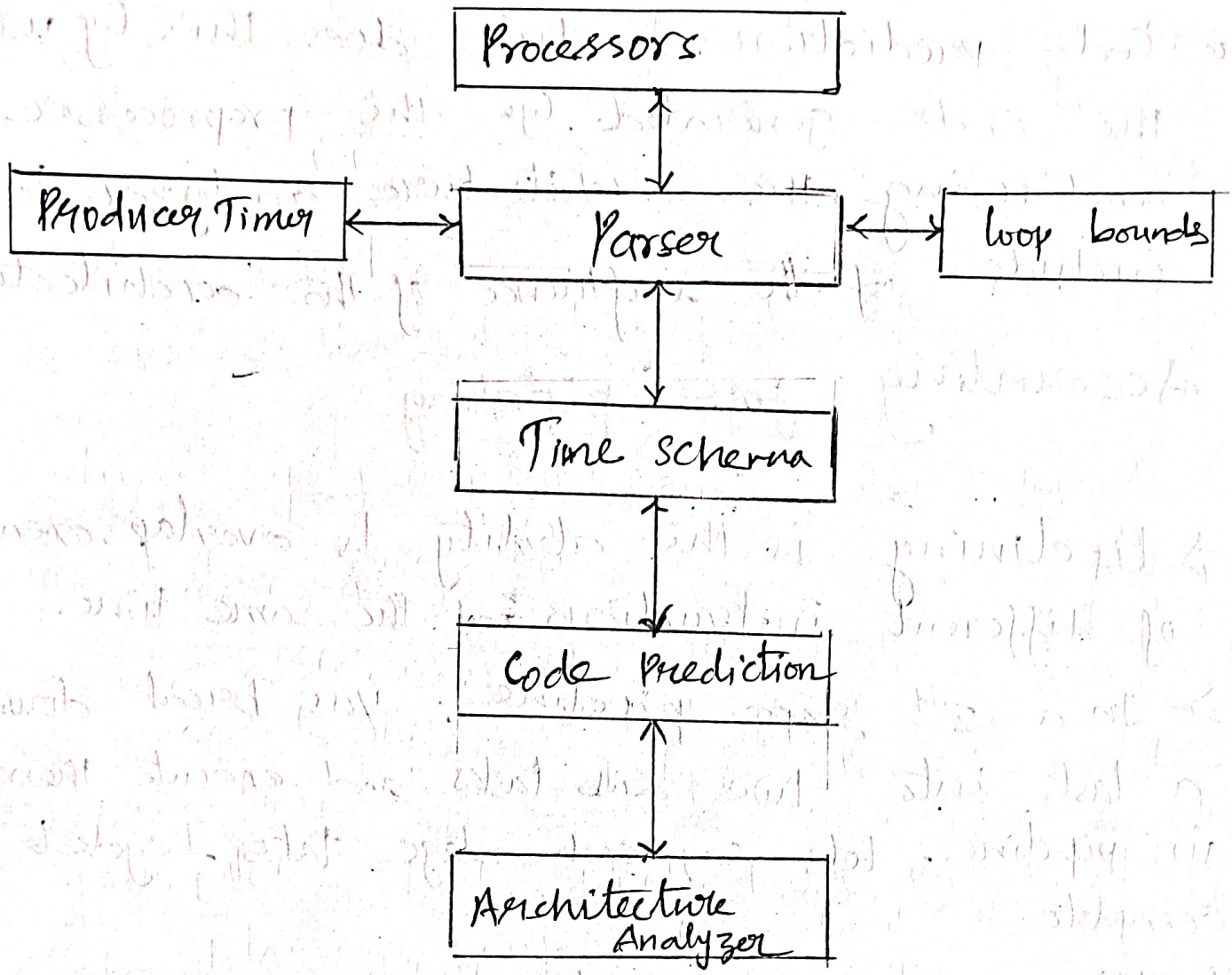
$$\sum_{i=1}^{3} Texec(Li)$$

Where Texec(Li) is the time needed to execute Li.

⇨ Execution time analysis is any structured method or tool applied to the problem of obtaining information about the execution time of a program or parts of a program.

⇨ The fundamental problem that a timing analysis has to deal with is the following: The execution time of a typical program is not a fixed constant, but rather varies with different probability of occurrence across a range of times.

⇨ The following schematic of timing estimation system.



Schematic of timing estimation system.

- Preprocessor produces compiled assembly language code. Parser analyze input source program.
- Producer timer maintain a table of procedures and their execution times.
- The loop bounds module obtaines bounds on the number of iterations for the various loops in the syste

- time schema is independent of the system, it depends only on the language.
- Code prediction module does this by using the code generated by the preprocessor and using the architecture analyzer to include the influence of the architecture.

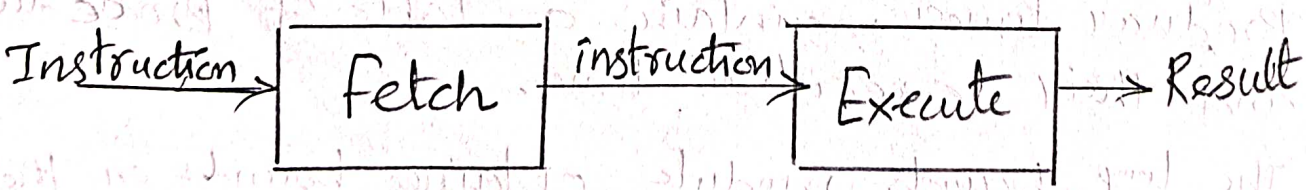
Accounting for pipeling

⇒ Pipelining is the ability to overlap execution of different instructions at the same time.

⇒ In a 2nd stage pipeline, you break down a task into two sub-tasks and execute them in pipeline. Lets say each stage takes 1 cycle to complete.

⇒ This means in a 2-stage pipeline, each task will take 2 cycles to complete.

⇒ An instruction has a no. of stages. The various stages can be worked on simultaneously through various blocks of production. This is a pipeline. This is also reffered as instruction pipeling



Pipeline of two independent stages

Task Assignment and scheduling

⇨ scheduling real time tasks on distributed and microprocessor systems consists of two subproblems.

1. Task allocation to the processor.

ⓐ The task assignment problem is concerned with how to partition a set of tasks and then how to assign these tasks to processors - task assignment can be: 1. static or 2. Dynamic.

ⓑ In the static allocation scheme, the allocations of tasks to nodes is permanent and does not change with time.

ⓒ In the dynamic task assignment, tasks are assigned to the nodes as they arise, different instances of tasks may be allocated to different nodes.

2. Scheduling of tasks on the individual processors: uniprocessor scheduling algorithms can be used for the task set allocated to a particular processor.

Static allocation algorithms:

⇨ The tasks are pre-allocated to processors.

⇨ No overhead incurs during run time since tasks are permanently assigned to processors at the system initialization time.

1. Next-Fit Algorithm for RMA
2. Bin Packing Algorithm for EDF
3. Utilization Balancing Algorithm.

⇨ Dynamic allocation Algorithms:

- In many applications tasks arrive sporadically at different nodes.
- The taks are assigned to processor as and when they arise.
- The dynamic approach incurs high run time overhead since the allocator component running at every node needs to keep track of the instantaneous load position at every other nodes.

1. Focussed Addressing and Bidding (FAB)
2. The Buddy Strategy Algorithm

Utilization - Balancing Algorithm

⇨ This algorithm attempts to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

⇨ Objective to balance processor utilization, and proceeds by allocating the tasks one by one and selecting the least utilized processor.

⇨ Maintains the tasks in a queue in increasing order of their utilizations.

⇨ It removes task one by one from the head of the queue and allocates them to the least utilized processor each time.

⇨ The objective of selecting the least utilized processor is to balance the utilization of different processors.

$$\frac{\sum_{i=1}^{P} (u_i^B)^2}{\sum_{i=1}^{P} (u_i^*)^2} \leq \frac{9}{8}$$

Where u_i^* = P_i's utilization under an optimal algorithm that minimizes $\sum \text{utilization}^2$

u_i^B = P_i's utilization under bet-fit algorithm.

Next-Fit Algorithm for RM-scheduling :

- This is a utilization-based allocation heuristic.
- The task set has the same properties as for the RM uniprocessor scheduling algorithms.
- 'M' is picked by user.
- corresponding to each task class is a set of processors that is only allocated to task of that class.

- It is possible to show that this approach uses no more than 'N' times the minimum possible number of processors.

- There are 'm' classes of tasks such that, each class of tasks are assigned to a corresponding set of processors.

  T_i belongs to class $j < m$ if

$$\frac{1}{2^{1+j} - 1} < \frac{e_i}{P_i} \leq 2^{i/j} - 1$$

  T_i belongs to class m otherwise.

Bin-Packing Assignment for EDF.

⇒ Same assumptions on tasks and processors as at Next-fit algorithm.

⇒ Problem : Schedule a set of periodic independent preemptible tasks on a multiprocessor system consisting of identical processors.

⇒ The task deadlines equal their periods and tasks require no other processors.

⇒ The task deadlines equal their periods and tasks require no other resources.

* Solution: EDF – scheduling on a processor and task set is EDF – schedulable if $U \leq 1$
* Assign tasks such that $U \leq 1$ foll all processors.
* The problem reduces to making task assignments to processors with the property that the sum of the utilizations of the tasks assigned to a processor does not exced one.

Focused Addressing and Bidding (FAB) Algorithm.

⇨ It uses dynamic allocations.
⇨ FAB is a simple algorithm that can be used as an online procedure for task sets consisting of both critical and non-critical real-time tasks.
⇨ Critical tasks must have sufficient time reserved for them so that they continue to execute successfully, even if they need their worst case executime time.
⇨ Non-critical tasks are either processed or not, depending on the systems ability to do so.

⇨ The guarantee can be based on the expected run time of the task rather than the worse-case run time. (non critical task)

⇨ THE UNDERLYING SYSTEM MODE IS:
When an noncritical task arrives at processor P_i, the processor checks to see if it has the resources and time to execute the task without missing any deadlines of the critical tasks or the previously guaranteed non critical tasks -- if yes, P_i accepts this new noncritical task and adds it to its list of tasks to be executed and reserves time for it.

⇨ The FAB ALGORITHM IS USED WHEN P_i determines that it does not have the resources or time to execute the task in this case, it tries to ship that task out to some other processor in the system.

⇨ Every processor maintains two tables called: status table and load table.

⇨ STASUS TABLE: indicates which tasks have been already committed to run including the set of critical tasks and any additional noncritical tasks that have been accepted at the different processors can be determined.

⇨ LOAD TABLE contains the latest load information of all other processors of the system, the surplus computing capacity available at the different processors can be determined.

* THE TIME AXIS is divided into windows, which are intervals of fixed duration, at the end of each window, each processor broadcasts to all other processors the fraction of computing power in the next window for which it has no committed tasks.

1. Every processor on receiving a broadcast from a node about the load position updates the system load table

2. Since the system is distributed, this information may never be completely up to date.

3. As the result, when a task arrives at a node, the node first checks whether the task can be processed locally, if yes, it updates its status table if not, it looks for a processor to offload the task.

* THE PROCESS OF OFF LOADING A TASK is based on the content of the system load table, an overloaded processor checks its surplus information and:

1. Select a processor (called focused processor) Ps that is believed to be the most likely to be able to successfully execute that task by its deadline.

2. The system load table information might be out of date

- The RFB (Requests For Bids) contains the vital statistics of the task.

3. The RFB asks any processor that can be successfully execute the task to send a bid to the focused Processor.

4. An RFB is only sent out if the sending processor P_S estimates that there will be enough time for timely response to it.

5. Specifically, two times t_{bid} & $t_{offload}$ are calculated → if $t_{bid} \leq t_{offload}$ then the RFB is sent out.

TIME CALCULATION BY FAB ALGORITHM.

1. t_{bid} = (Estimated time taken by RFB to reach its destination) + (The estimated time taken by the destination to respond with a bid) + (The estimated time taken to transmit the bid to the focused processor).

2. $t_{offload}$ = (Task deadline) − [(Current time) + (time to move the task) + (Task execution time)].

* If $t_{bid} \leq t_{offload}$; then RFB is sent out.

* when a processor P_t receives an RFB, it checks to see if it can meet the task requirements and still execute its already-scheduled tasks successfully.

Buddy Strategy:

* The buddy strategy tries to solve the same problem as the FAB algorith, soft real-time arrive at the various processors of a multiprocessor and, if an individual processor finds itself overloaded, it tries to off load some tasks onto less lightly loaded processors.

* The buddy strategy differs from the FAB algorithm in the manner in which the target processors are found.

STRETEGY:

⇨1 Each processor has 3 thresholds of loading: Under loaded (TU), fully loaded (TF), and over loaded (TV)

⇨2. The loading is determined by the no. of jobs awaiting service in the processor's queue. If the queue length is Q, the processor is said to be in:

a. State U (underloaded) if $Q \leq TU$;
b. State F (Fully loaded) if $TF < Q \leq TV$;
c. State V (overloaded) if $Q > TV$;

Fault Tolerance Techniques

- Fault-tolerance is defined informally as the ability of a system to deliver the expected service even in the presence of faults.
- A common misconception about real-time computing is that fault-tolerance is orthogonal to real-time requirements. It is often assumed that the availability and reliability requirements of a system can be addressed independent of its timing constrains.
- A real-time system may fail to function correctly either because of errors in its hardware and/or software or because of not responding in time to meet the timing requirement that are usually imposed by its "**environment**".
- Hardware fault is some physical defects that can cause a program to fail for a given set of inputs.
- An error is a manifestation of a fault. The fault latency is the duration between the onset of a fault and its manifestation as an error.

⊗ An error latency is the duration between when an error is produced and when it is either recognized as an error or causes the failure of the System.

⊗ Error recovery is the process by which the system attempts to recover from the effects of an error.

⊗ Recovery from an error is fundamental to fault tolerance.

⊗ Two main froms of recovery

1. Forward error recovery
2. Backward error recovery.

Forward error recovery

⇨ Forward recovery attempt to bring system to a new stable state from which it is possible to proceed

⇨ Forward error recovery continues from an erroneous state by making selective corrections to the system state.

⇨ This include making safe the controlled environment which may be hazardous or damaged because of the faiture.

⇨ It is system specific and depends on accurate predictions of the location and cause of errors. (i.e. damage assessment)

⇨ Example : Redundant pointers in data structures and the use of self-correcting codes such as Hamming codes.

Advantages forward-error recovery :

1. Less overhead

Disadvantages of forward recovery

1. In order to work, all potential errors need to be accounted for up-front.
2. Limited use
3. Cannot be used as general mechanism for error recovery.
4. Design specifically for a particular system.

Backward recovery :

- Most extensively used in distributed systems and generally safest. It can be incorporated into middleware layers.
- Backward recovery is complicated in the case of process, machine or network failure but no guarantee that same fault may occur again.

- It can not be applied to irreversible operations, e.g. ATM withdrawal.

Advantage backward – error recovery

1. Simple to implement.
2. Can be used as general recovery mechanism.
3. Capable of providing recovery from arbitrary damage.

Disadvantage of backward recovery.

1. Checkpointing can be very expensive – especially when errors are very rare.
2. Performance penalty.
3. No guarantee that fault does not occur again
4. Some components cannot be recovered.

Causes of failure:

- There are three causes of failure,
  1. Errors in the specification or design.
  2. Defects in the components.
  3. Environmental effects.

- Mistake in the specification and design are very difficult to guard against. Many hardward failures and all software failures occurs such mistake.
- If the specification is wrong, everything that processeds fro it, design and implementation, likely to be unsatisfactory.

Fault Types

- Fault are classified as temporal behaviours and output behaviours.

1. Temporal behaviours classification.

- Fault are of three types: permanent, intermittent & transient.

- **Transient faults:** These occur once and then disappear. For example, a network message transmission times out but works fine when attempted a second time.
- **Intermittent faults:** Those are the most annoying of component faults. This fault is characterized by a fault occuring then vanishing again then occuring. As example of this kind of fault is a loose connection.

Permanent faults:

- This fault is persistent: It continues to exist until the faculty components is repaired or replaced. Examples of this fault are disk head crashes, software bugs, and burnt-out hardware.

2. output behaviours classification.

- Malicious faults: Inconsistent output.
- Nonmalicious fault: Consistent output errors.
- Fail stop: Responds to up to a certain maximum numbers of failures by simply stoping, rather than putting out incorrect outputs. The component simply stoping, rather than putting out incorrect outputs. The component simply stops working. For instance, a hard disk which refuse to read or write.

Fail safe: Its failure mode is biased so that the application process does not suffer catastrophe upon failure. A component under too much load is likely to fail. A fail safe system, on detecting a large amount of load. processes such request slower to avoid failure

3. Independence and Correlation:

- Components failure may be independent and correlated.
- Independent : A failure is said to be independent it does not directly or indirectly cause another failure.
- Correlated : If the failure is said to be correlated if they are related in some way.

Reliability Evaluation:

⇨ Reliability refers to the property that a system can run continuously without failure. In contrast availability is defined interms of a time interval instead of an instant in time.

⇨ A highly reliable system is one that will most likely continue to work without interruption during a relatively long period of time. This is a subtle but important difference when compared to availability.

⇨ If a system goes down on average for one, seemingly random millisecond every hour, it has an availability of more than 99.9999 percent, but it still unreliable.

* Two methods are used for finding device failure rates: Collecting field data or life cycle testing in the laboratory.

* The most common accelerate is temperature. The higher the temperature the greater the failure rate. The acceleration factor is given by the following equation.

$$R(T) = Ae^{-Fa/kT}$$

where A is a constant, Ea is the activation energy and depends largely on the logic family used, k is the Boltzmann constant.

* To measure how quickly an error can propagate, we use fault injection. This is best done on a prototype. special purpose hardware is used to simulate a fault on a selected line. The status of related lines is monitored using logic analyzers to determine how far the error propagates and how quickly. If a prototype is not available, a software simulation can be substituted.

to the drifts of the clocks and to the uncertainty of the delay that the message carrying the time takes to travel along the distributed network.

* The drift of the clock usually is more influent than the message delays in the case the time-carrying communication is infrequent: with the decrease of the frequency of communication, the uncertainty due to the clock drift increases, while the uncertainty due to the message delays remains constant.

* A trival solution may be the decrease of the synchronization interval, but any clock synchronization schemes require the time information obtained through the communication to be processed.

UNIT-V. - Processing and Operating Systems

Introduction:

⇒ Microprocessor execute simple user applica-tion program. But some application requires large number of line of code. Writing and executing large program on microprocessor is complex task.

⇒ To study the writing complex program for microprocessor, we must understand concept of process and OS.

⇒ The process difines the state of an executing program, whicke the OS provides the mechanism for switching execution between the processes.

⇒ These two mechanisms together let us build applications with more complex functionality and much greater flexibility to saying satisfy timing requirements.

Multiple Tasks and Multiple Processes:

⇒ Task are units of sequential code implementing the system actions and executed concurrently by an OS.

⇒ Real time systems requires that tasks be performed within a particular time fuction. Task is related to the performance of the real time systems.

⇒ A task, also called a thread, is a simple program that thinks it has the CPU all to itself. The design process for a real-time application involves splitting the work to be done tasks responsible for a real-time application ie. a portion of the problem.

⇒ Each task is assigned a priority, its own set of CPU registers and its own stack area.

⇒ In the specified time constraint, system must produce its correct output. If system fail to meet the specified output, then the system is fail or quality decreases.

⇒ Real time systems are used for space flights, air traffic control, high speed aircraft, telephone switching electricity distribution, industrial process etc.

⇒ Real time system must be 100% responsive 100% of the time. Response time is measured in fab fractions of second, but this is an ideal not often achieved in the field.

⇒ Real time database is updated continuously. In aircraft example, flight data is continuously changing so it is necessary to update. It include speed, direction, location, height etc.

⇒ A process is a sequential program in exection. Terms like job and task are also used to denote a process.

⇒ A process is a dynamic entity that executes a program on a particular set of data. Multiple processes may be associated with one program.

⇒ Task is a single instance of an execlable program.

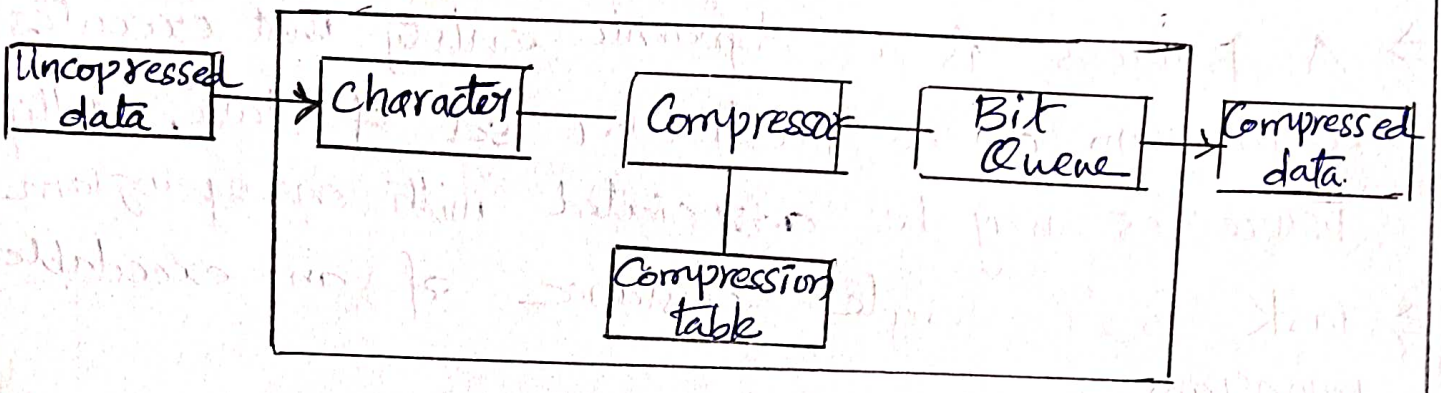
⇒ In a multiprogramming environment, usually more programs to be executed than could possibly be run at one time. In CPU scheduling it switches from one process to another process. CPU resource management is commonly known as scheduling.

⇒ Objective of the multiprogramming is to increases the CPU utilization. CPU scheduling is one kind of fundemental operating system functions.

⇒ Each process has an execution state which indicate what process is currently doing. The process descriptor is the basic data structure used to represent the specific state for each process.

⇒ Input and output of the compressor box is serial ports. It takes uncompressed data and processes it. Output of the box is compressed data. Given data is compressed using predefined compression table. Modem is used such type of box..

⇒



⇒ The program's need to receive and send data at different rates. It is an example of rate control problems. It uses asynchronous input. You can provide a button for compressed mode and uncompressed mode.

⇒ when user press uncompressed mode, the input data is passed through unchanged.

⇒ Sampling the button's state too slowly can cause the machine to miss a button depression entirely, but sampling it too frequently and duplicating a data value can cause the machine to incorrectly compress data.

⇒ This problem is solved by maintaining counter.

Multirate Systems:

⇒ More complicated control systems have multiple sensors and actuators and must support control loops of different rates. Multirate embedded computing systems includes automobile engines, printers and cell phones.

⇒ Tasks may be synchronous or asynchronous. Synchronous tasks may recur at different rates. Processors run at different rates based on computational needs of the tasks.

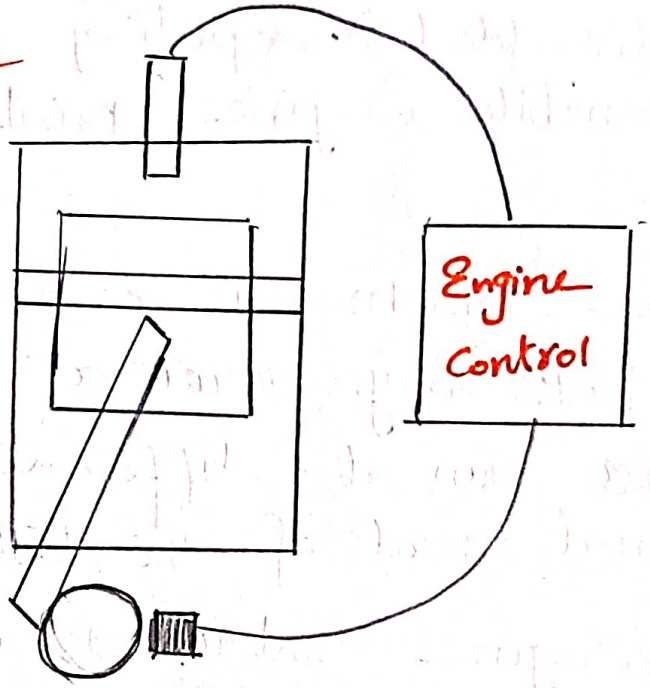
⇒ Automotive engine control is an example of multirate system. Tasks in automotive engine control are spark control, crankshaft sensing fuel/air mixture, oxygen sensors and Kalman filter.

⇒ The figure shows automotive engine control.

⇒ The spark plug must be fired at a certain point in the combustion cycle, but to obtain better performance, the phase relationship between the piston's movement and the spark should change as a function of engine speed.

⇒ Using a microcontroller that senses the engine crankshaft position allows the spark timing to vary with engine speed.

Engine Control



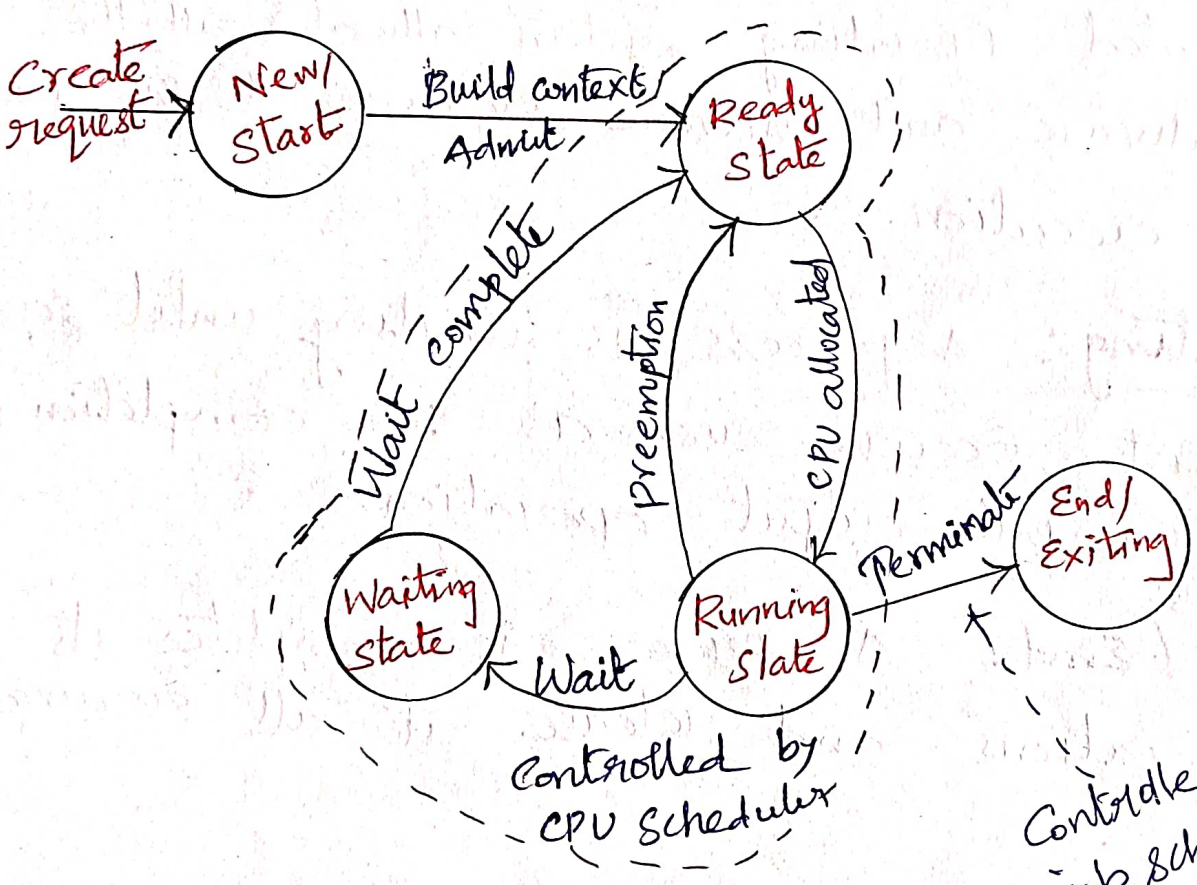
* Automobile engine controllers use additional sensors, including the gas pedal position and an oxygen sensor used to control emissions. They also use a multimode control scheme.

⇒ The longer number of sensors and modes increase the number of discrete tasks that must be performed.

Process State and Scheduling:

⇒ Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structures used to represent the specific state for each process.

⇒ A state diagram is composed of a set of state and transition between states.



Process State Diagram

⇨ State diagram is used by process manager to determine the type of service to provide to the process. The process states are as follows:
" New, ready, **running, waiting and end.**

New: Operating system creates new process by using fork() system call. These process are newly created process and resources are not allocated.

Ready: The process is competing for the CPU. Process reaches to the head of the list (queue)

Running: The process that a currently being executed. Operating system allocates all the hardware and software resources to the process for execution.

Waiting: A process is waiting until some event occurs such as the completion of an input-output operation.

Exit / End: A process is completes its operations and releases it all resources.

⇒ Five types of inter-process communication as follows:

1. shared memory permits processes to communicate by simply reading and writing to a specified memory location.
2. Mapped memory is similar to shared memory except that it is associated with a file in the file system.
3. Pipes permit sequential communication from one process to a related process.
4. FIFOs are similar to pipe, except that unrelated processes can communicate because the pipe is given a name in the file system.
5. Sockets supports communication between unrelated processes even on different computers.

Purposes of IPC:

1. Data Transfer: one process may wish to send data to another process.
2. Sharing Data: Multiple processes may wish to operate on shared data, such that if a process modifies the data, that change will be immediately visible to other processes sharing it.

INTERPROCESS COMMUNICATION MACHANISM:

⇒ Exchange of data between two or more separate, independent processes / threads is possible using IPC, Operating Systems provides facilities / resources for Inter-Process Communication (IPC), such as **message queues, semaphores, and shared memory.**

⇒ A complex programming environment often uses multiple cooperating ~~progress~~ processes to perform related operations. These processes must communicate with each other and share resources and information. The Kernel must provide mechanisms that make this possible. These mechanisms are collectively referred to as **interprocess communication.**

⇒ Distributed computing systems make use of these facilities / resources to provide Application Programming Interface (API) which allows IPC to be programmed at a higher level of abstraction (e.g. send & receive)

Five types of inter-process communication are as follows:

1. Shared memory permits processes to communi-cate by simply reading and writing to a specified memory location.
2. Mapped memory is similar to shared memory, except that it is associated with a file in the file system.
3. Pipes permit sequential communication from one process to a related process.
4. FIFOs are similar to pipes, except that unrelated processes can communicate because the pipe is given a name in the file system.
5. Sockets support communication between unrelated processes even on different computers.

Purposes of IPC:

1. Data transfer: One process may wish to send data to another process.

2. Sharing data: Multiple processes may wish to operate on shared data, such that to operate if a process modifies that data, that change will be immediately visible to other processes sharing it.

3. Event modification: A process may wish to notify another process or set of processes that some event has occurred.

4. Resource sharing: The Kernel provides default semantics for resource allocation: they are not suitable for all application.

5. Process control: A process such as a debugger may wish to assume complete control over the execution of another process.

IPC has two forms: IPC on same host
IPC on different hosts.

IPC is used for 2 functions:

1. Synchronization
2. Message passing.

Features of Message Passing:

1. **Simplicity**: It should be possible to communicate with old and new applications.

2. **Uniform semantics**: Message passing is used for two types of IPC.
   a. **local communication**: Communicating processes are on the same node.
   b. **Remote communication**: Communicating processes are on the different nodes.

3. **Efficiency**: IPC become so expensive if message passing system is not effective

4. **Reliability**: Distributed systems are prone to different catastrophic event such as node crashes or physical link failure. loss of message because of communication link fails. To handle the loss message, we required acknowledgement and retransmission policy.

5. **Correctness**: It is a feature to IPC provides for group communication. Issues related to correctness to a group of receivers will be delivered to either all of them or none of them

(i) Automicity : Every message sent to a group of receivers will be delivered to either all of them or none of them.

(ii) Ordered delivery : Message arrive to all receivers in an order acceptable to the application.

(iii) Survivability : Message will be correctly delivered despite partial failures of processes, machine or communication links.

6) Security :

Message passing system must provide a secure end to end communication.

7) Portability : Message passing system should itself be portable.

IPC Message passing :

⇒ Message passing system requires the synchronization and communication between the two processes

⇒ The actual function of message passing provided in the form of a pair of primitives.

(a) send (destination_name, message)

(b) Receive (source_name, message)

Design characteristics of message system for IPC

1) Synchronization between the process.
2) Addressing
3) Format of the message
4) Queueing discipline.

Message structure

Actual data or pointer to the data	Structural information: Number of bytes/ elements	Structural information: Type	Sequence number or message ID	Addresses: Receiving Process address	Addresses: Sending Process address
← Variable Size collection of typed data			← Fixed-length header.		→

⇒ The header block of a message may have the following elements:

1. Address: A set of character that uniquely identify both the sender and receiver.
2. Sequence number: It is the message identifier to identify duplicate and lost message in case of system failure.
3. Structural information: It has two part.
   1) The type part that specifies whether the data to be sent the receiver is included within the message
   2) ~~length of the Variable Size message.~~

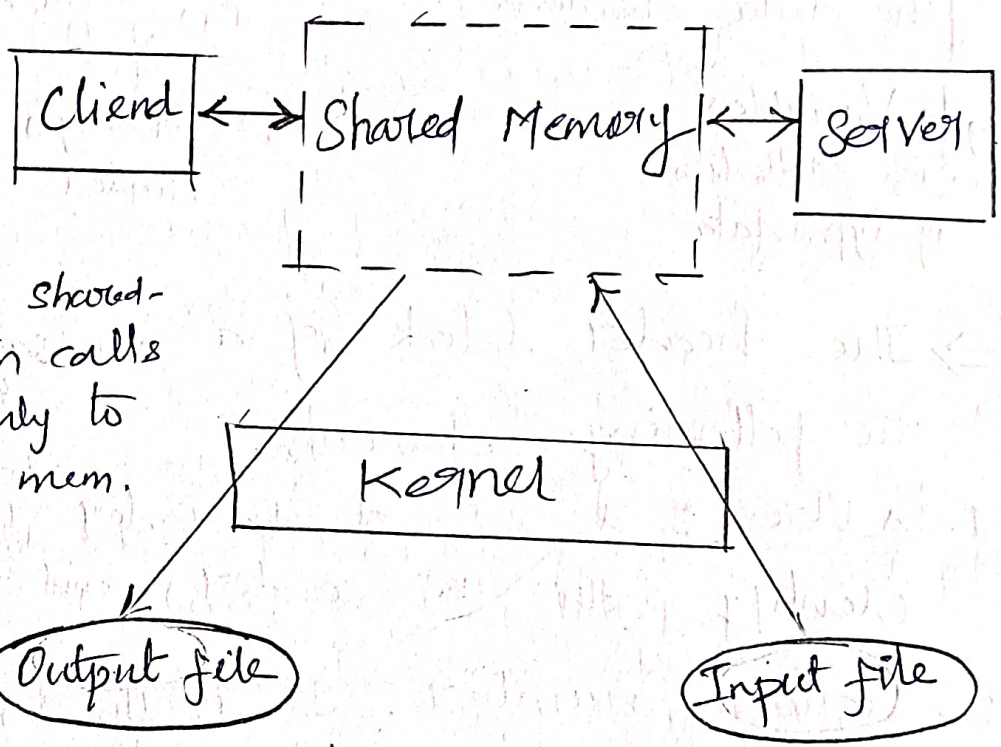
Shared Memory:

⇒ A region of memory that is shared by co-operating processes is established. Processes can be then exchanged information by reading and writing data to the shared region.

⇒ Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer.

⇒ Shared memory is faster than message passing

⇒ In contrast, in shared-memory, the system calls are required only to establish, shared mem. regions.



Advantages:

1. Good for sharing large amount of data.
2. Very fast.

Limitations:

⇒ No synchronization provided - application must creat their own.

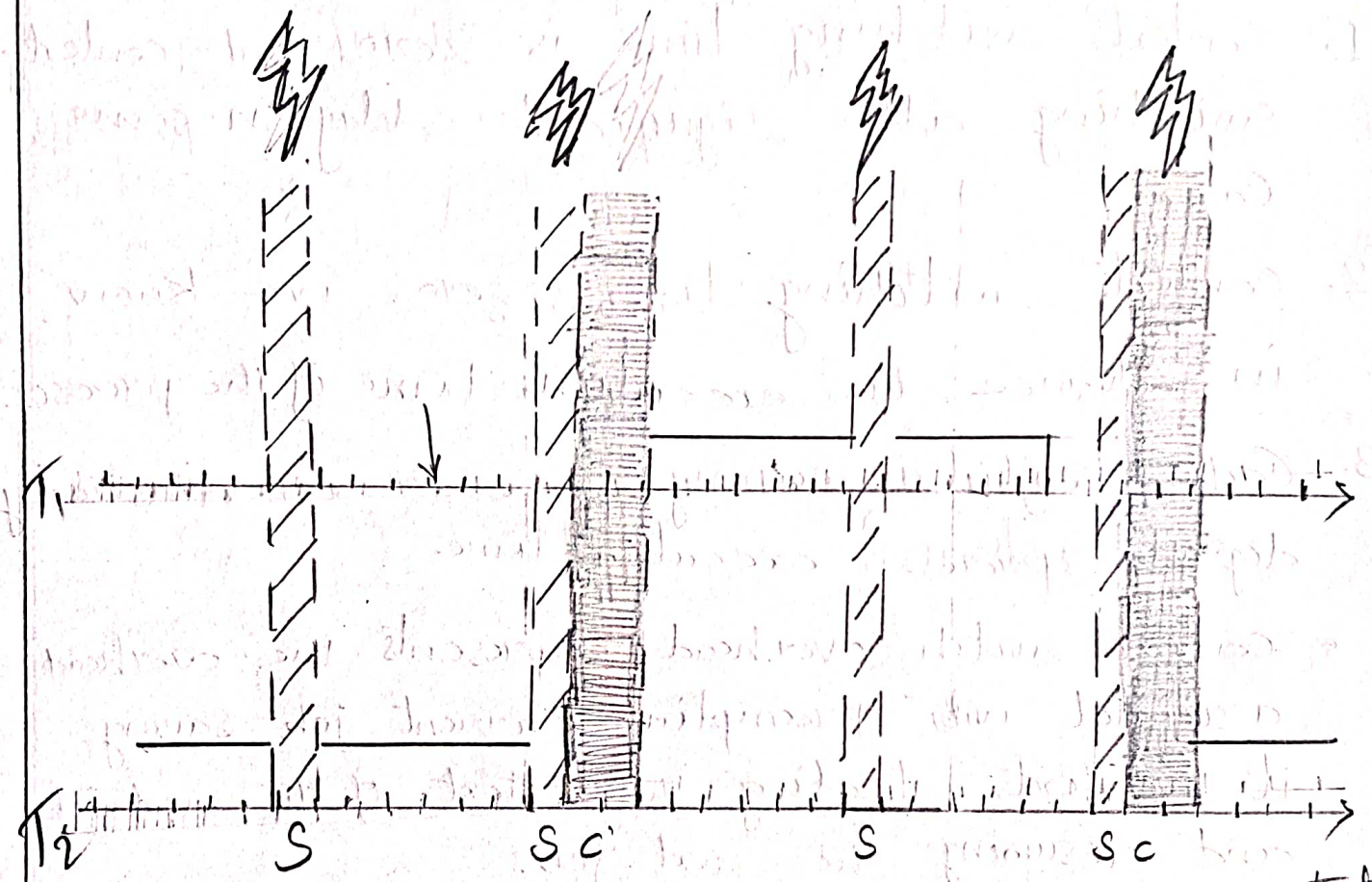
Evaluating Operating System Performance.

⇨ Scheduling algorithm is not used for calculating performance of the real system running processes. Assumption about analysis of scheduling policy are as follows:

1. Context switching time is zero, But context-switching adds significant delay in some cases.
2. context switching time is zero, we know in advance, the execution time of the processes.
3. Cache conflicts among processes can drastically degrade process execution time.

- context switch overhead represents the overhead associated with preempting current job, saving its context, loading the cotext of the next job and resuming the next job.
- The context switch into task (T_i) is performed at the kernel priority level, but only when task (T_i) is ready to execute.
- Figure next shows the scheduling the context switch overheads in a time-triggered preemptive system.

- Consider two tasks T_1 with $C_1 = 10$ and T_2 with $C_2 > 17$. clock interrupt handler is invoked with period $T_0 = 8$ and computation time $C_0 = 1$.
- The execution of the Task T_2 is regularly preempted by the clock handler. Somewhere before the second clock tick T_{II} is released



Clock tick ↓ Event release ▨ Scheduling overhead ▨ Context Switch overhead

Scheduling (S) and context switch (C) overheads in a time-triggered preemptive systems.

⇒ After executing for 10 time units T_1 is finished and released the processor until the next clock tick. During the next scheduler invocation the process is switched back to T_2.

Power optimization strategies for Processes.

⇨ RTOS and system architecture can use static and dynamic power management mechanisms to help manage the system's power consumption.

⇨ Power management policy in general examines the state of the system to determine when to take actions.

⇨ Power management policy is a strategy for determining when to perform certain power mangement operations.

⇨ Dynamic Power management (DPM) is a design methodology for dynamically reconfiguring system to provide the requesting services & performance levels with a minimum number of active components or a minimum load on such components.

⇨ Going into a low-power mode takes time; generally, the more that is shut off, the longer the delay incurred during restart. Because power-down and power-up are not free, modes should be changed carefully.

⇨ In most real-time operating systems, a context switch requires only a few hundred instruction, which with only slightly more overhead for a simple real-time scheduler like RMS. When the overhead time is very small relative to the task periods, then the zero-time context switch assumption is often a reasonable approximation.

⇨ Process execution time is not contant. Extra CPU time can be good. Extra CPU time can also be bad because next process runs earlier, causing new preemption.

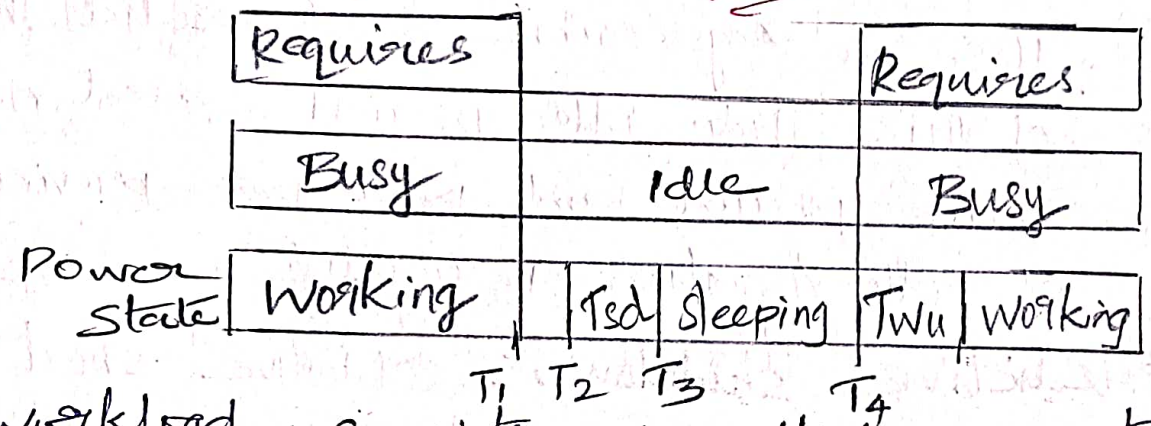
⇨ Processes can cause additional caching problems. Even if individual processes are well-behaved, processes may interfere with each other.

⇨ Worst-execution time with bad behavior is usually much more worse than execution time with good cache behavior.

⇨ Determining when to switch into and out of a power-up mode requires an analysis of the overall system activity.

⇨ System-level power management saves power of subsystems. Examples of devices include I/O controllers, hard disk drives, network interface cards and displays. Shutting down hard disks and displays is the most widely adopted system-level power management on PCs.

Power management concept



⇨ A workload consists of multiple requests. For hard disks, requests are read or write commands; for network cards, requests are packets to send or to be received.

⇨ When there are requests, the device is busy; otherwise, it is idle. Here, the device is idle between T1 & T4. When the device is idle, it can be shut down to enter a low-power sleeping state.

⇨ The device is shutdown at T2 and woken up at T4 when requests arrive again. Changing power states takes time: Tsd and Twu are the shutdown and wake-up delays.

⇨ In the example of hard disks and displays, it takes several seconds to wake-up these devices. Furthermore, waking up a sleeping device may take extra energy.

Predictive shutdown: shutdown as soon as a new idle period starts based on history. Avoid wasting energy before reaching timeout threshold.

Predictive wakeup: wakeup when predicted idle time expires, even if no new activity has occurred.

- Use a regression of predict the length of this idle period based on preceding active period and previous n pairs of idle / active periods.

Predictive shutdown: Observe short active period tends to be followed by long idle period in some applications. If the preceding active period is less than as threshold, the idle period is predicted to be longer than break-even time.

Advanced Configuration and Power Interface

⇨ Advanced Configuration and Power Interface (ACPI) is an open industry standard for power management services. It is designed to be compatible with a wide variety of OSs.

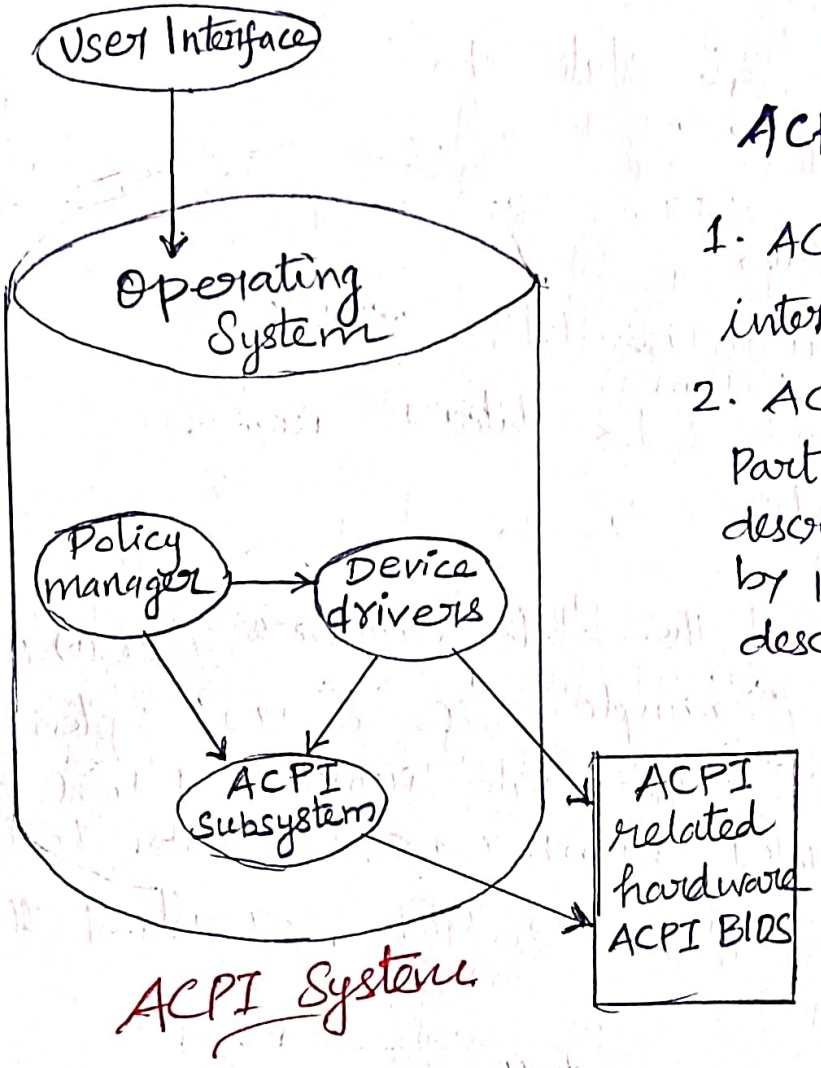
⇨ Power management states and required functionality are defined for multiple levels of the system.

1. Global View : Gx states
2. System : Sx states
3. Processor : Cx states
4. PCI / PCI-X bus : Bx states
5. PCI / express links : Lx states
6. Devices : Dx states

- General trend of the state numbers : (Zero) 0 is the active state (example : G0, S0, D0). System is available to user. The state number 1 to n are sleep states. Higher number corresponds to lower power. User perception is OFF for all of these sleep state.

- ACPI functions are as follows :
  1. System power management
  2. Device power management
  3. Processor power management
  4. Plug and play
  5. System events
  6. Battery managements.
  7. Thermal management
  8. Embedded controller.

⇨ The following figure ACPI system was designed to enable the operating system to set-up and control the individual hardware components.



ACPI run-time components

1. ACPI tables: Describe the interface to the hardware.
2. ACPI registers: The constrained part of the hardware interface, described (at least in location) by the ACPI subsystem description tables.
3. ACPI BIOS: ACPI system firmware. The firmware boots the machine and compatible with ACPI spec.

Global System states

- G0: Working (System operational)
- G1: Sleeping - no user threads System looks off
- G2/S5: Soft off
- G3: Mechanical off

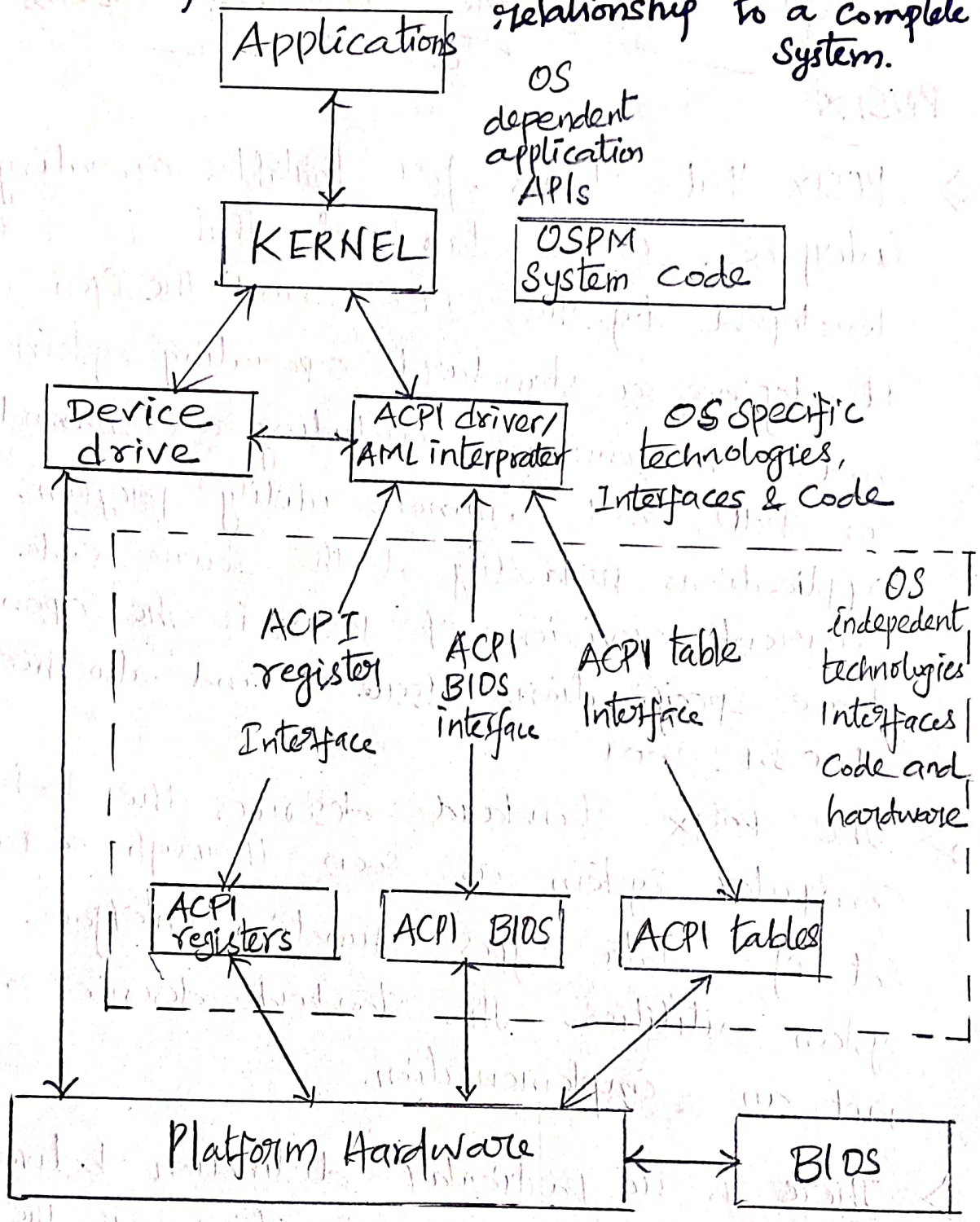
Processor power states:

- C0: Full power, instruction execute
- C1: Processor stopped
- C2: Processor stopped, less power than C1
- C3: Processor stopped cache ignore snoops

Device power states

D3: off - Power off to device
D2: Less power than D1
D1: Less power than D0
D0: Fully-on.

Advanced Configuration & Power Interface & its relationship to a complete System.



Sleeping States:

S0: System working

S1: low latency sleeping State Processor context maintained

S2: low latency sleeping sleeping State Processor context not maintained.

S3: low latency sleeping state DRAM still maintained

S4: lowest power longest wake-up DRAM not maintained

S5: Soft off state

Example Real Time Operating Systems.

POSIX

⇨ POSIX that stands for Portable Operating System interface is a standard that is being jointly developed by the IEEE and the Open Group. It defines a standard operating system interface and environment, including a command interpreter (or shell), and common utility program to support applications portability at the source code level. The current revision of POSIX is The Open Group Base Specifications Issue 6 and also the IEEE std. 1003.1 : 2001

⇨ The POSIX standard describes the behavior of a computer system as seen through a particular set of data types, function interface and system utilities. The standard describes an interface not an implementation.

⇨ There is no particular distinction between system functions and library functions, as the C99 libraries are included with the POSIX libraries.

⇨ The basic goal was to promote portability of applications programs across UNIX system environments by developing a clear, consistent and unambiguous standard for the interface specification of a portable

operating system based on the UNIX system documentation. The Standard codifies the common, existing definition of the UNIX system.

- The standard is composed by four major components:

1. **Base Definitions**: This includes general terms, concept and interfaces common to entire Standard.

2. **System Interfaces**: This comprises the definitions for system service functions for the C programming language, function and portability issues, error handling and recovery.

3. **Shell and Utilities**: It contains the definitions for a standard source code-level interface to command interpretation services.

4. **Rationale**: It contains information that does not fit well into the rest of the document structure.

Posix.1: IEEE 1003.1-1990, adapted by ISO. As ISO/IEC 9945:1:1990 standard gives standard for base operating system API.

Posix.1b: IEEE 1003.4:1993 Gives standard APIs for real time OS interface including inter-process comm.

Posix.1c: Specifies multi thread programming interface

⇨ To ensure program confirms to POSIX.1 standard user should define _POSIX_SOURCE as

1. #define _POSIX_SOURCE OR
2. Specify -D_POSIX_SOURCE to a C++ compiler.

Windows CE:

⇨ Windows CE is based on windows 95 with the usual interface, adapted for small devices. The development for this operating system under the code name Pegasus began in 1995. Specially designed for micro-computers, the abbreviation **CE stands informal for "Compact Edition"**.

⇨ The first version of windows CE requires as a minimum 4MB of ROM, 2MB of RAM and a processor of the SuperH3, MIPS 3000 or MIPS4000 architecture.

⇨ **Windows CE 2.0** came in October 1997 with the first devices manufacturers itself. TrueType fonts improving now the appearance by the device manufacturers with a display of 640×480 pixel full VGA resolution and 24-bit color depth.

⇨ The manageable memory can now be up to 4MB. The software "Handled PC Explore" is renamed to ActiveSync.

⇒ The update **windows CE 2.10** in July 1998 allows the use of TCP/IP and the file system FAT32. With the modular file wrapper can be incorporated up to 256 different file systems. The RAM can now be up to 16MB.

⇒ The new command line processor allows in this release for the first time the use of commands without a graphical user interface. An infrared port and USB controller increases the scope.

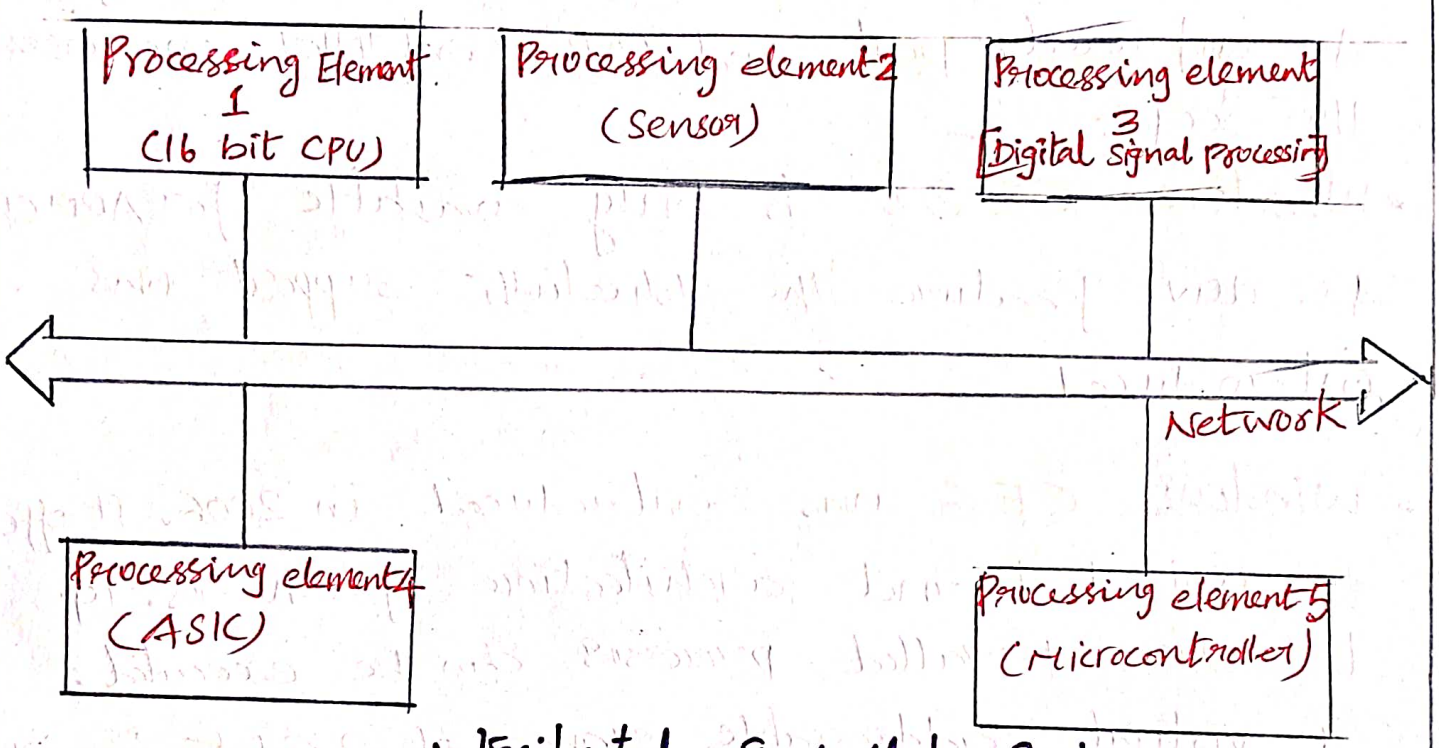
- **Windows CE 3.0** is only available for ARM CPUs. As new feature the Bluetooth support was introduced.

- **windows CE6** was introduced in 2006. It offers a revised kernel architecture of the os. Up to 32,000 parallel processes can be executed. A virtual addressable range of 2 Gbyte is possible for every process

⇒ The multimedia capabilities have been expanded and now support HD-DVD, DVD-UDF 2.5, multichannel audio and much more. The Compatibility to existing Windows CE applications and drivers are kept.

Distributed Embedded Systems.

- In a distributed embedded system, several Processing Elements (PEs) are connected by a network that allows them to communicate. below figure shows an example of a distributed embedded system.



Distributed Embedded System.

- Processing element may includes DSP, CPU or microcontroller. Nonprogrammable unit such as the ASICs is also to implement as PE.
- By using this entire processing element, it forms bus topology. It is also possible to form other topology also. It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them.