EC352 - Embedded and RTOS
Embedded system

02/01/2023

- RTOS - Real Time Operating Systems

- An Embedded system is a device which is a combination of computer hardware and software designed for a specific function.
- Systems can be programmable or have a fixed functionality.

03/01/23

→ Embedded System & General Purpose Computing (Laptop) machine

- Firmware - a software basically used in ES

E.S ≈ GPC ☆(2M)

→ for a specified application
→ Have a specified hardware and firmware (software)
→ Pre-programmed (cannot update)
→ Time critical
Ex: Washing Machine, Fridge, ATM
→ Power management.

→ Generic processor
Generic O.S
→ We can change the OS (Operating system)
→ Not time critical
Ex: Laptop, Personal Computer, Mobile phones
→ Performance oriented.

• Related to Cyber Security : Embedded System

Ex: Fire alarm system, Health monitor, ECG

• ES applications even in Space exploration, Military-US (during WW-II)
Telecom, IT revolution.

At Initial stage: Telephone - keypad, Radio
Embedded systems → (Electromechanical device)

Categorization :

① Based on generation
- 1G – 8 bit MP, 8 bit MC
- 2G – 16 bit MP, MC
- 3G – 32 bit MP, 16 bit MC
- 4G – SOC (system on chip)
- 5G –

(In macbooks, M1 chip, M2 chip)

② Based on Complexity and Performance requirements

③ Based on Deterministic behaviour
- R.Ts (Real Time systems) — Soft ES
- R.Ts (Real Time systems) — Hard ES

④ Triggering – Event or Time based

Ex: 1 G(ES) – Telephone keypads, (Stepper motor)

2G ES – Data Aquasition systems, SCADA

3G ES – DSP, ASICs (application specified Integrated Circuits)

4G ES – Pipelining

- Quadrotor – Drone 

Other Examples of ES :

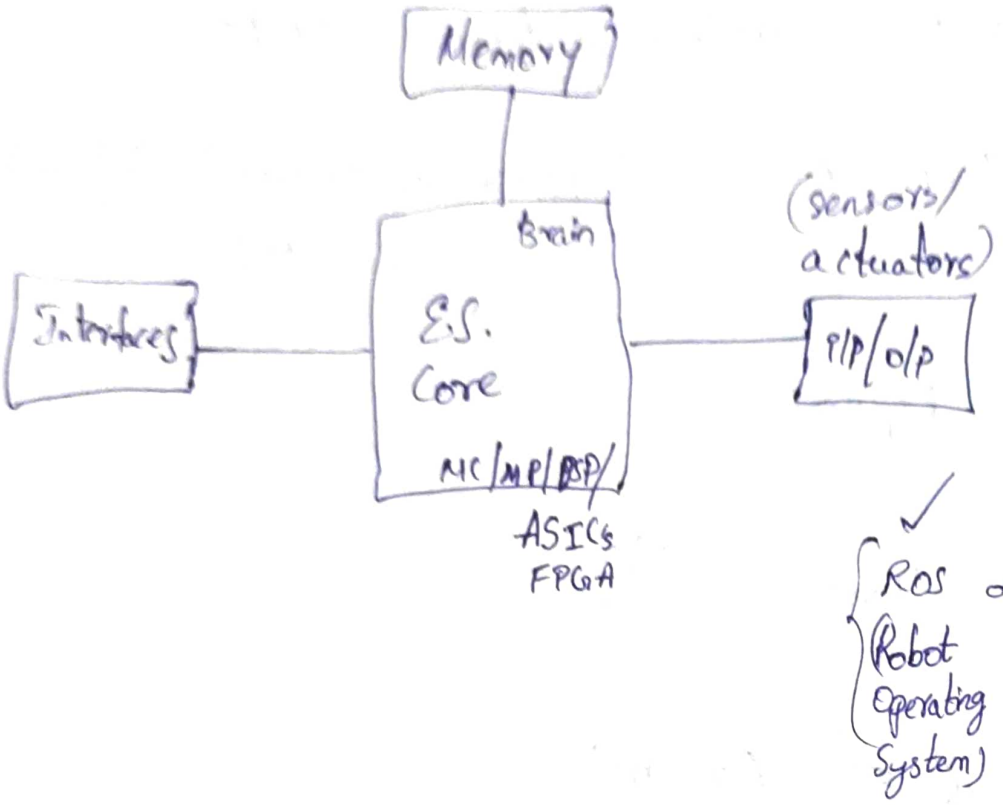
- Fridge, Refrigerator, Smart watches, Washing machine, Health monitoring, N/w devices – Routers, ONVs (optical N/w units) Hubs, Modems
- Cruise control (drive assist feature in car)

ESs used in a Car :

- Audio systems, Dash board, Air bags, Automatic Wipers, DVDs, Reverse cam, Monitoring System – Seat belt, Engine oil, Tyre pressure, Door lockkey
- Electronically Adjustable mirrors
- Automatic climate control AC
- Smart keys

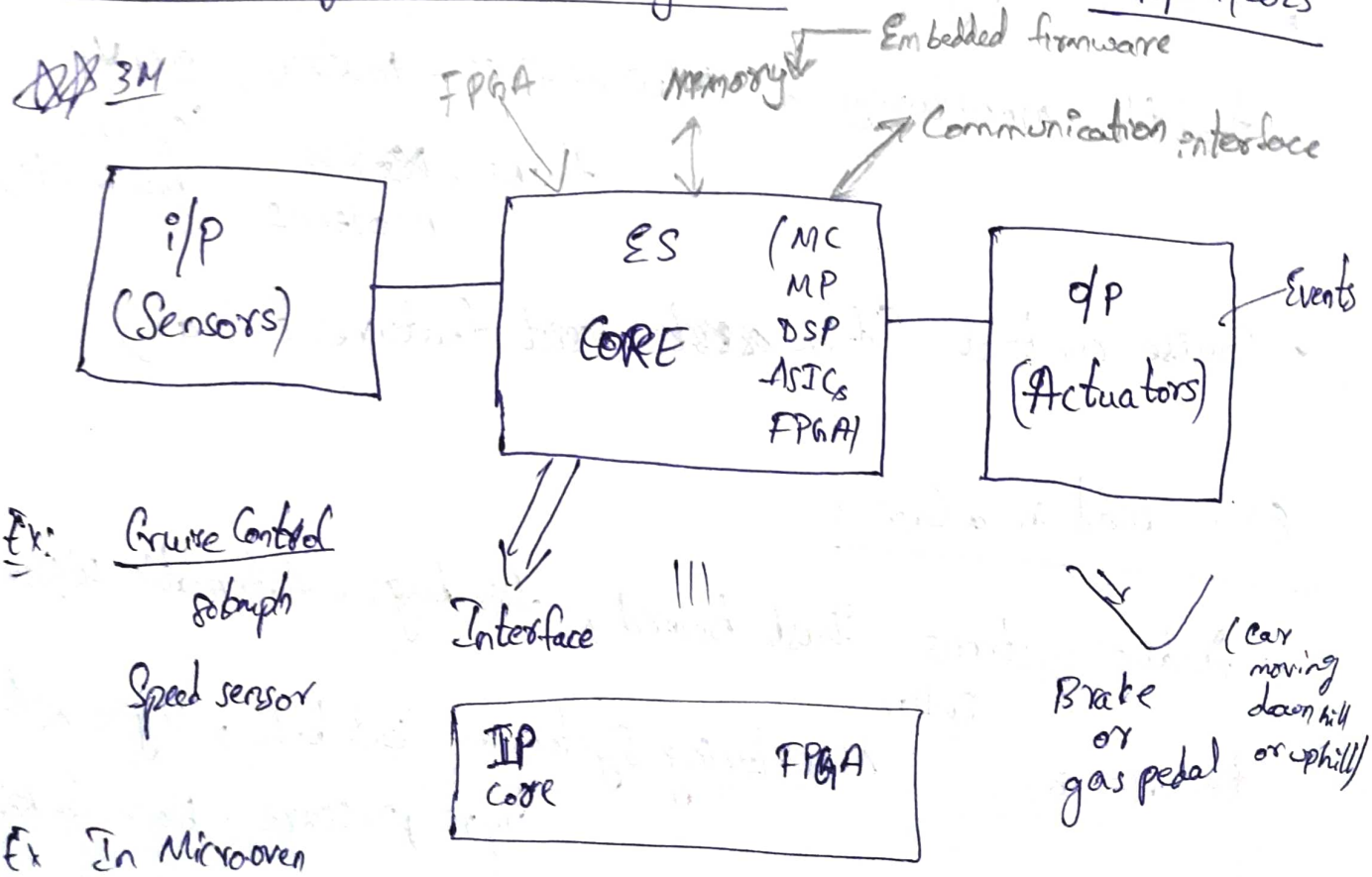
Block Diagram

05/01/2023



Embedded System Block Diagram

09/01/2023



Ex: Cruise Control
Sobugh
Speed sensor

Ex: In Microoven
(Temperature sensor)
Thermostat

• All IOT devices are E.S devices. (Converse need not to be True)

Design aspects of E.S.

(1) Application specific

(2) Efficiency (or) Performance

(3) Time critical (if application for military (security purposes))

• ES has IP core and FPGA

• FPGA – Field Programmable Gate Array.

☆ IP – Intellectual Property – is a functional block of logic or data used to make a FPGA or ASIC for a product.

Ex: for Audio system – Codec
Audio filter (for mobile)

• Matlab – can be installed only in PC (not in mobiles, ipad) (utilization)
(ipad – E.S - cannot change o.s)

• Playstation → E.S

☆ Software used in E.S is called as Firmware

Intellectual Property (IP) (its a branch related to Law)

IP core (IC) — design is patent (protected by law)

- MS office can work in - AMD
  INTEL
  ↳ (not much efficient, not reliable)

- GPC does not have "Hardware-Software Codesign"
  ↳ (E.S.)

- E.S. → glued software
  → more applications

Life cycle :

① Software - LC

- Agile (technology) - Prototype

② E.S. - LC

- Hardware - Software codesign

10/01/2023

M C

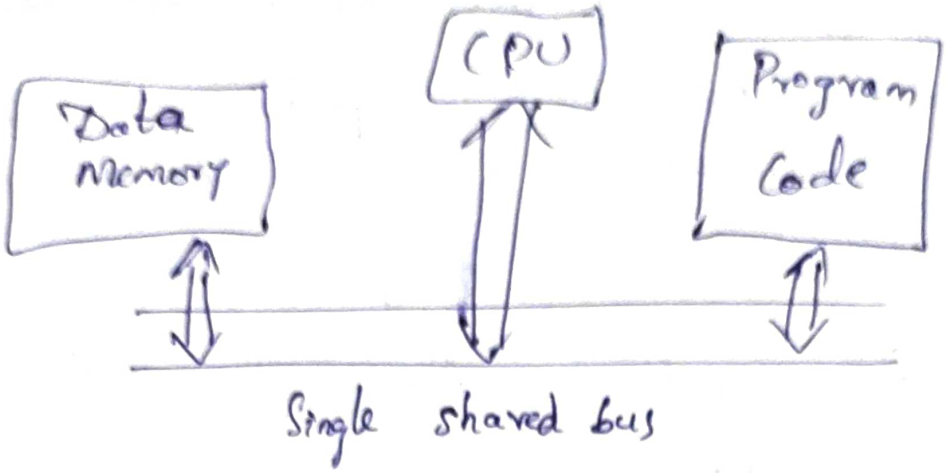
- CPU, Registers, I/O ports, Flash memory, Interrupt controller, Peripherals
- Processor speed - (500MHz)
- Includes lot of power saving features.

M P

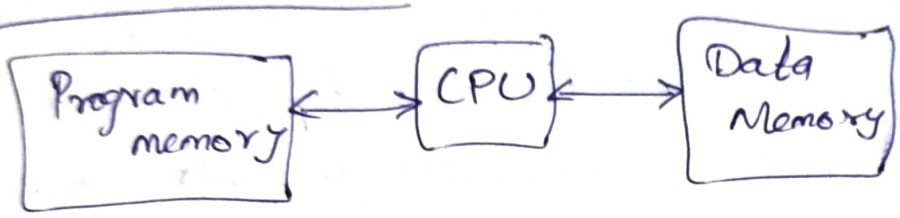
- CPU ≡ ALU + CU (control unit)
- General purpose
- Speed ~ 4GHz
- Limited Power saving options compared to MC.

Von-Neumann Architecture:

- Sharing common Bus

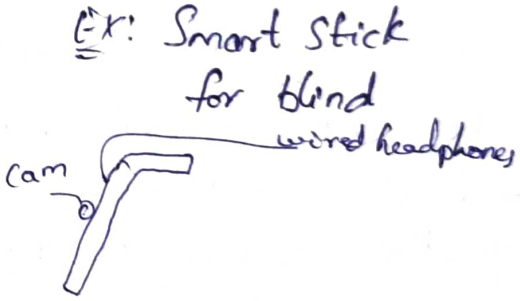


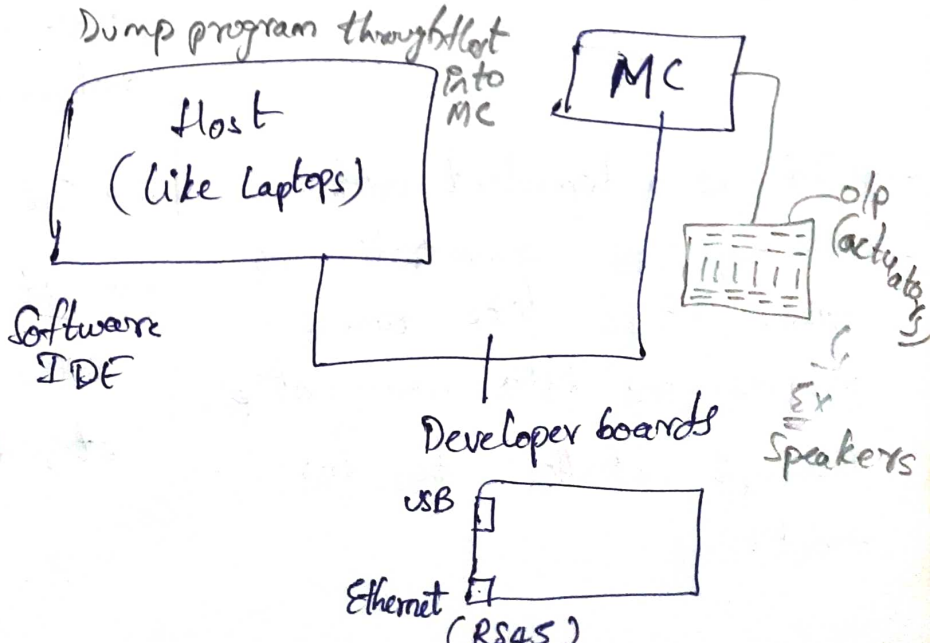
Harvard Architecture:



Experimental Setup for E.S:

1. MC core
2. Sensors (camera)
3. Jumper wires
4. Developer board
5. Bread board
6. Actuators (speaker)





Harvard Architecture

→ Separate buses for instruction and data fetching.

→ Easier to pipeline, so high performance can be achieved

→ Comparatively high cost

→ No memory alignment problems.

→ Since data memory and program memory are stored physically in different locations, no chances for accidental corruption of program memory.

Von-Neumann Architecture

→ Single shared bus for instruction and data fetching.

→ Low performance compared to Harvard Architecture

→ Cheaper

→ Allows self modifying codes

→ Since data memory and program memory are stored physically in the same chip, chances for accidental corruption of program memory.

Microprocessor

→ It is a Si chip representing a CPU, which is capable of performing arithmetic as well as logical operations according to a pre-defined set of instructions.

→ It is a dependent unit.

→ It requires combination of other chips like timers, program and data memory chips, interrupt controllers etc, for functioning.

Microcontroller

→ It is a highly Integrated chip that contains a CPU, scratchpad RAM, special and general purpose arrays, on chip ROM / FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports.

→ It is a self-contained unit.

→ It doesn't require external interrupt controller, timer, UART, etc, for its functioning.

Microprocessor

→ Most of the time general purpose in design and operation

→ Does not contain a built in I/O port. The I/O port functionality needs to be implemented with the help of external programmable peripheral interface chips like 8255.

→ Targeted for high end market where performance is important

→ Limited power saving options compared to micro controllers

Microcontroller

→ Mostly application-oriented or domain-specific

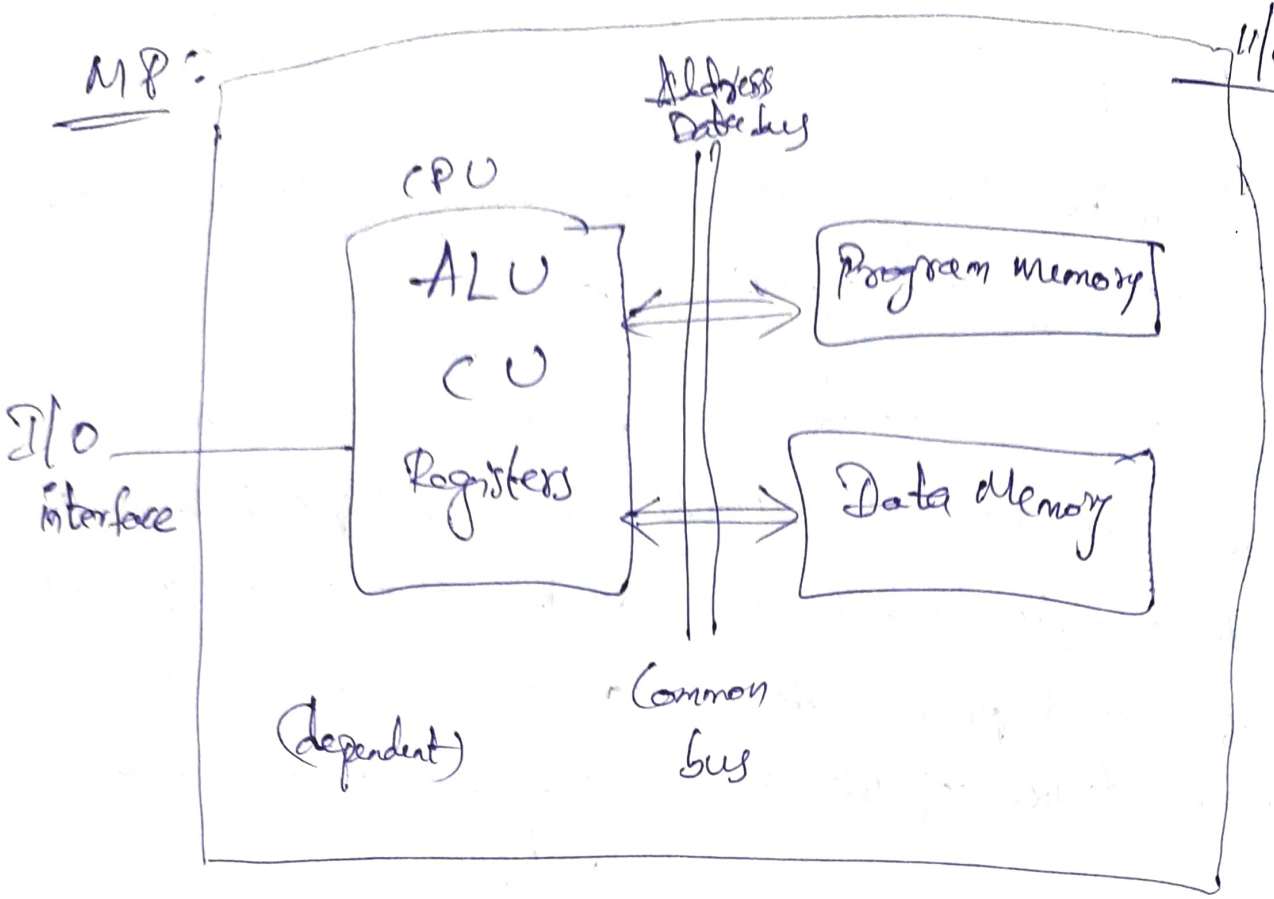
→ Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins.

→ Targeted for embedded market where performance is not so critical

→ Includes lot of power saving features.

11/01/2023

MP:



Micro Computer : GPC like Laptop, Desktop

MC : is a computer on single chip

PIC, size, Es

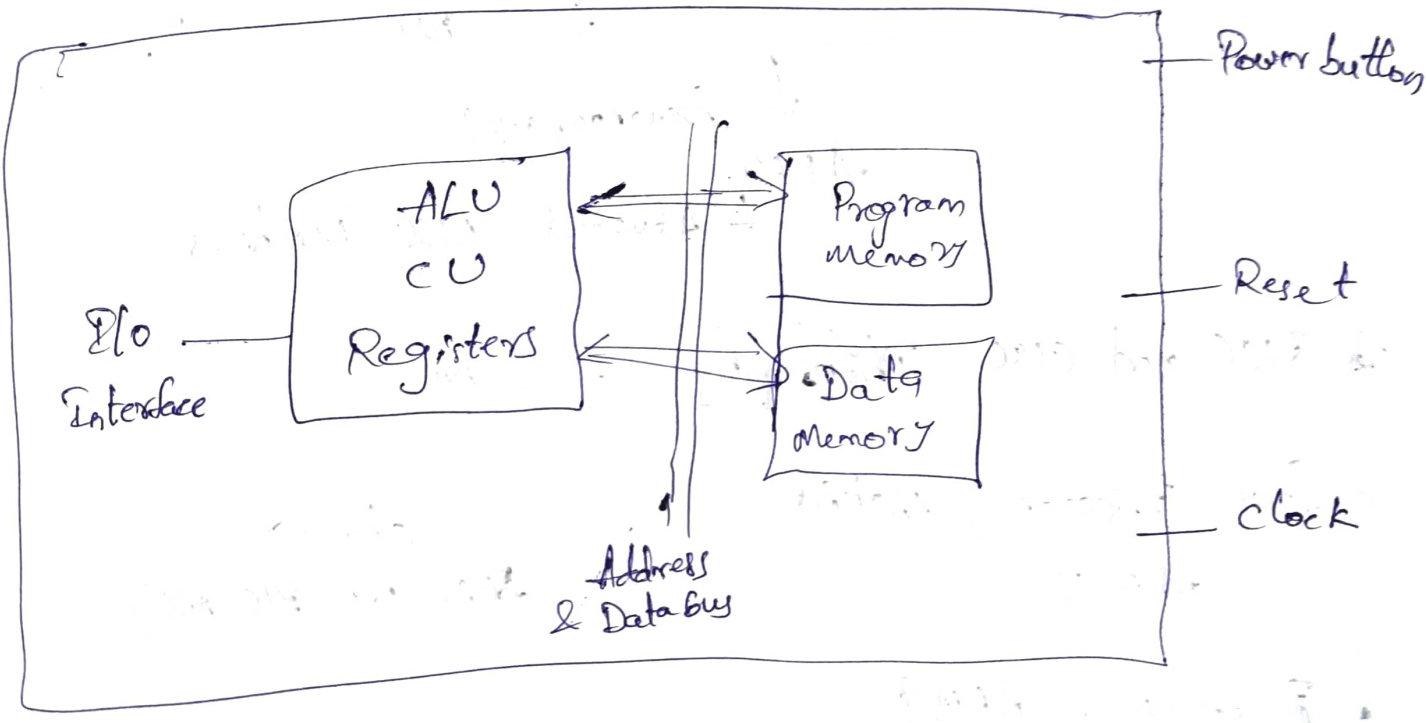
Design aspects of ES

1. Low Power consumption
2. Operation speed : 500 MHz
3. Efficiency
4. Application Specific
5. RISC based

Q. Difference b/w MC & GPC like laptop. (Mid-term)

OS: no task, only firmware	OS is changeable
on chip memory (memory limitations)	Disk memory
for single task	Multi tasking

Schematic diagram of MC - Microcontroller :



MC evolution :

1970 - intel 8051

mid 1985 - ASIC

reduced micro-IS

Intel based : Dell - based on CISC → RISC

Arm based : Mac like SOC → system on chip
M1
M2

Power efficient

Intel based (RISC) — Arm based

Dell — Mac

3hr — Power efficient (2days)

Performance oriented

History: Apple → Acron → 1st company, VLSI
↳ developed RISC based system BBC

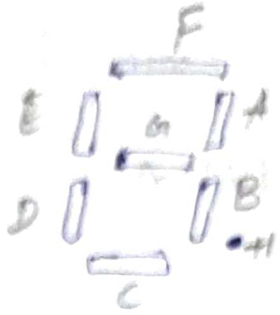
↓ (New company)

ARM
↳ Advanced RISC Machines

* RISC and CISC Philosophy – 2M

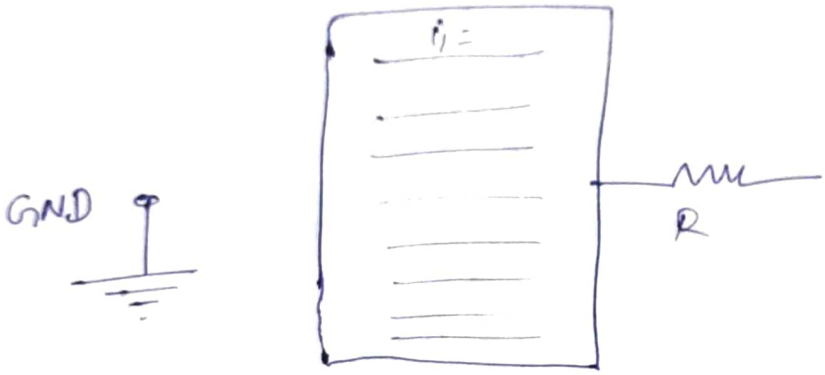
- Sony Ericsson, Nokia
  ARM-7 (choice)
  (ARM or PIC MC)
- E-phone – ARM 9

7 Segment LED.



- BLED
  - 0–9 → decimal
  - 0–F → hexa decimal
    (16)

- LED to glow : Current → 5–10 mA
  Voltage drop → 1.2 V



$i = 40mA$
$R = ?$
$= \frac{5-1.2}{40m}$

→ ATD, DTA converters

17/01/2023

I/O Devices

1) 7 Segment display
2) LCD
3) Speaker
4) Microphone
5) LM35
6) LDR
7) Touch screen
8) Smoke detector
9) Relays

17/01/2013

LCD: Liquid Crystal Display

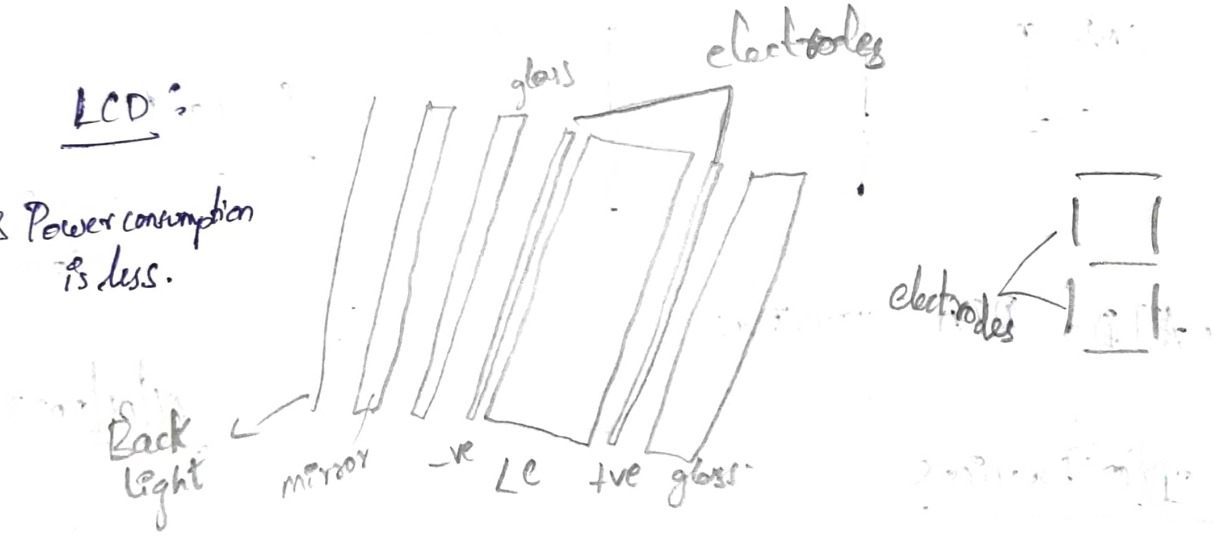
- JHD162A: 16×2

→ LCD (vs) LED

- for E.S applications
- OLED
- H & energy efficient
- Power consumption is less

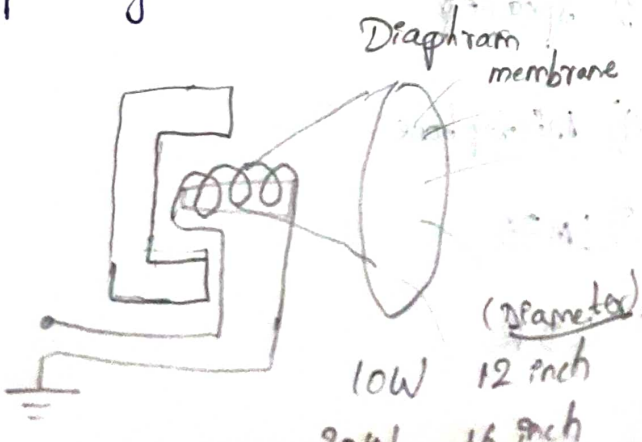
LCD:

- Power consumption is less.



Speaker:

- It has horse-shoe shape magnet, coil
- i/p – Electrical signals
- o/p – Audio signals
- It produces sound



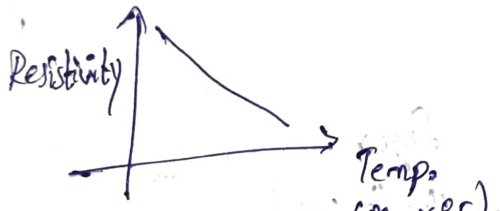
Microphone :

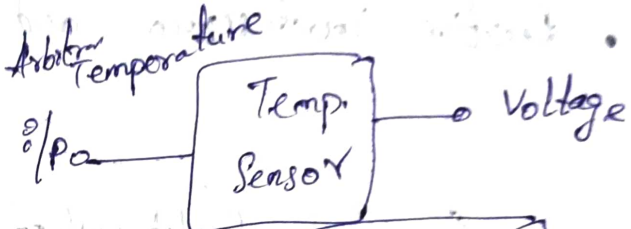
- Like a inverter of speaker.
- i/p - Audio signals & o/p Electrical signals.
- Diameter size in mm, μm
- Current in mA, μA

→ For Speaker, Microphone :- o/p is Analog

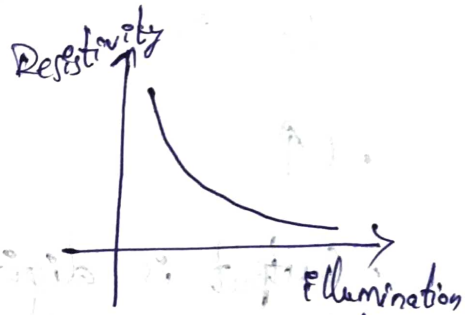
LM35

- Temperature sensor





- It is a Linear device
- $V = K_1 T$
- $K_1 = \frac{10mV}{1°C}$
- It is a semiconductor device.



- Property of a Semiconductor material

Resistivity : Linear w.r.t Temperature
Non-linear w.r.t Illumination

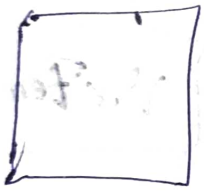
- Output is digital
- Applications : Laptops, AC's,

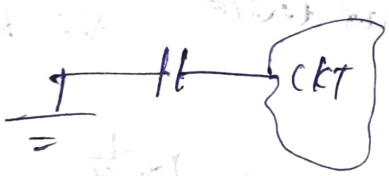
LDR : 2 pin device

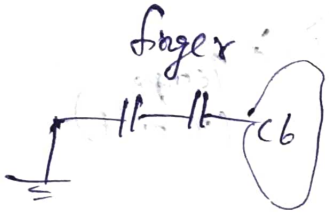
- ~~Light Dependent~~
- Light Detection Resistor
- o/p is Digital
- Application : In Home automation, phone brightness

Touch Screen : 3 pin device

- Resistive Touch screen and Capacitive Touch Screen





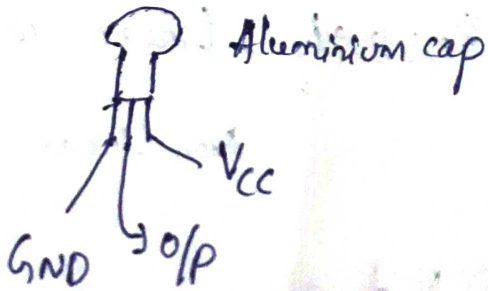


- C ↑
- Output is digital.

Application : Tracking the finger
(o/p is analog)

Smoke Detector

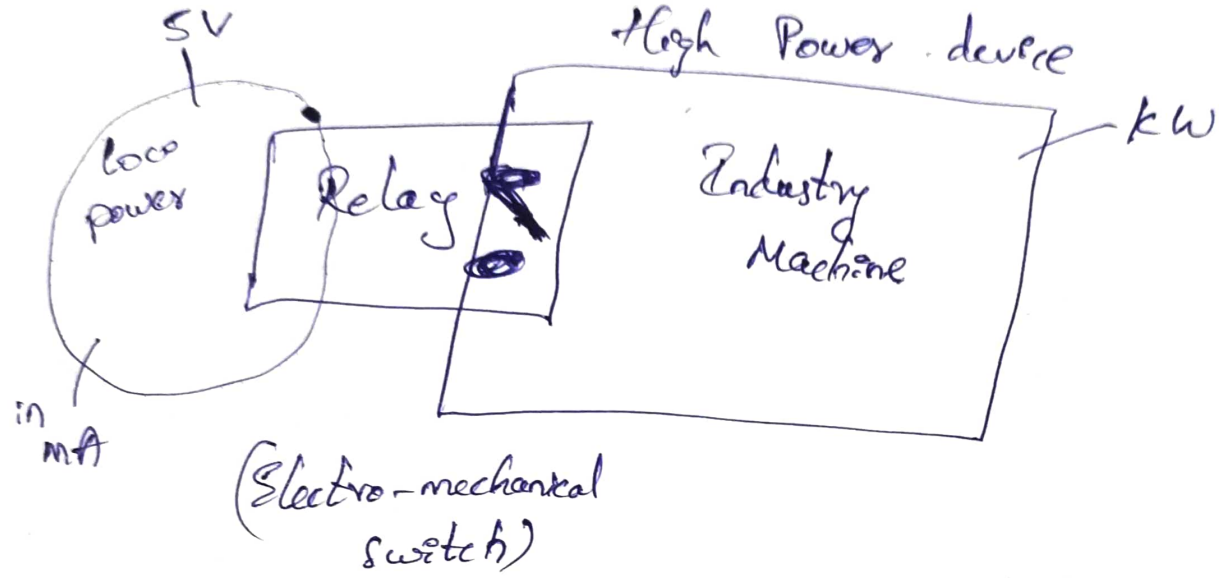
- Can detect finite set of gases -
  Benzene, Alcohol



Relays: (in 1831)

Electro-
- Mechanical switches.

- For high power application devices/instrument - (Industry - Machines)



(Electro-mechanical switch)

- Working mechanism same like Solenoid

- Relay is a 4 pin device
- Types: ① (Having Coil) Electromechanical
  ② Solid state (more reliable)

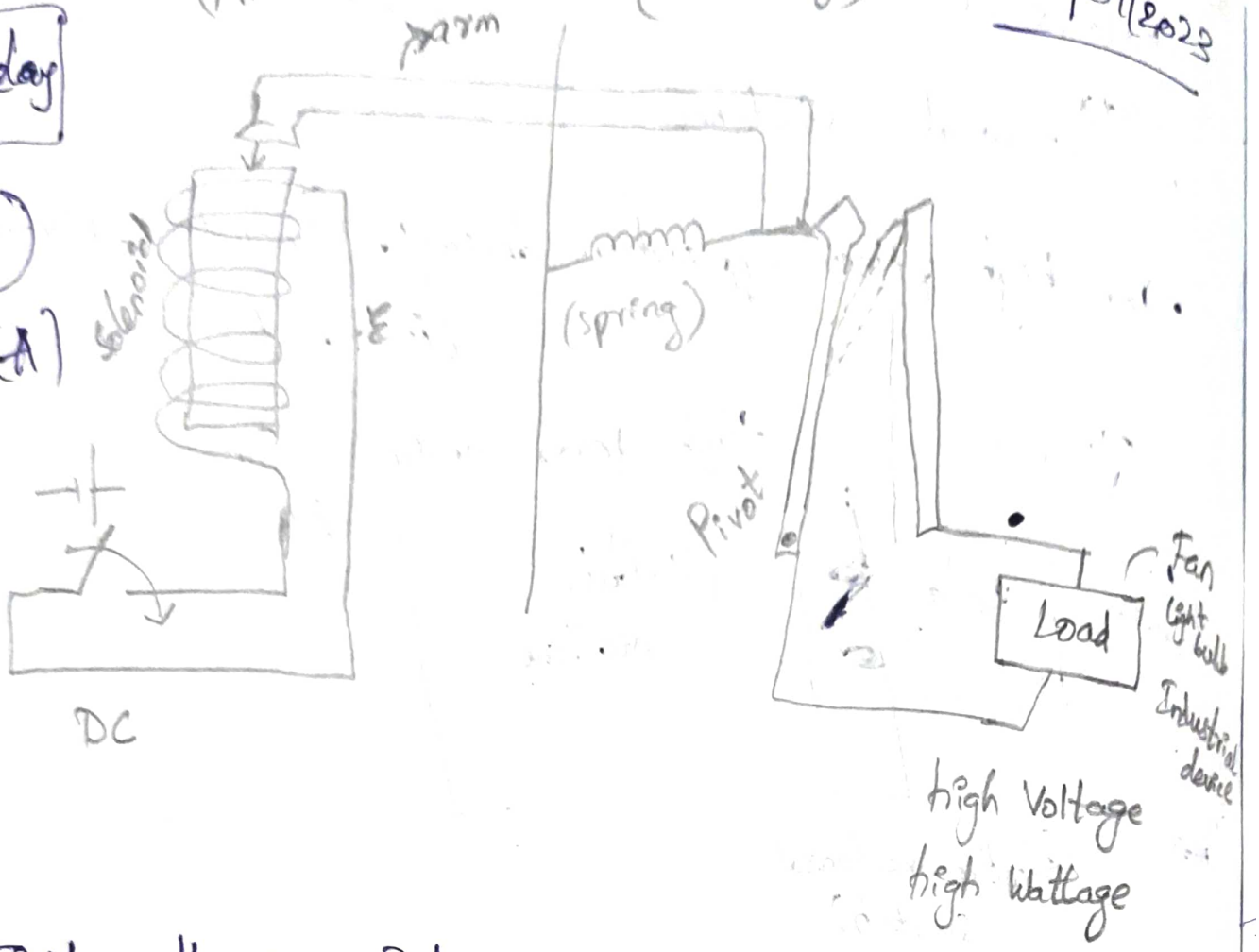
- Application: used in Telegraph

(Primary) (Secondary) ∴ 18/01/2023

Relay

(1)

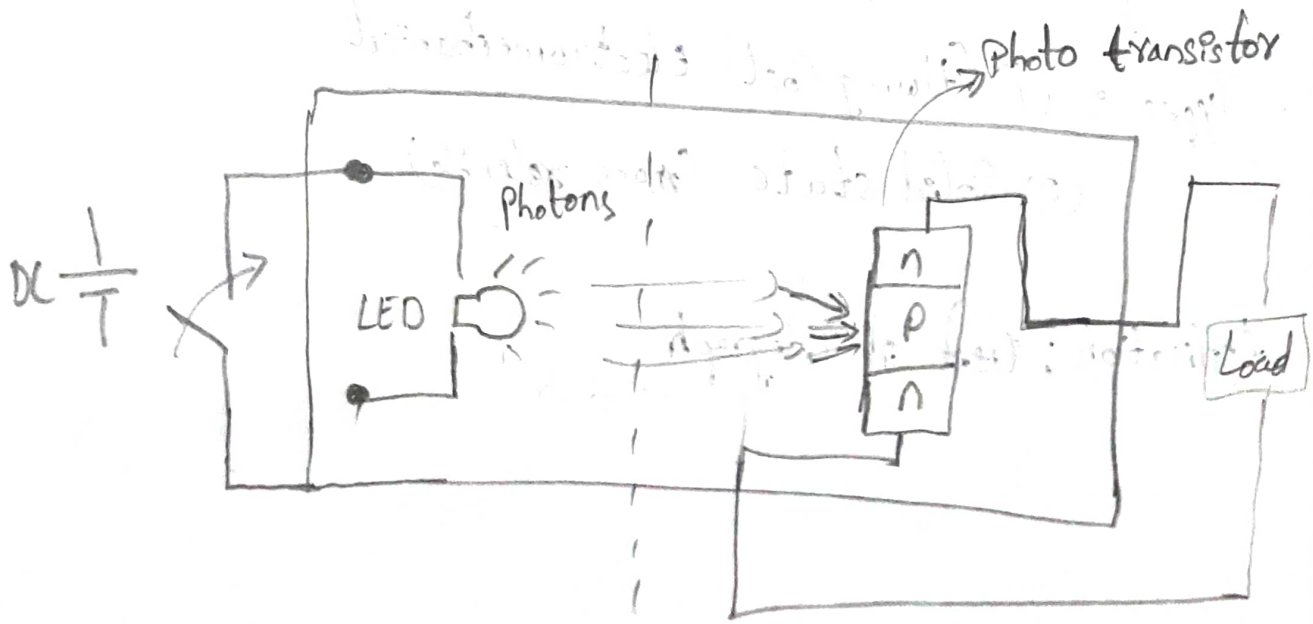
(A)

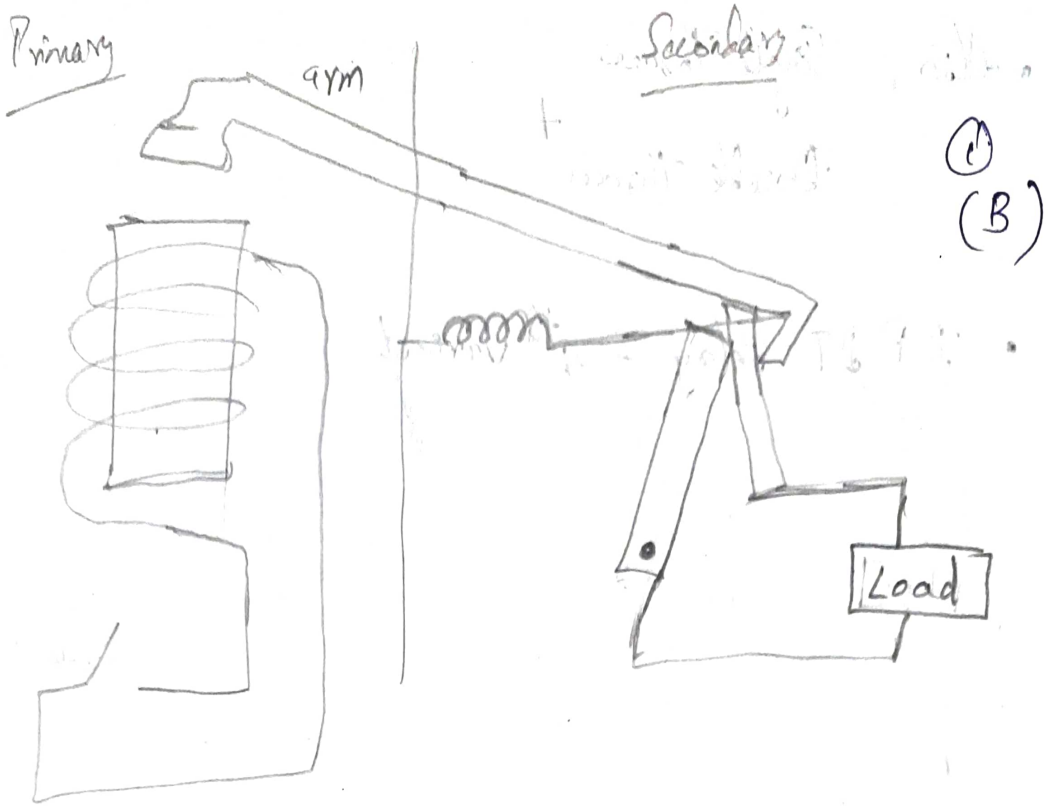


(A) Normally open Relay i.e. primary-open then secondary-open

(B) Normally close Relay i.e. 1° → open then 2° - closed

(2) Solid State Relay: (instead of movable arm, it has LED)



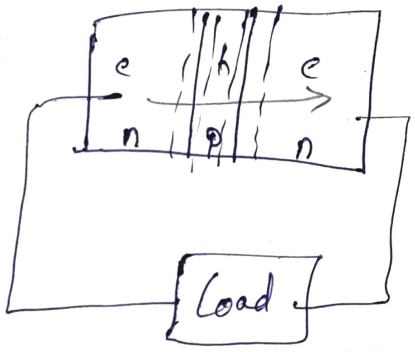


(A)

(B)

(2) Solid state Relay

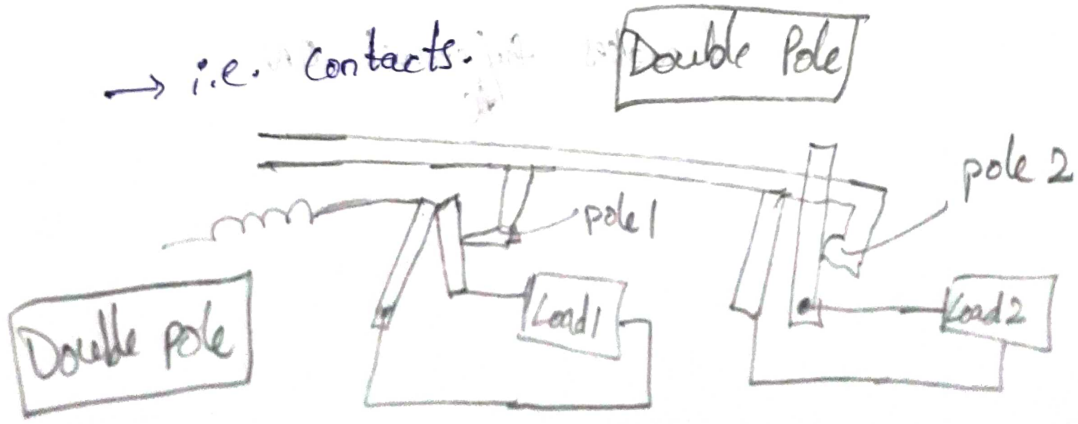
- Photo transistor :



- photons repel holes
  & so e^- moves
  ⇒ current flows ✓

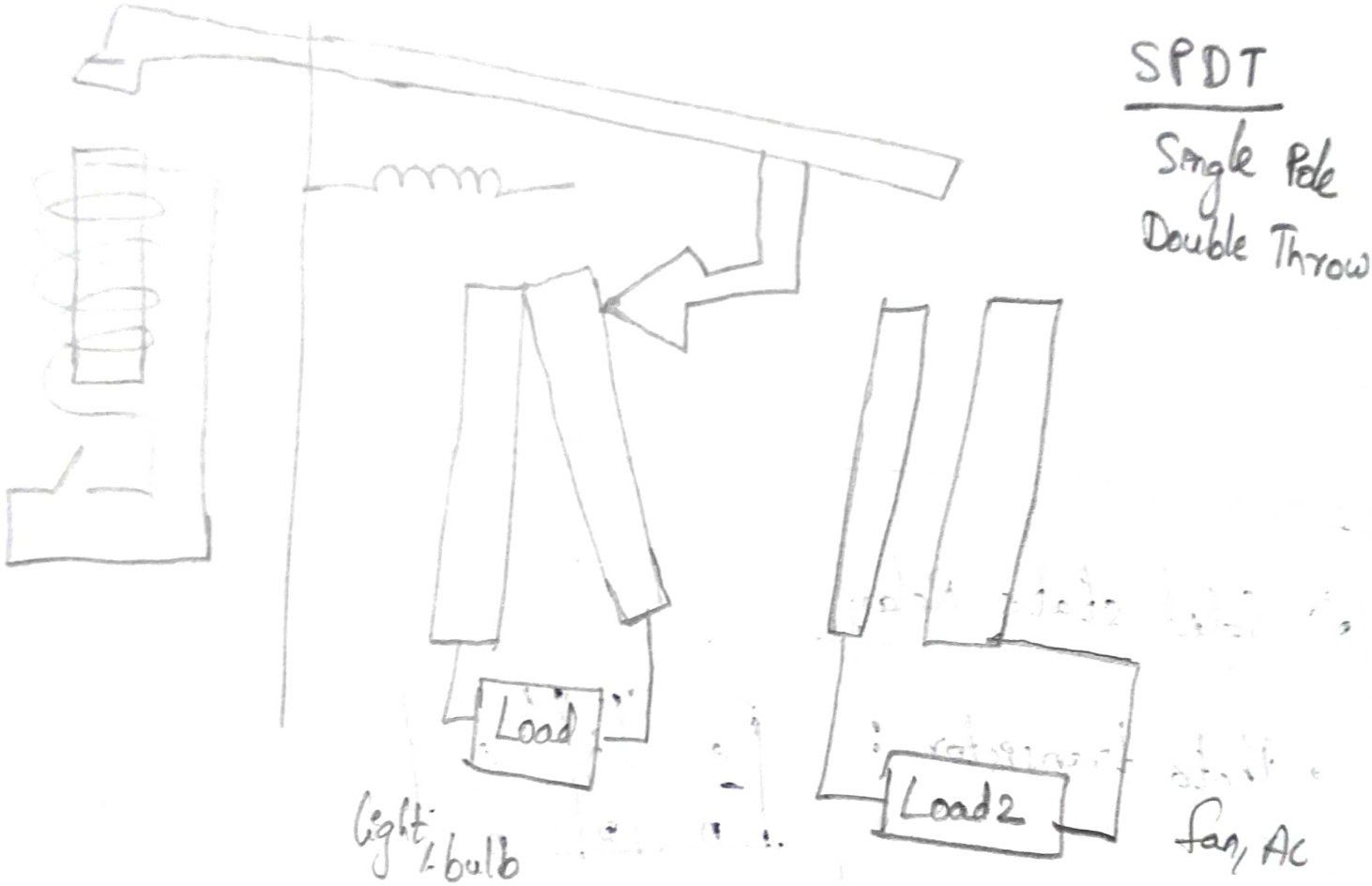
(1) Electromechanical Relay — can have Single pole and Double pole

→ i.e. contacts.



- Also, Single Throw + Double Throw

- DPDT Relay – 4th variant



Relays:

- Applications: Telegraph
  Telephone

- Bimetallic strip – when Temp. high, it expands & makes fan/AC circuit ON

- PIC - it is a 4bit μC

  ↳ How many I/O devices can handle at a time?

  I/O devices → [MC]

- Multiple sensors, Relays

Ex: Parking a Car : ① back camera,

② proximity sensor,

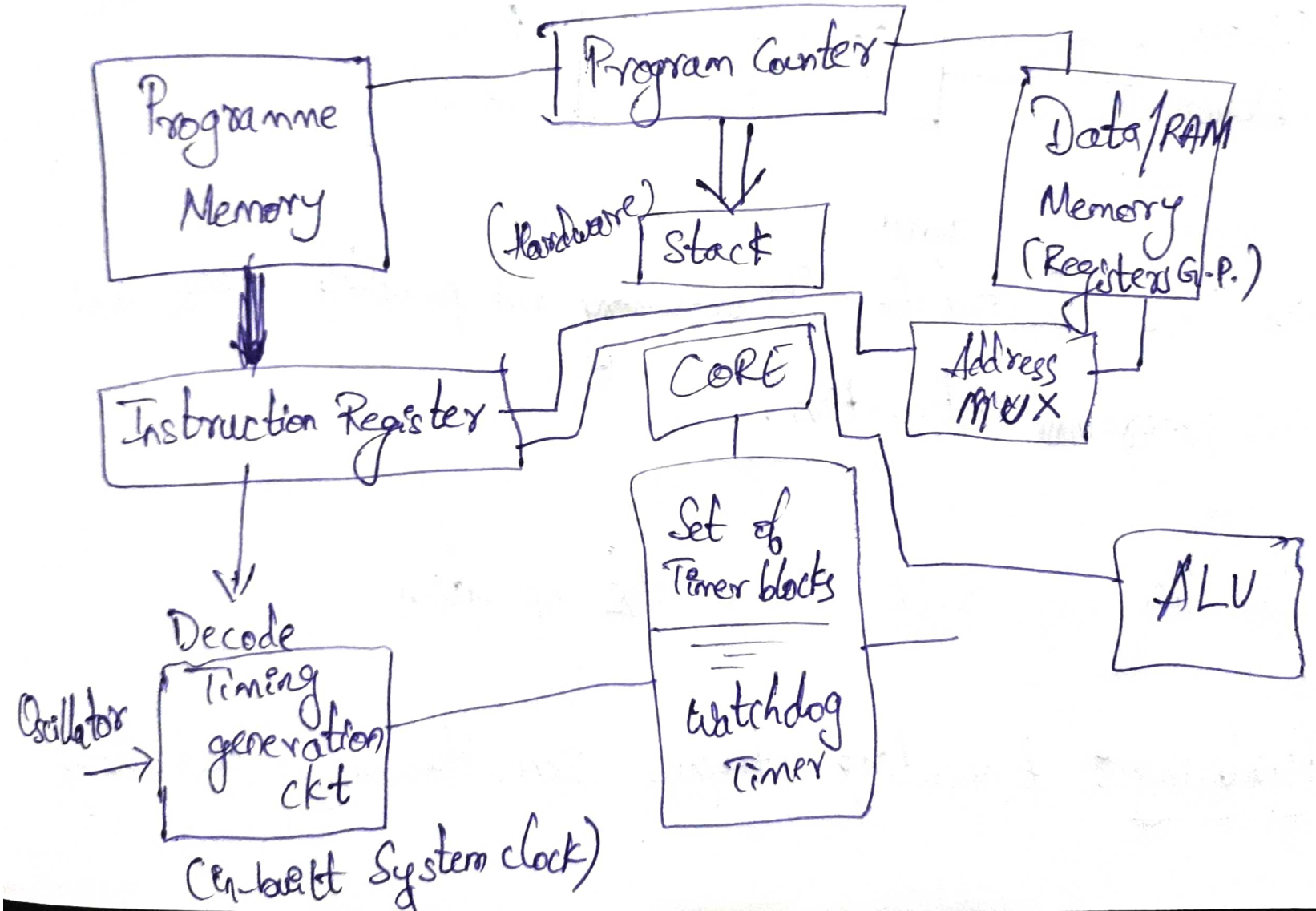
③ Beep (sound) Speaker

④ Lights

23/01/2022

(Memory Units:) Peripheral Interface Controller

PIC Mid Range Block Diagram : (Harvard type)



• Program counter :

→ Can fetch information from Program memory and Data memory.

→ Contains the address of next instruction.

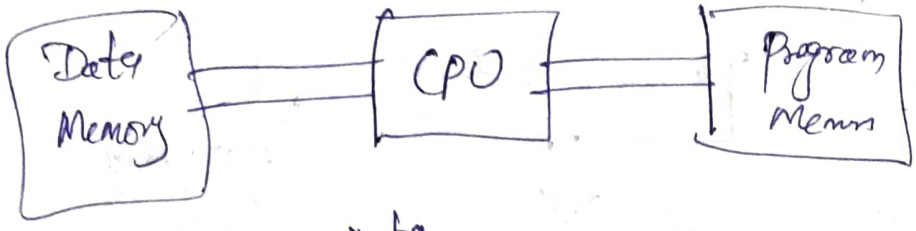
• Instruction length is variable in CISC

• Instructions executed per clock for the case of CISC is more than 1 (while for RISC is 1)

• 90% of MC (based on) follows RISC

→ PIC belongs to RISC

Pipelining :-



→ CPU fetches data from Datamemory and parallely access next programme instruction.

. In this way, RISC eases the operation.

Pipelining : Executing stages simultaneously / parallely

Watch dog Timer:

- It has Count variable
- If it does not receive Reset from core, then this watch dog timer comes to picture

→ Instruction Register has both Program & Data

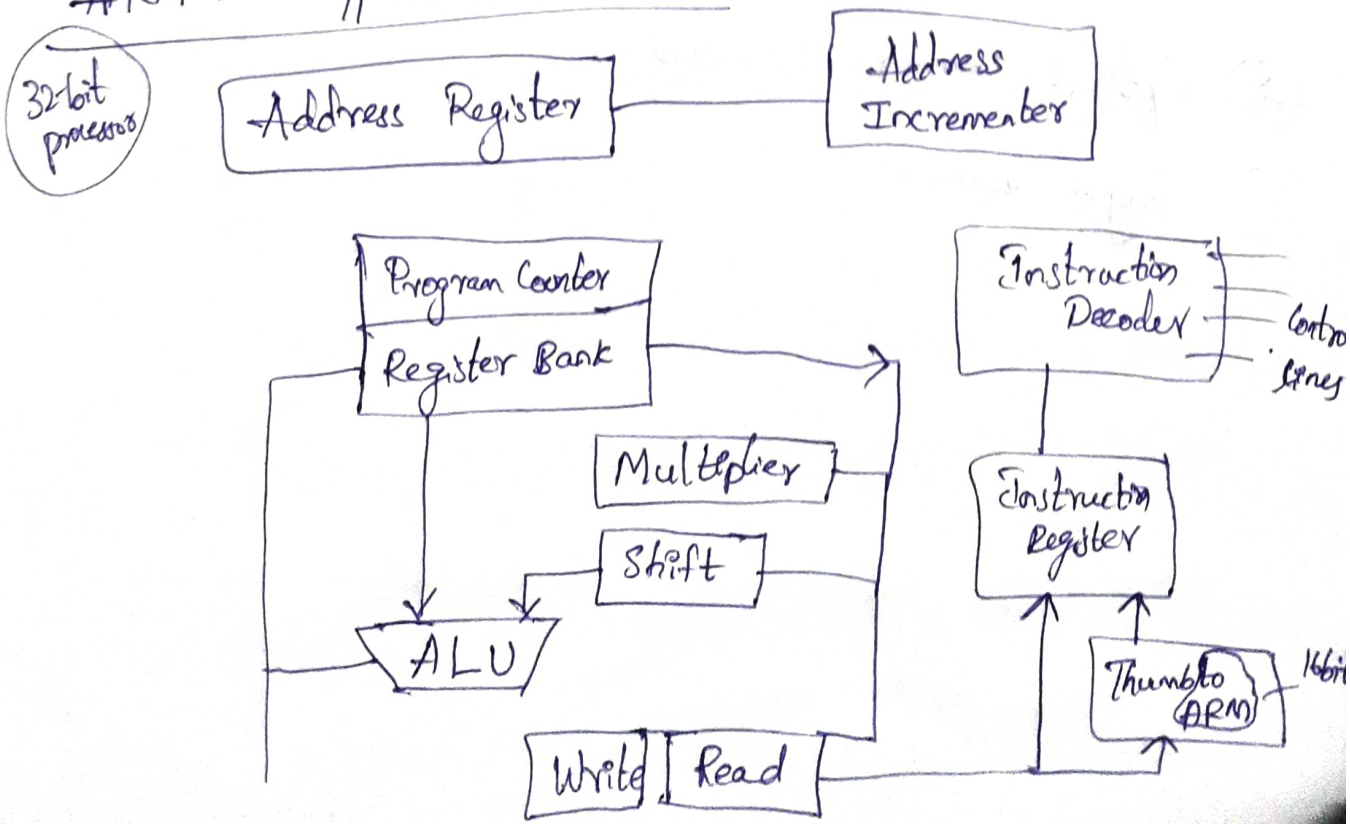
→ PIC has either 8/16 bit bus (low end)

→ Core interfaces with Multiple sensors/actuator via series or parallel I/O ports.

→ PIC can handle 5 no of multiple sensors concurrently.

24/01/2023

ARM 7 - Typical Architecture:



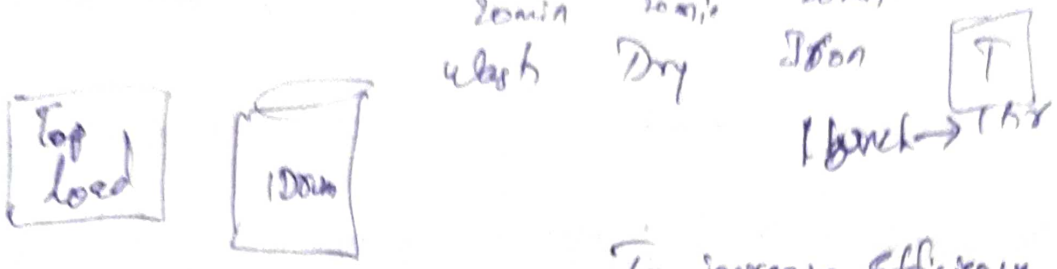
- ARM 7 → 32 bit processor
  - ↳ Von-Neumann
- ARM 9 → Harvard
- ARM 4 → Slightly deviates from RISC
- ARM 7 → 3 stages : Fetch, Decode, Execute
- ARM 9 → 5 stages : Fetch, Decode, Execute, Memory, Write
- ARM 10 → 6 stages : Fetch, Issue, Decode, Execute, Memory, Write

(1) Statistical Signal Processing
(2) Game Theory
(3) Decentralized Detections
(4) Optical communication (POW)
(5) Crypto Currencies

— Dr. John Bob Sir (Specializations)

Pipeline feature:

Example: Washing Machine



To increase Efficiency

1 Bunch clothes = B_1

→ 4 No: of Bunches need to be washed

— Semi auto W.M

(3 stages)

W	B1	B2	B3	B4		
D		B1	B2	B3	B4	
I			B1	B2	B3	B4

B1 → B2 →

4 bunches → 2 hours

$B_1 \rightarrow 3\tau$

$B_2 \rightarrow 4\tau$

$B_3 \rightarrow 5\tau$

$B_4 \rightarrow 6\tau$

$B_N \rightarrow (N+2)\tau$

- 'N' bunches $\Rightarrow (N+2)\tau$
- If 'K' stages $\Rightarrow (N+(K-1))\tau$ (instead of 3 stages)

$$\therefore T_K = [N+(K-1)]\tau \quad ; \quad K - \text{No. of stages}$$

In fridge: every 1 bunch → 1 hour (after) (without pipelining)

(with pipelining) every one bunch → after every τ (20 min) we are getting bunches

Time required for each stage

⇒ Efficiency factor = $\frac{60}{20} = 3$ (max efficiency)

- T_0 – represents "without pipeline" Time required without pipeline for N bunches

→ Speed up (S_k) = $\frac{T_0}{T_k} = \frac{NK\tau}{[N+(K-1)]\tau} = \frac{NK}{N+K-1}$

→ When $N \to \infty$ ⇒ Speed up = K

- Pipeline Efficiency = $\frac{N}{K+(N-1)} = \frac{S_k}{K}$

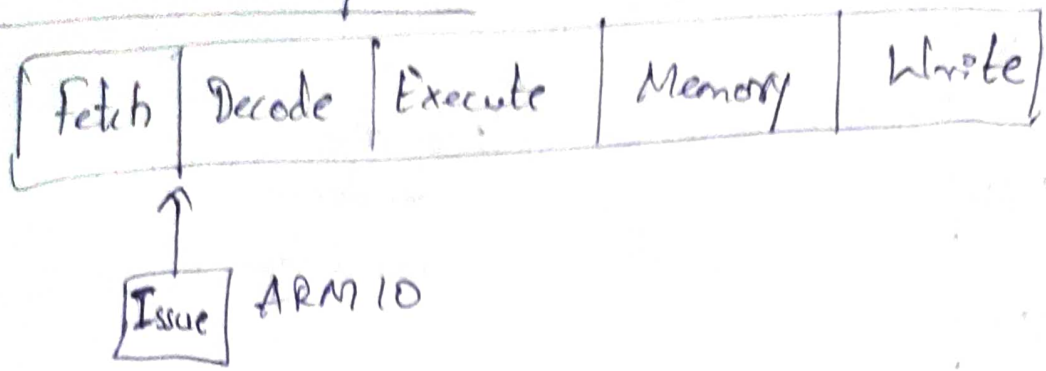
Pipeline throughput : No. of jobs done per time

$$H_k = \frac{N}{T_k}$$

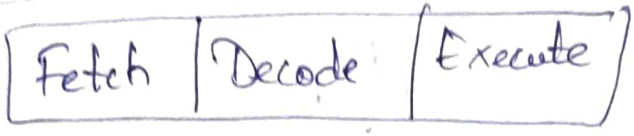
- For ARM7 → it has 3 stages (Fetch, Decode, Execute)
- ARM9 (Harvard) → 5 stages (Fetch, Decode, Execute, Memory, Write)

→ For each particular stage or between each stage, we need to have Latch.

ARM 7 TMI Pipeline

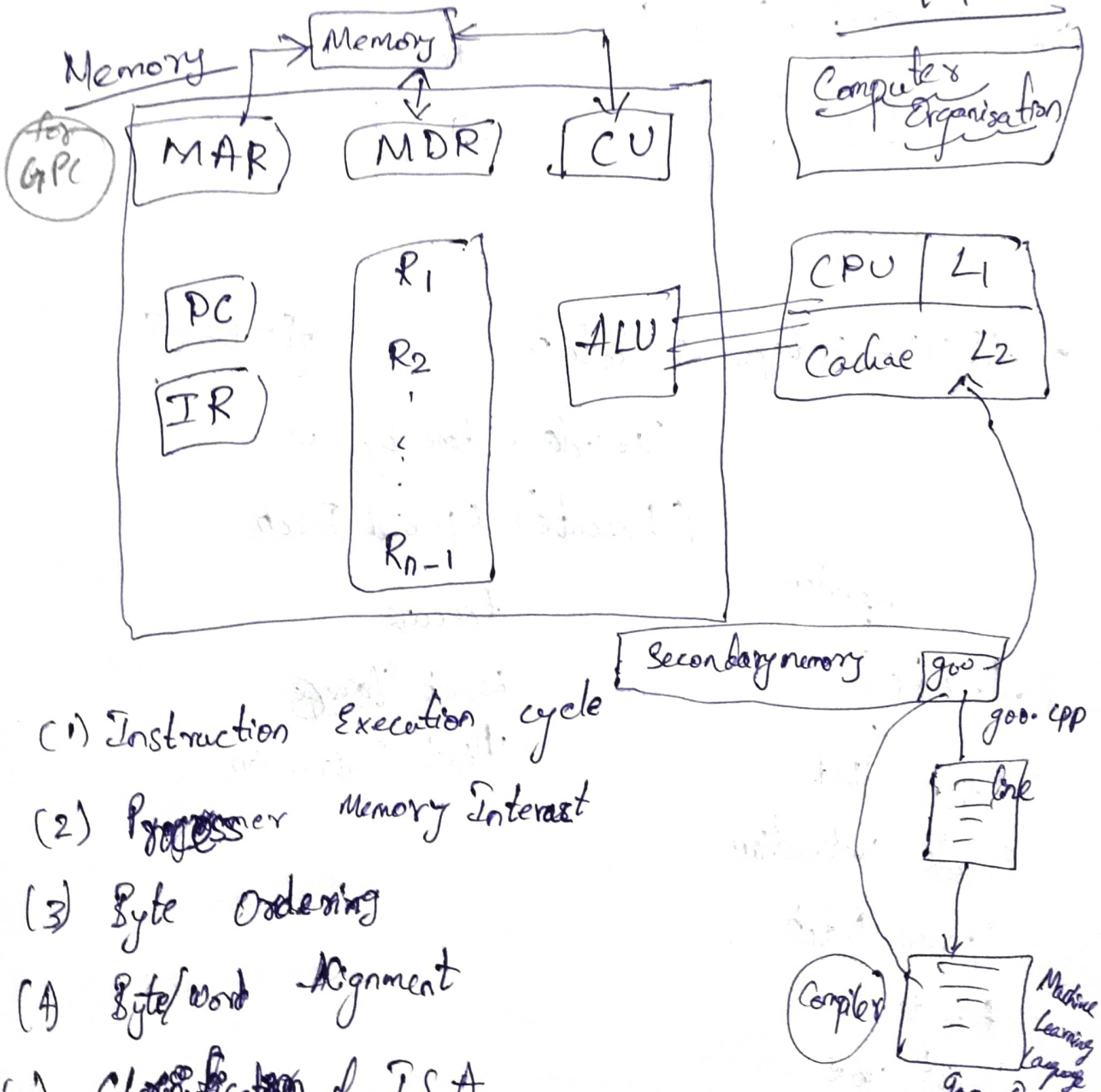


ARM 7 TDMI pipeline



30/01/2023

Computer Organisation



(1) Instruction Execution cycle

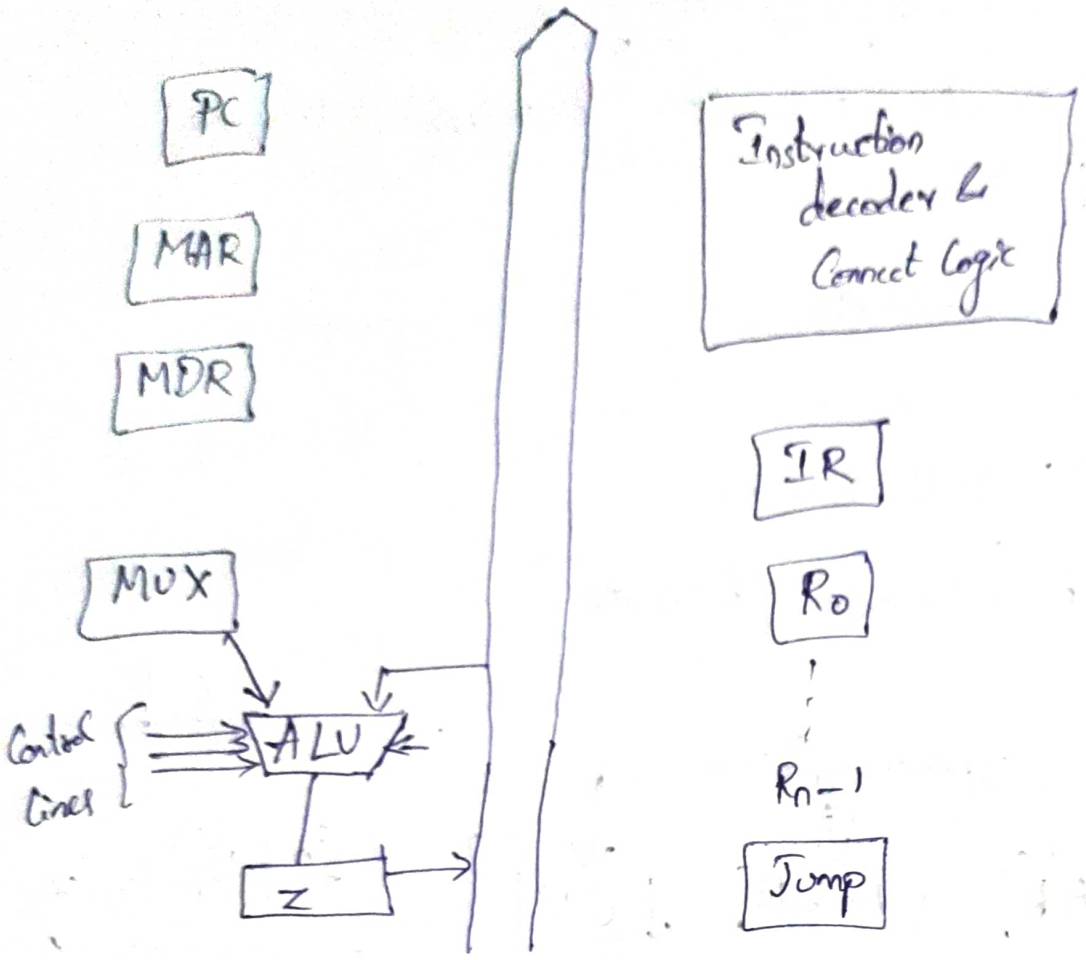
(2) Processor Memory Interact

(3) Byte Ordering

(4) Byte/word Alignment

(5) Classification of ISA

30/01/2023



- ARM7 – 3 stages : Fetch
  Decode - done by CU
  (Execute) Operand Fetch
  Execute
  Result Storage
  Next Instruction

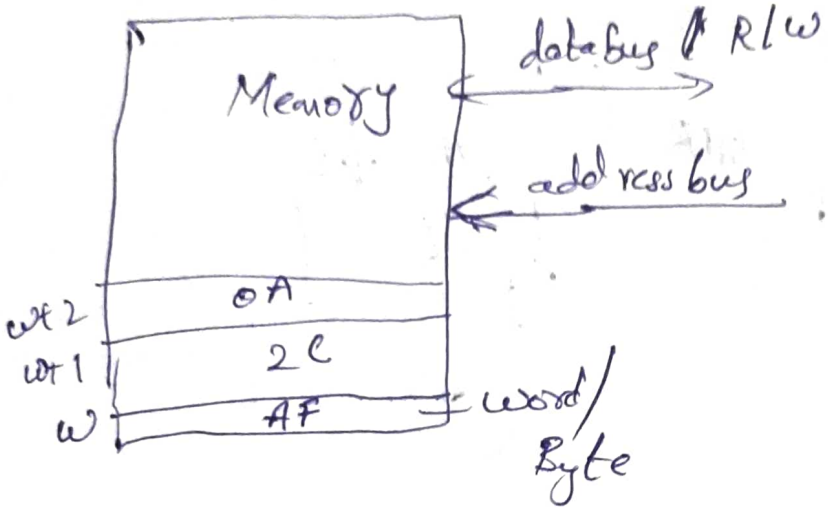
A = B + C

operator

A = B + C

Fetch Inst.
Decode Instruction
Operand Fetch
Execute
Result store
next Inst.

. Memory stored based on Memory Architecture.



Byte Ordering

(1) Little Endian

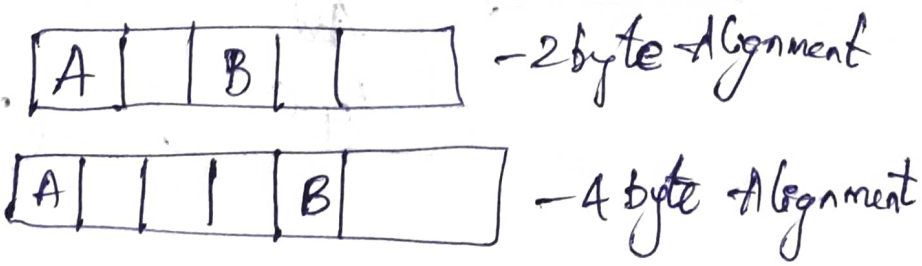
(2) Big Endian

0A2CAF

(1) Little Endian : LSB stored in Lowest address.

(2) Big Endian : LSB stored in Highest address

Word Alignment:



Classification of ISA :

(1) Stack

(2) Accumulator

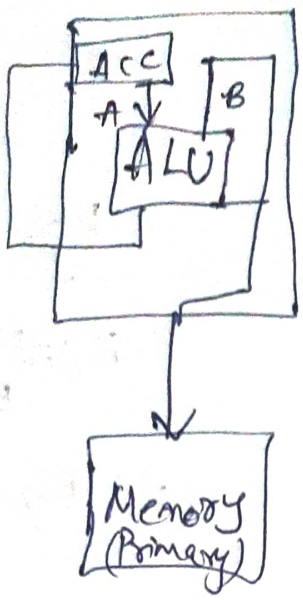
(3) Register-Memory Architecture

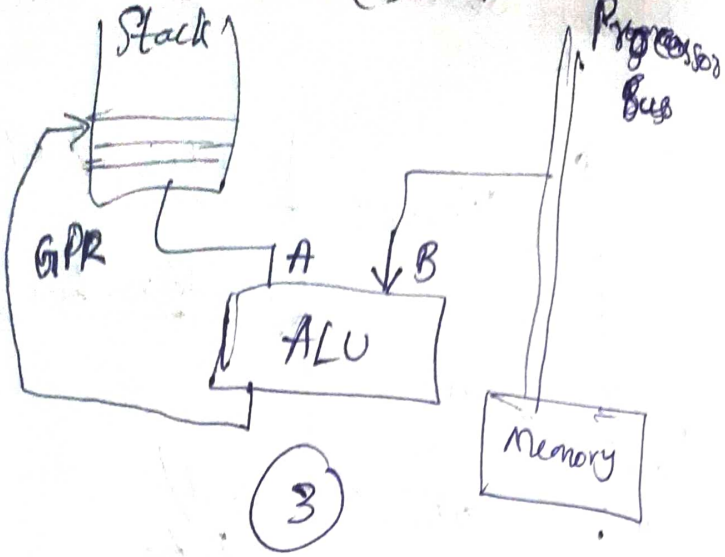
(4) Load & Store

CPU

C = A + B

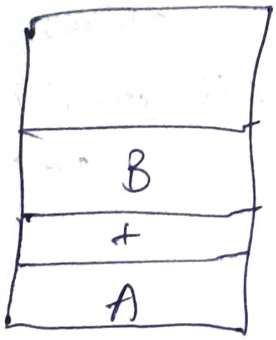
(2)





(3)

(1)



(1)

Push A
Push +
Push B
Add
Pop C

(2)

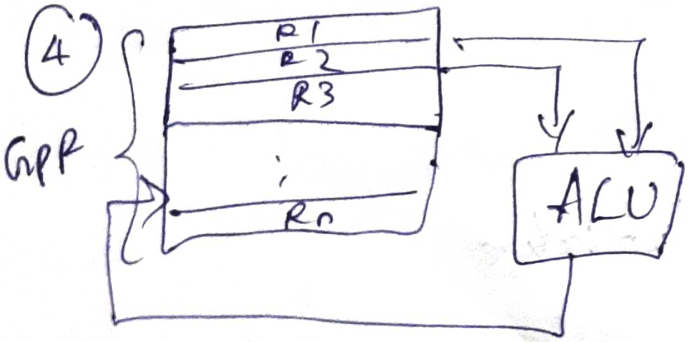
Load A
Add B
Store C

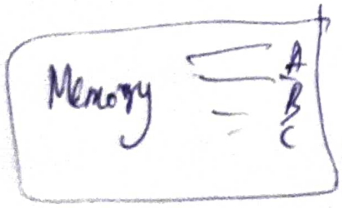
(3)

Load R_1, A
~~Add~~ R_3, R_1, B
Store R_3, C

. Op code
Assembly language

(4)





(Mostly Primary memory)

(4)

Load R_1, A
Load R_2, B
Add R_3, R_1, R_2
Store R_3, C

ROM) / Code Memory

31/01/2022

Volatile

RAM vs SAM

→ MROM

→ OTP

→ EROM

→ EEROM

→ EE PROM

→ NVRAM

SRAM

DRAM

NVRAM

• RAM – Volatile Memory

• Serially Addressed Memory – SAM

• Directed Address Memory – DAM

• ROM – Non Volatile memory

FLASH : Ex: Pen drives, μ SD card (Secure-Digital Card) (latest)

• SAM – Ex: CD, Magnetic tapes, Oldies

• ROM (or) Code Memory

E.g. Program ≡ Control Algorithm – it is dumped on ROM.

• Masked ROM ≡ MROM

• One Time Programmable ROM ≡ OTP ≡ PROM

• Erasable ROM

• Electrically Erasable Programmable ROM

• Non-Volatile RAM

• ROM – Non Volatile Memory

→ ROM stores control algorithm.

• ROM is independent of the power.

• We can erase & access it fast, so its called as FLASH

- MROM - Cheap, stores control algorithm once, cannot erase.

- OTP - Stores once & erases once (updating firmware is allowed for only once)

- MROM based on FET

PROM:

- Poly Silicon (or) Nichrome

- UV - exposure - can change memory

EEPROM: Some crystal is used

- FLASH is the latest one.

- In FLASH, we can erase & reprogram it for 10^6 cycles (based on manufacturing)

- Accessibility is more for RAM when compared to ROM. (RAM → 3 X ROM)

- BIOS - Basic Input Output System Configuration

- NVROM - has inbuilt battery - duration of 10 years

SRAM cell

- Cells ~~charge~~ defined by "voltage"
- 6 CMOS
- Don't require refreshing
- Expensive
- Faster comparitively

DRAM cell

- Cells ~~charge~~ defined by - "charge" - (0,1)
- CMOS + Capacitor
- Refreshing required
- Less Expensive
- Less faster

- Memory Selection for E.S.

Software-Hardware Co-Design & Development

1/2/2023

(7.1) Pg.210

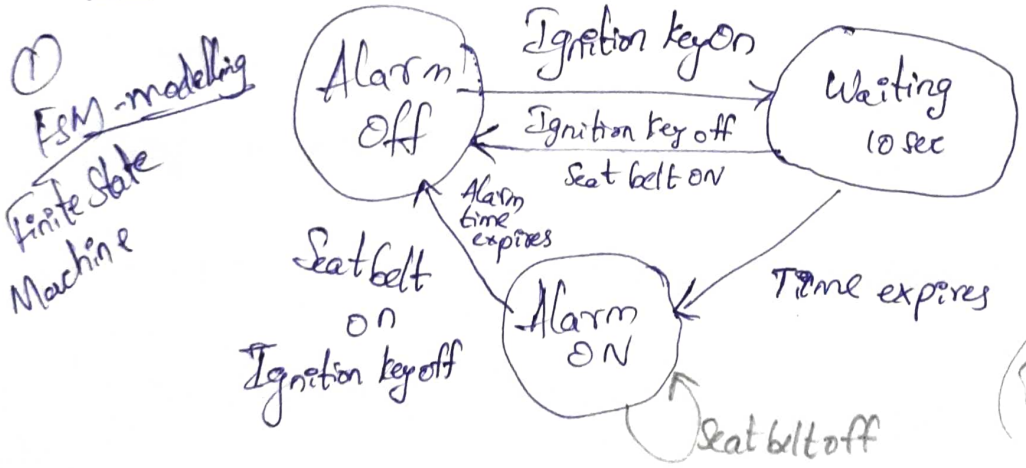
- UML diagram
- OOPS

→ Seat belt ES design: (Software cycle)

Agile cycle

1. System Level Requirements
2. Model (i/p, Data, State of Machine, o/p)
3. Testing
4. Revalidation

Seatbelt Alarm



(Reg. 7.3, 7.4, 7.5, 7.6)

(2) Data Driven Graphs / Data Flow graphs (DFG)

FSM: (1) State of System

(2) Elements (which trigger the states) - messages

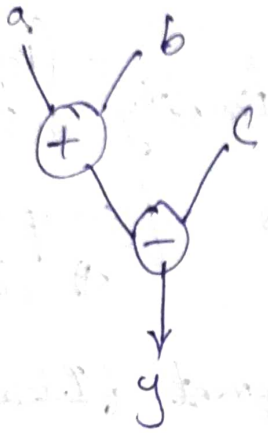
(3) Transition - in terms of "errors"

(4) Actions - "nodes"

Ex: Coffee Vending Machine.

- ES for ① DFG (Data Driven System) (Ex: Camera) V.C. ② CDFG (Condition Data flow Graph)

Ex:



$x = a + b$

$y = x - c$

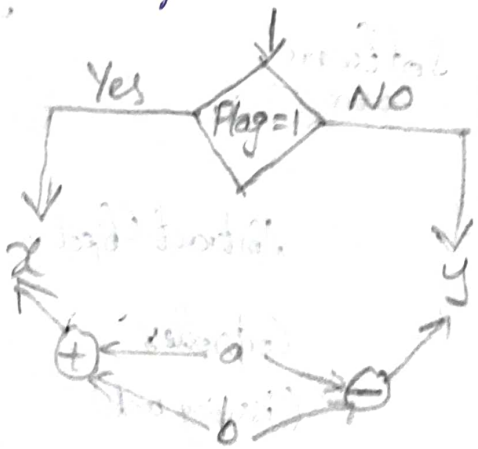
DFG

Ex: flag = 1, $x = a + b$ else $y = a - b$

CDFG

leaves are data

(Textbook 2.2.1)



06/02/2023

- UML - Unified Modelling Language: contains Symbols & arrows.
- OOPS - Object Oriented Programming System.
- Views: → User,
  - → System Designer,
  - → Testing / Validation

6/2/2023

- 'C' is a kind of sequential language.
- OOPS - Ex: Java, C++

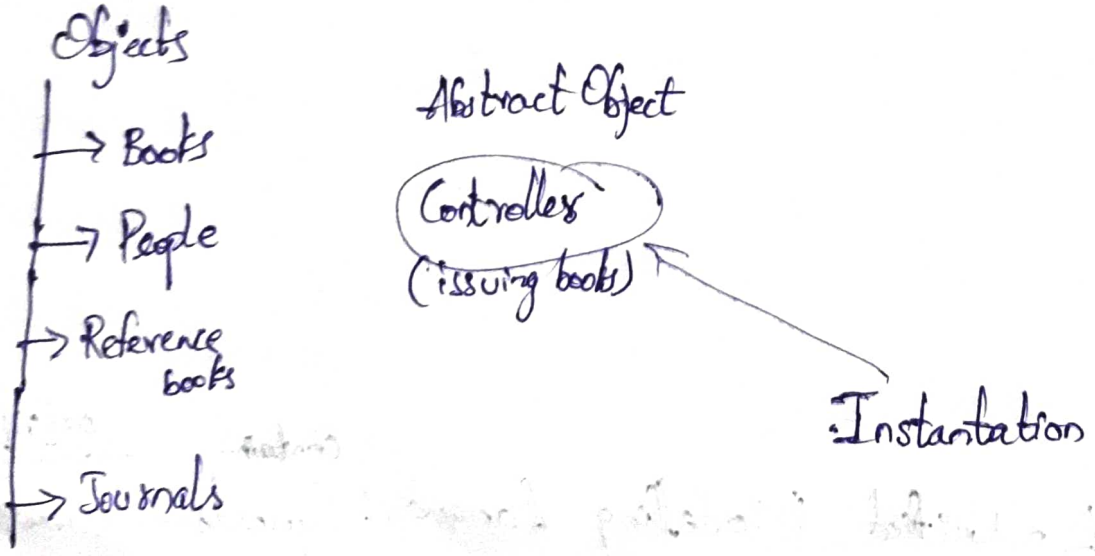
C	OOPS
Sequential	Java, C++
	Procedure / functionality
	Objects, classes / encapsulation

(Abstraction)
(Polymorphism)
(Inheritance)
(data hiding)

Design & Development of Ideas (E.S.)

Concurrent programing

Ex: Library Software



OOPS — Objects + Methods (functions)

UML - Unified Modeling Language

- Contains Symbols & arrows.

- Origin: OMT — 1990
  - Booch — 1991
  - OOSE — 1992
  - Others —

- Current Version is UML 2.0 (0.8, 0.9) (2.5)

Model (vs) Design

- A Design is a Model of a system.
- There can be multiple models for a system.

Requirement Analysis
Model Analysis
Solution ≡ Design

- There can be multiple solutions for a model

Views:

Ex: Banking Software

1. User view
2. Structural View
3. Behavioural View
4. Implementation View
5. Environmental View

Designer's perspective

↳ Methods (through which objects communicate)

UML

Diagrams:

1. Use case diagrams
2. Sequential Diagrams
3. Collaboration Diagrams
4. State Chart Diagrams (related to FSM)
5. Activity Diagrams (Flow charts)

7/2/2023

Things

1. Structural
2. Grouping
3. Behavioural
4. Annotational

Relationships

Factoring Use Cases

3 ways of factoring:

<<include>>
↳ mandatory

<<extend>>
↳ optional

Hardware - Software Tradeoffs

8/2/2023

- Design Consideration : Power, Memory, Performance
- Software techniques to reduce file size:
  - CODEC — Sampling, Compression
    - → Compression & Decompression
  - Lossy
- CODEC examples : VLC, Mx player | MP3, MP4 } wrappers → contains Navigation
  - (Software)
  - (Codec's for audio & video)
- Hardware CODEC ? ADC, DAC
- File System (FS) for Linux, Windows, Mac, Red hat
- 8 GB RAM on MAC = 16 GB RAM on Windows
- APFS - Apple File System

→ Processing Time

→ Man power — # gates/inch = density → Reconfigurable / Freq. of change

→ Memory size → Reliability (Hardware)

Endsem

→ Processing time ... Endsem

→ Man Power

→ Memory size

→ Reliability (Hardware)

→ Reconfigurable / Frequency of change

- keil (& both) emulator, Simulator, Embedded Language

: Difference between emulator & simulator?

(It is a software/H.W that simulates other system.

Non-OS based ES

(1) Out of circuit — dump

(2) ISP_{mode} (In system Programming)

(3) IAP (In Application Programming)

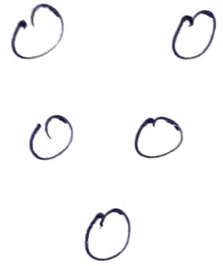
(μC - devices)

RTOS

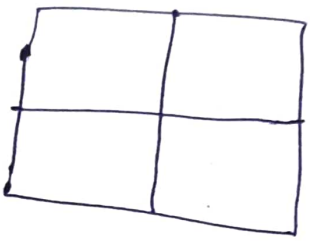
Pegion hole Problem

- Scheduling
- Resource Allocation
- Deadlocks

holes = h



Pegion = p



→ if $\boxed{p > h}$ – there will be a hole which can be occupied by more than 1 pegion.

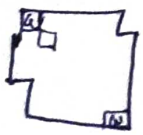
Problem1: A chessboard contains 64 squares. A Biscuit can occupy 2 squares. So, no. of biscuits which can occupy whole chess board is 32.

(1) If one chess square is removed

Biscuits count = (32)

(2) If 2 small squares removed

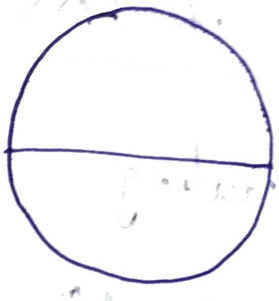
Biscuits count = (30)



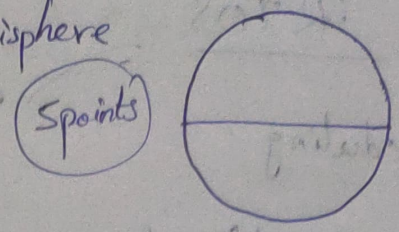
32 - White,
30 - Blacks

Problem 2 :- Globe : Covered Hemisphere

5 points



Problem 2: Globe: Covered Hemisphere



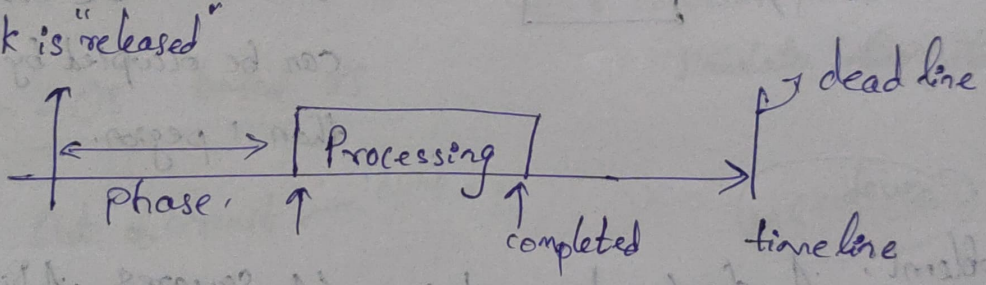
27/02/2022

RTOS

→ Real Time Operating System

- Task, Event
- Job – a unit of work

Ex: Manager
Scheduler



RTOS Vs OS

(1) Time-critical
Ex: (Nuclear) Chemical reactors, Seat belt, Air bags

(2) Response time

(3) Fairness not met (not required)

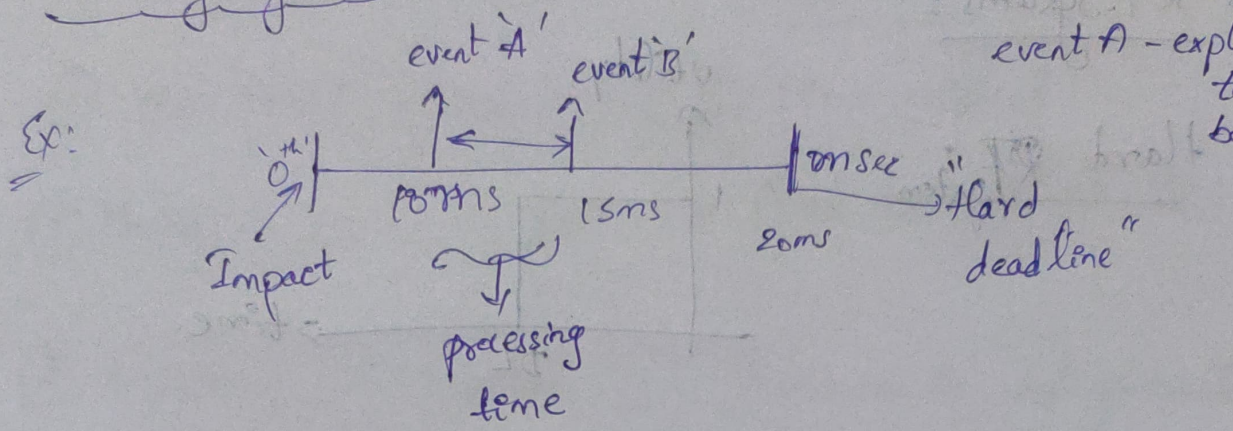
(1) Not time-critical
Ex: Multiuser

(2) No response time

(3) fairness required)

Air bag system:

event B – filling gas
event A – explode the bag

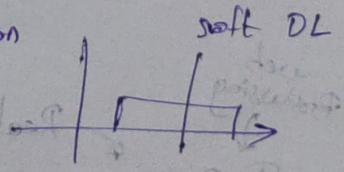


→ Here, Response time is 15 ms
↳ Time where task is completed.

Deadlines:

(1) Hard deadline — Job after deadline is of no use. (E-S)
(2) Soft deadline — Probability; early is better
(3) Firm — DL's are ignored

Soft deadline example: Braking system, (IRCTC) Railway reservation



Hard deadline example: Air Bag system

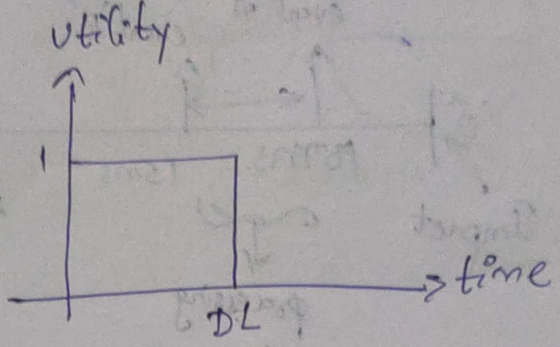
- Based on the deadlines: Hard RT systems, Soft RT systems and Firm RT systems

Firm DL Example: Surveillance, Video conferencing, telemetry

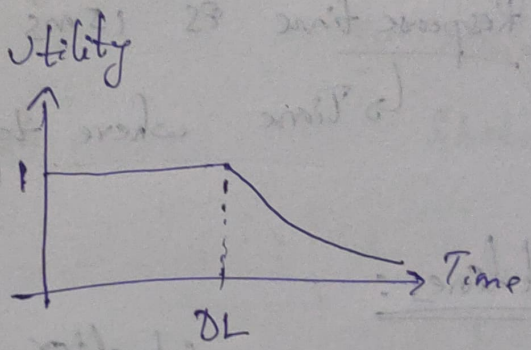
Soft DL example: Web browsing, all interactive application

RT Systems:

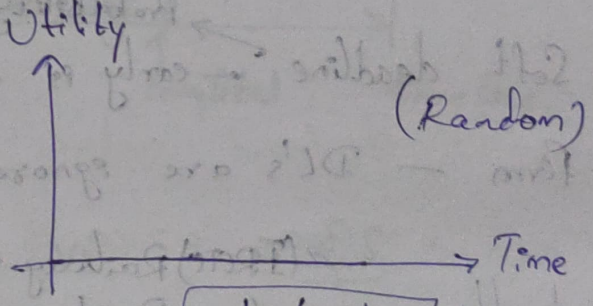
Hard RT system:



Soft RT system:

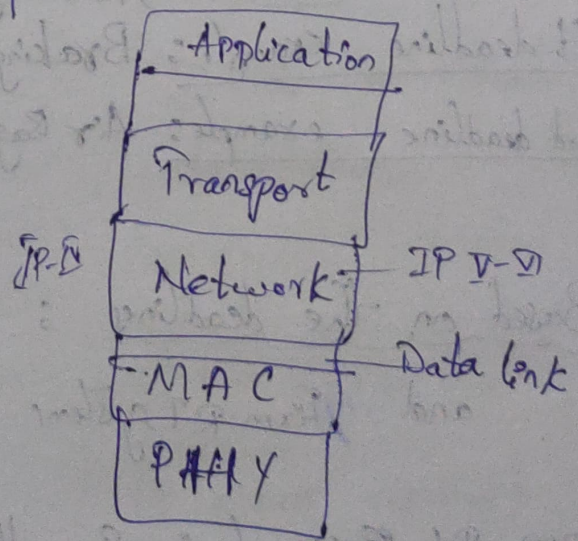


firm RT system:



web Browsing → Real-Time

- TCP, UDP - Protocol
- OSI layers (7)
- TCP - Paket
  Best effort
  Traffics



OS

1. Not time-critical
2. Meeting all DLs not required
3. Fairness is important
4.

RTOS

1. Time critical
2. Need to meet all DL's
3. Fairness is not guaranteed
4. Logically correctness criteria

28/02/2023

Multiple Tasks of RTOS:

→ File systems
→ Memory
→ Networking
→ Interfacing with various I/O
→ Scheduling & buffering I/O

Nomenclature / Lingo: Tasks ≡ Jobs

Why RTOS?

GPOS vs RTOS

Ex: Air bag safety system

Scheduler

- Release time ≡ Schedule time
- Phase time (interval)
- Processing time ≡ Execution time
- Response time
- Deadline (DL)
- Types of DLs
- Types of RTs

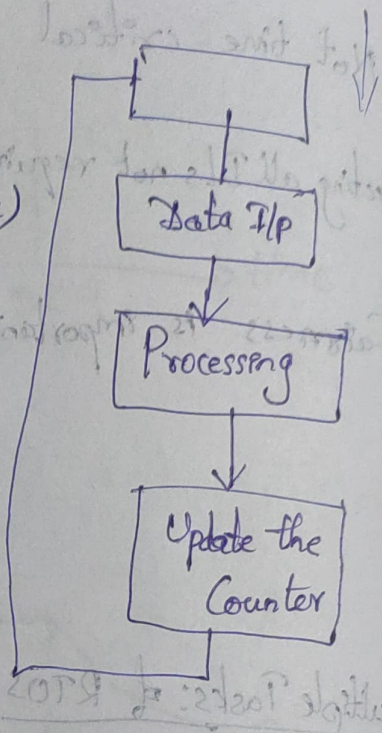
Scheduler

Cyclic executives (fixed timing requirement, say DL = 10msec)

(Round Robin)

→ Simple

→ Memory < 4K



Ex:

Tasks	Time Required (DL)
T_1	20 msec
T_2	100 msec
T_3	50 msec

Multirate

- clock used as interrupt, but here time requirements are different so its called as Multi rate requirements constraints

⇒ Table Driven Scheduler

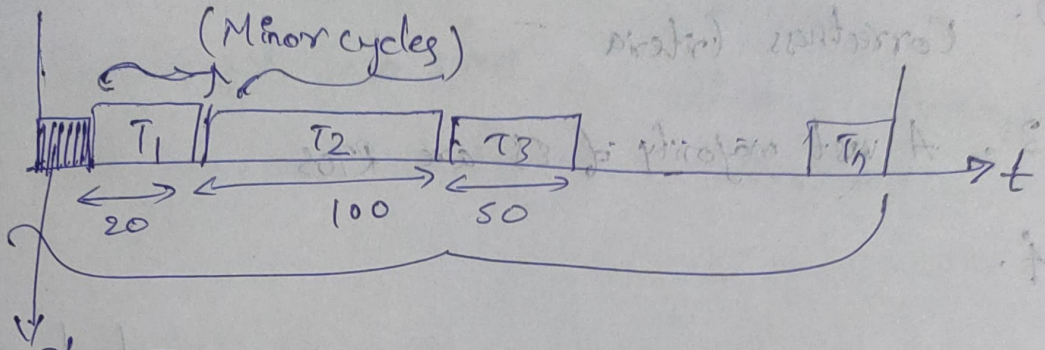
Code:

```
Const int SchedTable size = 10
time_handler()
{ int next time, task current,
    current = SchedTable [entry], tsk
    entry = (entry+1) % size of
    set timer()
    Execute_task (current)
}
```

Overhead: OS → lot of time

for N tasks (No pre-emptive)

Major/Main cycle = LCM($T_1, T_2, T_3, \ldots, T_N$)



Phase I

- Pre-emptive scheduler

Revision:

01/03/2023

- RTS ⊃ ES

Ex: for non-RTS but ES ? - Firm RTS

RTS - Industrial applications: SCADA applications, Laser Printer, MPFI (multi point fuel injection) system,

- Engine (Railway) - 1st E.S.

- Sporadic application :- RTS
  - ↳ Surveillance
  - ↳ Safety critical

RTOS characteristics

1. Time constraint / Deadline
2. Correctness Criteria
3. A vast majority of ES are RTOS
4.

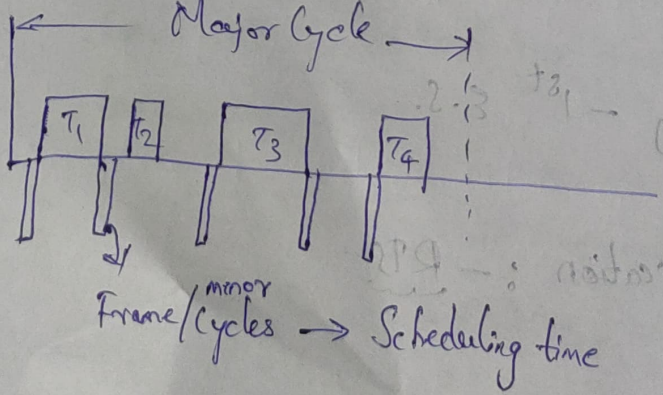
06/03/2023

- Table Driven Scheduler - drawback is OS overhead
  - ↳ Setting up time
- Cyclic Scheduler (clock)

- Task (Execution Time, Period, DL)

$T_i \rightarrow P_i$

Major Cycle — is the LCM of periods (P_i)



Frame / minor Cycles → Scheduling time

- Major Cycle also called as Hyperperiod

Design steps

1. Choose frame size
2. Partition jobs into slices
3. Place jobs / slices into frames

At Frames boundaries

- Cyclic executive performs scheduling.
- There is no preemption within frame.

Cyclic Executive Example

- Job = (computation time, period, relative DL).
- $A = (1, 10, 10)$ $B = (3, 10, 10)$

  $C = (2, 20, 20)$ $D = (8, 20, 20)$

  Two slices $c(D_1) = 2$, $c(D_2) = 6$
- Major cycle = 20 msec
- We select frame size = 10msec
- Frame size: $f \geq \max\{\text{Execution Time } (ET_i)\} \forall i$

Frame size constraints

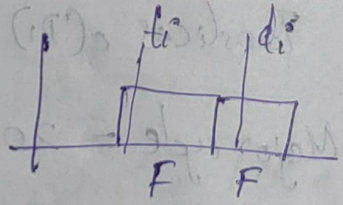
1. Every job needs to start and complete within a frame

   $f \geq \max(T_i)$; $1 \leq i \leq n$
2. Frame size f divides H (the Hyperperiod)
3. B/w release time & DL of every job there is atleast one frame.

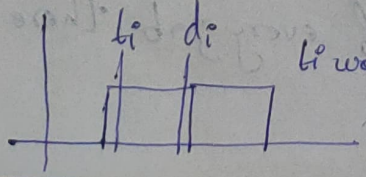
"Selecting an appropriate frame size(f)"

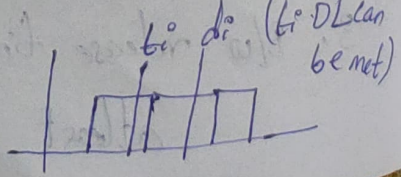
- **Minimum Scheduling overhead & chances of consistency:**
  - → f should be larger than each task size
  - → Sets a lower bound
- **Minimization of Table size:**
  - → f should squarely divide major cycle.
  - → Allows only a few discrete frame size

- **Satisfaction of Task DL**
  - → B/w arrival of a task & its DL: atleast one full frame must exist.
  - → Sets an upper bound
    d_i - relative DL



- If there is not even a single frame: The task would miss its deadline,
- By the time it could be taken up for scheduling, the DL could be imminent.
- The worst case for a task occurs when the task arrives just after a frame has started.





07/03/2023

Criteria:

(1) $f \geq \max\{\text{Ex. Time}_i (T_i)\}\ \forall i$; f - frame size - frame size must include all tasks

(non-trivial)

(2) $2f - \gcd(P_i, f) \leq d_i \quad \forall i$

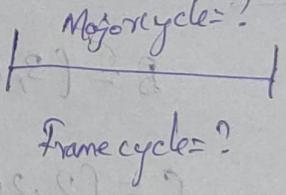
(3) $f * (\text{LCM}(P_i)/f) = \text{LCM}(P_i)$ - frame has to divide squarely the major cycle

Example:

T = (computational time, period, relative deadline)

A = (1, 6, 6) msec

B = (2, 8, 8) msec



Sol: Major cycle = $\text{LCM}(P_i) = \text{LCM}(6, 8) = 24$

Frame size: possible values: $\{1, 2, 3, 4, 6, 8\}$

Case 1: f = 2 msec

Total no. of frames = $\frac{24}{2}$ = 12 frames

Instances of A: $\frac{24}{6} = 4$

Instances of B: $\frac{24}{8} = 3$

Total = 7

No. of empty frames:

Aim: Find max. of 'f' such that we can minimize clock overhead (minimum no. of clock interrupts)

Case 2: f = 4

Total no. of frames = 6

Total no. of instances (A+B) = 7

. Not possible

15/03/2023

Example: A = (1, 10, 10)

B = (3, 10, 10)

C = (2, 20, 20)

D = (8, 20, 20) < D_1 = (2
D_2 = (6
(Slicing the job)

$T_i = (T_i, P_i, d_i)$

Sol: Major cycle = LCM (10, 10, 20, 20) = 20 ms

Possible Frame sizes: {8, 10✓, 20✓
✗

for f = 10: No. of job instances = 2

15/03/2023

Example: A(1, 6, 6)

B(2, 4, 8)

f=3: No. of instances = 8

lite stabs
4 A balls
3 B balls

A B — — — — — —

· Different pattern of assigning 7 jobs (A, B) to 8 frames — $^8C_7$

No. of cyclic schedulers are possible = $^8C_7 \times {}^8C_4 \times {}^4C_3$ (?)

Cons of Cyclic schedulers:

- Inflexible: Difficult to modify and maintain
- As number of tasks increases: It becomes very difficult to select a suitable frame size.
- Fragile: Overrun may cause system to fail
- Difficult to handle sporadic and aperiodic tasks.

Pro of Cyclic Schedulers:

- Simple and efficient

Event-Driven Schedulers:

(Lect 6: Exercises on Timeslice selection last 5 min)

(Lect 7: Event-driven Schedulers)

- Foreground tasks
- Background tasks

→ Real-time tasks are run as foreground tasks.

→ Sporadic, aperiodic and non-real time tasks are run as background tasks.

→ Among foreground tasks, at every scheduling point: highest priority foreground task is scheduled.

Background task completion time:

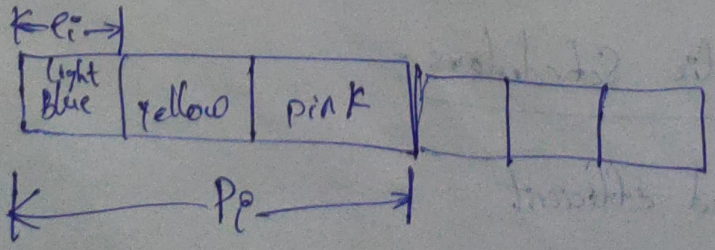
$$ct_B = \frac{e_B}{1 - \sum \frac{e_i}{P_i}}$$

- A background task can run when no foreground task is ready.

- Let T_B be the only background task
- Its processing time be e_B.
- The time for it to complete would be:

$$ct_B = \frac{e_B}{1 - \sum \frac{e_i}{P_i}}$$

— Completion time for the Background task.



Scheduling time
(e_i, P_i, d_i) task for every P_i
↳ repetation of task (finite)

EDF

$i = 1, 2, \cdots, N$ — high priority task for the task which has shortest DL.

- Fraction of time the high priority task engages

$$= \sum \frac{e_i}{P_i}$$

Ex: $e_1 = 50$ msec, $P_1 = 100$ msec, $d_i =$ —

$e_B = 1020$ msec

- Completion time for background task $= \frac{1020}{1 - (\frac{1}{2})}$

$= 1020 \times 2$

$= 2040$ msec.

Scheduling Points for Event-Driven Schedulers:

- Scheduling points: Defined by task completion and arrival events.

- Preemptive schedulers: On arrival of a higher priority task, the running task may be preempted.

- Simplest even-driven scheduler: Foreground – Background scheduler

- Scheduling decisions are made when:
  - → A task becomes ready.
  - → A task completes execution.

- Preemptive schedulers:
  - → When a higher priority task becomes ready any executing lower priority task is preempted.

- Greedy schedulers:
  - → These never keep the processor idle if a task is ready.

Preemptive Scheduling:

- Considering only independent tasks executing on a uniprocessor.
- The two algorithms: EDF, RMA
- RMA - Rate Monotonic Algorithm: is the optimal static priority scheduling algorithm.
- The task priorities once assigned by the programmer do not change during runtime.

- EDF - Earliest Deadline First: is the optimal uniprocessor scheduling algorithm

Event-Driven Dynamic schedulers:

→ The task priorities can change during runtime

→ Based on the relative urgency of completion of tasks. (↑ priority for the least DL)

Uniprocessor - Independent tasks:

→ Tasks do not share resources.

→ There is no precedence ordering among the tasks.

- EDF is the optimal uniprocessor scheduling algorithm.
- At any scheduling point, the scheduler dispatches the task having the shortest deadline among all ready tasks.
- Preempts any task (with longer DL) that might be running.

EDF Schedulability check:

- Sum of utilizations due to all tasks is less than one:

$$\sum_{i=1}^{n} \frac{e_i}{P_i} = \sum U_i \leq 1$$

Example: $T_i = (e_i, P_i, d_i)$

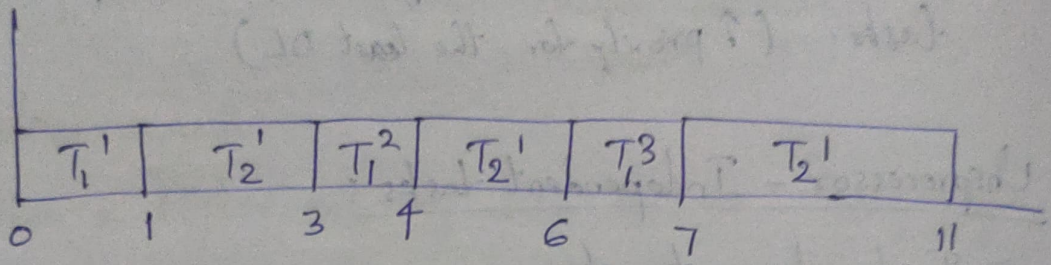
$T_1 = (1, 3, 3)$ & $T_2 = (8, 12, 12)$

Sol: Schedulability check: $\frac{1}{3} + \frac{8}{12} = \frac{1}{3} + \frac{2}{3} = 1 \leq 1$ ✓

→

Sol: $T1 = (1, 3, 3)$ $T_2 = (8, 12, 12)$

$d_1 = 3$ $d2 = 12$



- T1 repeats after every 3 seconds i.e. at 3, 6, 9, ...
- At t = 9sec, T_1 has $d_1 = 3$ & T_2 has $d2 = 2$ & so T2 chosen
- Here T2 divided as [2 | 2 | 2 | 2] ⇒ Total = 8 ms

Example 2: $e_1 = 1$, $P_1 = 4$, $d_1 = 4$ i.e. $T1 = (1, 4, 4)$

$T2 = (1, 5, 5)$ $T3 = (5, 20, 20)$

Sol: Schedulability check:

$$\frac{1}{4} + \frac{1}{5} + \frac{5}{20} = \frac{9}{20} + \frac{5}{20} = \frac{14}{20} = \frac{7}{10} = 0.7 \leq 1$$ ✓

Dynamic

- The long
  higher
  schedu
- EDF

Disadva

→ Poor
→
→ Poo

- Poor tr
  more tim
  arise on
- When a
  is e
  ets
- In EI
  a prior
- The co
  is log

20/03/2023

Dynamic priority:

- The longer a task waits in ready queue, the higher the chance of its being taken up for scheduling.
- EDF is simple & optimal, not very efficient.

Disadvantages of EDF:

→ Poor transient overload handling.

→ Runtime inefficiency.

→ Poor support for resource sharing among tasks.

- Poor transient overland handling: A task might take more time that estimated and too many tasks might arise on some event.
- When an executing task takes more time, in EDF: it is extremely difficult to predict which task would miss its deadline.
- In EDF implementation, tasks need to be maintained sorted in a priority queue based on task deadlines is required.
- The complexity of inserting a task into a priority queue is $\log(n)$; n - is the number of tasks.

MLF : A variation of EDF

- Minimum Laxity First
- Laxity = relative deadline – time required to complete task execution.
- The task that is most likely to fail first is assigned highest priority.

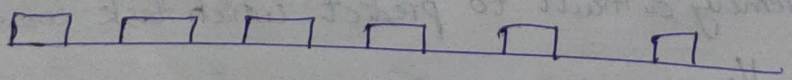
(e_i, P_i, d_i) Lower P_i – high Priority
higher P_i – Low Priority

- EDF = MLF when tasks have equal execution times and deadlines.

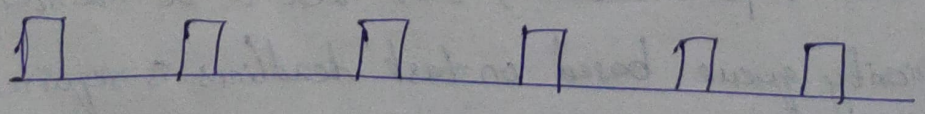
Rate Monotonic Schedulers:

- Process priority determined by arrival rate (since rate = $\frac{1}{\text{Period}}$)

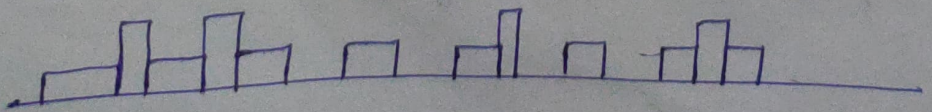
Process 1: higher priority



Process 2: Lower Priority



Preemptive Scheduling



- If RM

no othe

- Schedulab

UB-I :

→ Sum

$\sum_{i=1}^{n} \frac{e_i}{P_i}$

UB-II :

→ $\sum U$

- For a

$\sum u_i \leq$

RMA :

Example :

$T_1 =$

Sol: Sched

I & II

- If RMA cannot schedule a set of periodic tasks then no other static priority scheduling algorithm can.

- Schedulability tests : Utilization bounds.

✡ UB-I :

→ Sum of U_i due to tasks is less than one

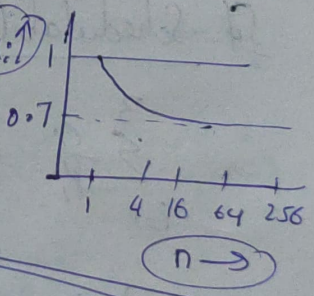
$$\sum_{i=1}^{n} \frac{e_i}{P_i} = \sum U_i \leq 1$$ (necessary but not sufficient condition)

✡ UB-II : (Liu & Layland 1971)

→ $\sum U_i \leq n(2^{1/n} - 1)$; n - no. of tasks

- For a very large no. of tasks i.e. $n \to \infty$

$$\sum U_i \leq \lim_{n \to \infty} n(2^{1/n} - 1) = \ln 2 = 0.693$$



RMA :

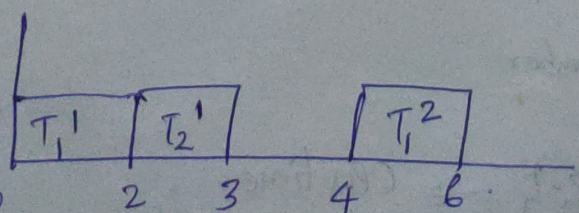
d_i

Example : $T_i = (C_i, P_i)$

$T_1 = (2, 4)$ & $T_2 = (1, 8)$

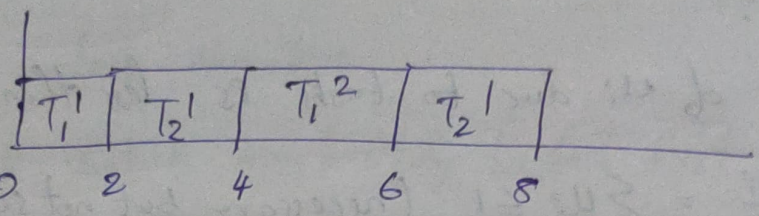
Sol: Schedulability check :

(I) & (II) $2/4 + 1/8 = 0.5 + 0.125 = 0.625 \leq 2(\sqrt{2} - 1) = 0.82$



Example2: $T_1 = (2,4)$ & $T_2 = (4,8)$

Sol $2/4 + 4/8 = 0.5 + 0.5 = 1.0 > (2)(\sqrt{2}-1) = 0.82$



- Some tasks sets that fail Liu-Layland may be schedulable under RMA → Liu-Lehoczky test.

Example: $T_1 = (1,4,4)$ $T2 = (2,6,6)$ $T3 = (3,20,20$

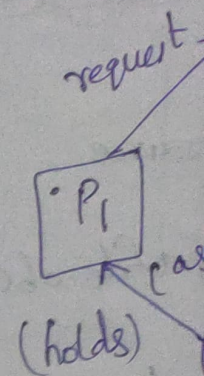
Sol. Schedulability check:

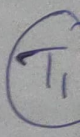
$$\sum_{i=1}^{3} \frac{c_i}{P_i} = \sum U_i = \frac{1}{4} + \frac{2}{6} + \frac{3}{20} = \frac{44}{60} = \frac{11}{15} = 0.733 < 1$$

$$n(2^{1/n} - 1) = 3(2^{1/3} - 1) = 0.778$$

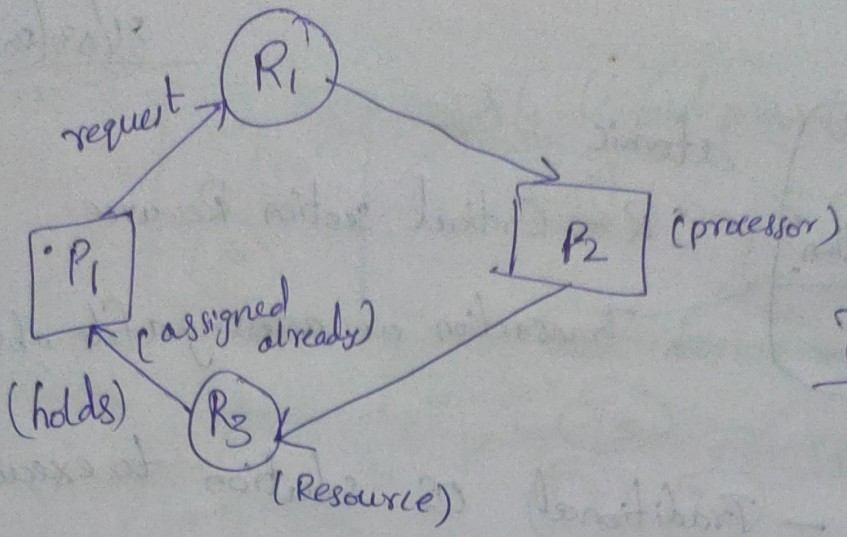
$$\sum U_i = 0.733 < 0.778$$

- RMA : Resource sharing
- T_x^y ← instance (superscript)
  T_x ← task number
- Resources $\{R_i\}$ → CPU time
  → Data structure time
  $\{T_i\}$

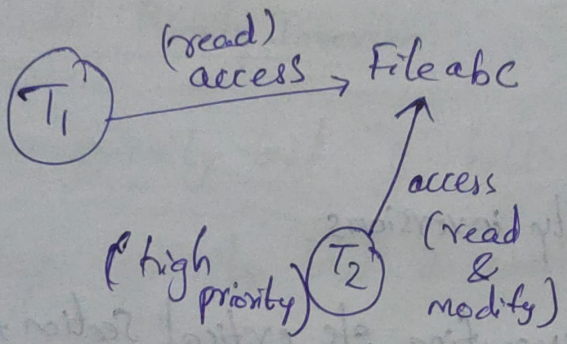




- RMA
- Priority
- Non-
Ex: files
- A piece
resour
in th
→ Sem
to



Deadlock



inconsistence

- RMA Performance ? Difficulties ?
- Priority inversion?

21/03/2023

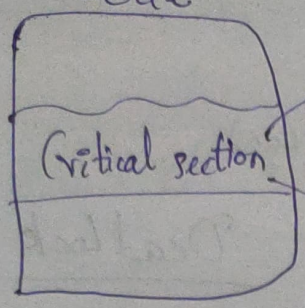
- Non-premptable Resources ⇒ Critical sections

Ex: files, data structures, devices etc.

- A piece of code in which a shared non-premptable resource is accessed is called a Critical Section in the operating systems literature.

→ Semaphores are the traditional OS solution to execute critical sections.

21/03/2023



atomic → T_{CR}

CR – Critical section Resource

Transaction or accessing a file abc

"Semaphores" – Traditional OS solution to execute Critical sections.

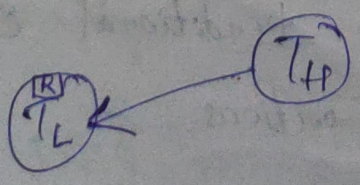
→ Priority inversions

→ Unbounded Priority inversions

- A task instance executing its critical section, cannot be preempted – Priority Inversion.

Consequence: A higher priority task keeps waiting, while the lower priority task progresses with its computations.

- When a resource needs to be shared in the exclusive mode.
- A task may be blocked by a lower priority task which is holding the resource.



Reduce time for CR

T_{CR} ↓

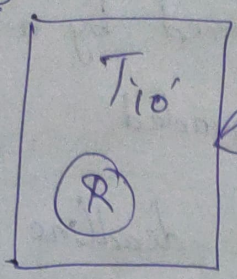
RTOS

Unbounded

- A low pr
- A high
- Intermedia
the res
task fro

T1
T2
T3
T4
T5
T6

RTOS



T_1 (highest priority)

wait

Ex: Mars Pathfinder

Task doesn't require R

(Intermediate tasks) T_5 ✓ execute

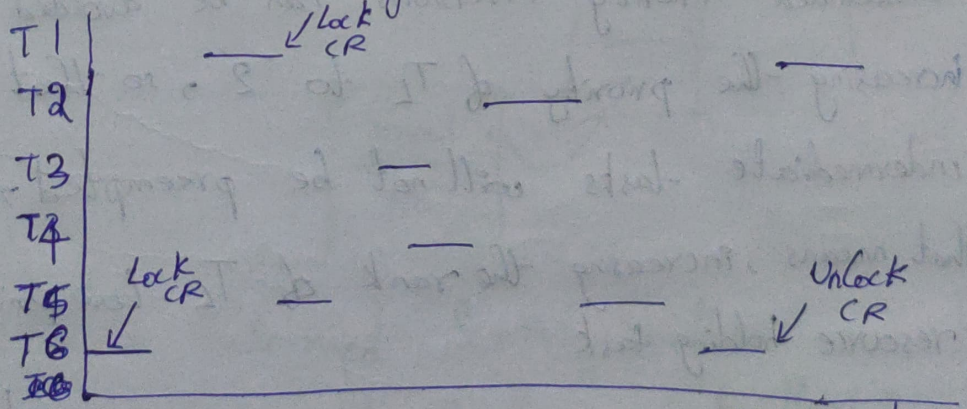
T_3, T_6

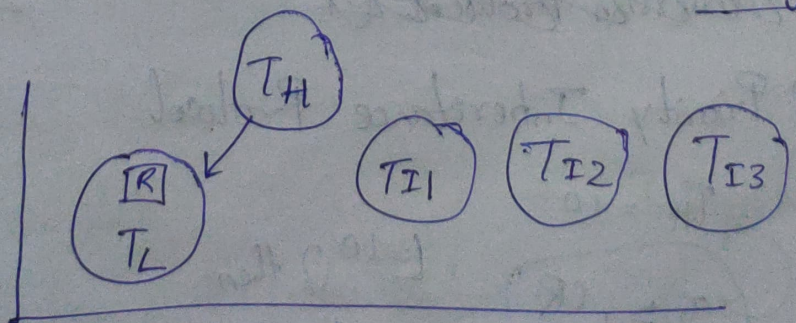
Unbounded Priority Inversion

Solution: Increase Priority of T_{10} to 2 so T_5, T_3, T_6 will not be preempted (increase Rank)

Unbounded Priority Inversion

- A low priority task is holding a resource.
- A high priority task is waiting
- Intermediate priority tasks which do not need the resource repeatedly preempt the low priority task from CPU usage





#Chapter4

- No. of priority inversions suffered by a high priority task : can become unbounded
- A high priority task miss its deadline.
- In the worst case, the high priority task might have to wait indefinitely.
- Unless handled properly, priority inversions by a critical task can be too many causing it to miss its deadline.
- Most celebrated example is Mars path finder.

Solution:

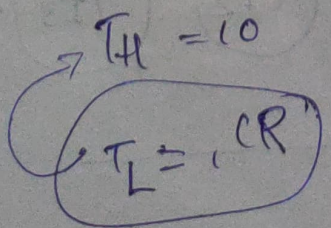
→ Priority inversions can be avoided by reducing (optimize) critical section resource time. ($T_{CR}\downarrow$)

→ Unbounded Priority inversions can be avoided by increasing the priority of T_L to "2", so that the intermediate tasks will not be preempted.

- That means, increasing the rank of T_L - Low priority resource holding task.

27/03/2023

- ~~Priority Inversion Protocol~~
- PIP : Priority Inheritance Protocol



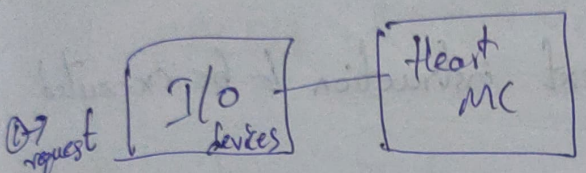
$L = 10$ then
$L = 1$

priority entity?

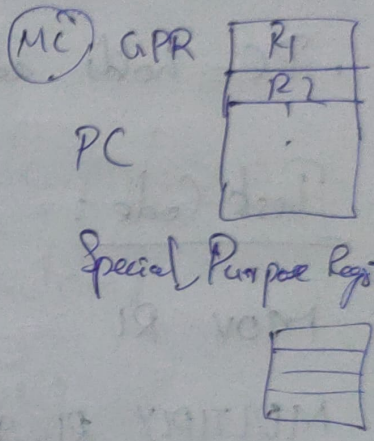
#Chapter4 From David Book. An Embedded software primer (276 pages) 27/03/2023

Interrupts

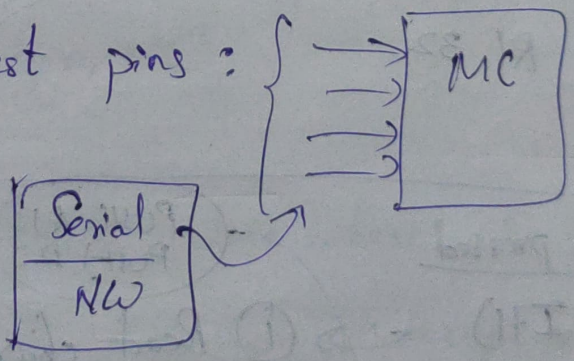
- System level Subroutines
- Why interrupts needed? for pre-emption



N/w card
Serial I/O
keyboard



- I/O devices have Interrupt Routines
- Interrupt request pins :



- ISR : Interrupt Service Routines
- IH : Interrupt Handler, handler()

- Assembly Code
- Assembler
- Interpreter
- Compiler : HLL to LLL

- executes each code one at a time

- **Assembler** : Human readable code
  Single assembly code → Single instruction

- **Compiler** : '.c' code → .object
  Single instruction → Multiple Instruction

- PC - holds address of next instruction to be executed

Task Code :

```
MOV R1
MULTIPLY R1, 9
DIVIDE R1, 5
ADD R1 32
:
```

Temperature °C to °F

when key is pressed
IR (IH)

PUSH R1
PUSH R2

① Read character
② Store R1
③ Enabling Interrupt request

POP R1
POP R2

Restore the status back

RETURN

- nested ISR : add on features of μC
- where these IR(I-H) stored ?
- In 86-family has special name for 'RETURN' (Intel)
- Syntax of subroutines in assembly code:

CALL() , RETURN()

Operations that an assembly code performs :

→ ① Arithmetic operations (Accumulator)
② Conditional Jumps (JCOND)
③ Unconditional Jumps ④ Subroutines

Ex: SUBRACT R1, 3
JCOND Zero, NO

- Interrupt vector table : place where IR is stored
- Based on the family of processor, interrupt vector table is stored

for family 86 → 0x0003
80186 → 0x0000
different family → predefined location

- Need to see the datasheet
- μP or μC do not execute an interrupt in the middle of a current executing task.

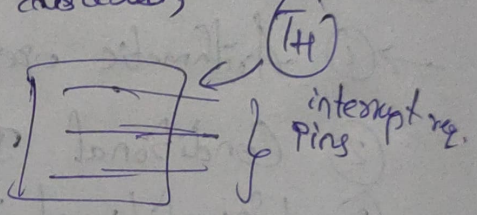
Zilog Z80 ⟶ X86

- Only in moving a large file, interrupt is executed

  Mov file to location

- Based on priority, 2 interrupt tasks are executed
- nested interrupts
- If interrupt request pins disabled,

  Time stamp
  Priority



- If interrupt request pins are disabled & they are not enabled back ⇒ system is in fault.

28/03/2023

- Assembly code :- executes each code one at a time
  - atomic part
  - TH ISR – Interrupt (can't happen)

- Example: (Code) (TH) Nuclear reactor: To ≠ To (what) normal function Buzzer

```
static int iTemperature[2];

void interrupt vReadTemperatures(void)
{
    iTemperature[0] = !! read in the value from hardware
    iTemperature[1] = !! read in the value from hardware
}
void main(void){
    int iTemp0, iTemp1;
    while(TRUE){
        iTemp0 = iTemperature[0];
        iTemp1 = iTemperature[1];
        if (iTemp0 != iTemp1)
            !! set off howling alarm;
    }
}
```

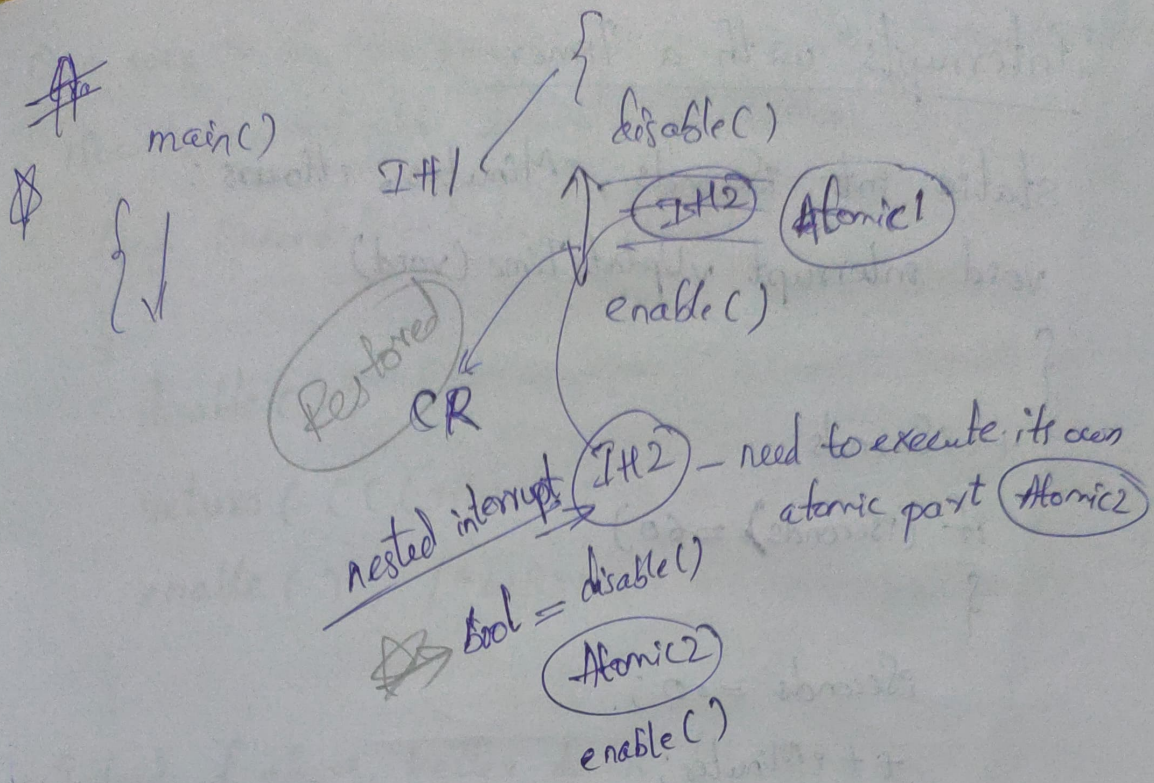
T=10ms

```
MOVE    R1 (iTemperature[0])
MOVE    R2 [iTemperature[1])
SUBTRACT R1, R2
JCOND   ZERO, TEMERATURES_OK
; Code goes here to set off the alarm
TEMPERATURES_OK:
```

Solution:

```
static int iTemperatures[2];

void interrupt vReadTemperatures(void)
{
   iTemperatures[0] = !! read in value from hardware
   iTemperatures[1] = !! read in value from hardware
}

void main(void)
{
   int iTemp0, iTemp1;

   while(TRUE)
   {
      disable(); /* Disable interrupts while we use the array */
      iTemp0 = iTemperatures[0];
      iTemp1 = iTemperatures[1];
      enable();

      if (iTemp0 != iTemp1)
         !! set off howling alarm;
   }
```

- nested interrupts?
- PIP



- What is meaning of Static? Volatile? Const? (keywords) (In C or C++)

Static variable (Scope/Environment)

Volatile

Const:

Disabling interrupts in Assembly Language:

```
    .
    .
    .
    DI      ; disable interrupts while we use the array
    MOVE    R1, (iTemperature[0])
    MOVE    R2, (iTemperature[1])
    EI      ; enable interrupts again
    SUBTRACT R1, R2
    JCOND ZERO, TEMPERATURES_OK
    .
    .
    .
    ; Code goes here to set off the alarm
    .
    .
    .
TEMPERATURES_OK:
    .
    .
    .
```

Interrupts with a Timer

```
static int iSeconds, iMinutes, iHours;
void interrupt vUpdateTime (void)
{
  ++iSeconds;
   if (iSeconds) == 60)
   {
      iSeconds ==0;
      ++iMinutes;
      if (iMinutes >=60)
      {
         iMinutes == 0;
         ++iHours;
         if (iHours > -24)
            iHours=0;
      }
   }
   // Do whatever needs to be done to the hardware
}
long iSecondsSinceMidnight (void)
{
  return(((((iHours * 60)+iMinutes) * 60)+iSeconds));
}
```

. One way to fix this problem is to disable interrupts while lSecondsSinceMidnight does its calculation.

```
long lSecondsSinceMidnight (void)
{
  disable();
  return (((((iHours * 60) + iMinutes) * 60) + iSeconds));
  enable(); /* WRONG: This never gets executed! */
}
```

. Instead of above, better do it like this:

```
Long lSecondsSinceMidnight (void)
{
  long lReturnVal;
  disable();
  lReturnVal == (((((iHours * 60) + iMinutes) * 60)
                                    + iSeconds;

  enable();
  return (lReturnVal);
}
```

- If the μP's registers are too small to hold a long integer, then assembly language could be something like:

```
MOVE   R1,(lSecondsToday)  ; Get first byte or word
MOVE   R2,(lSecondsToday+1) ; Get second byte or word
  .
  .
  .
RETURN
```

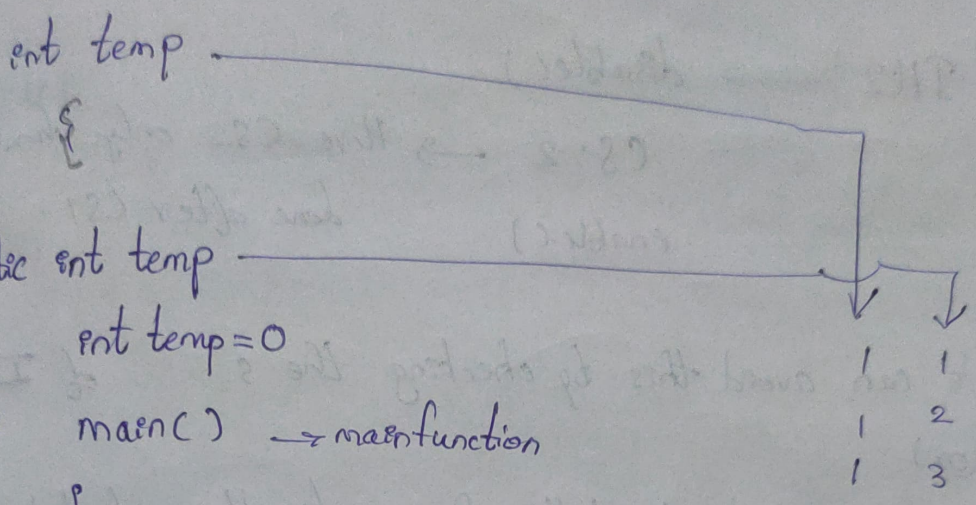
Disabling and Restoring Interrupts

```
long lSecondsSinceMidnight (void)
{
 long lReturnVal;
 BOOL fInterruptStateOld; /* Interrupts already disabled? */
 fInterruptStateOld == disable();
 lReturnVal == (((iHours * 60) + iMinutes) * 60)
                                    + iSeconds;
 /* Restore interrupts to previous state */
 if (fInterruptStateOld)
   enable();
 return (lReturnVal);
}
```

Static S. Variable in C

29/03/2023

Difference b/w



→ main function

→ new function called by main

```
func1()
{
  ++temp;
  printf()
```

→ So if we use int temp=0 then the value of the temp is erased every time the function is read

→ But if we use static int temp=0, the static will store the value of temp in cache.

→ The shared data is a problem for nested interrupts.

→ for example when we are checking the interrupts

IH1 ⟶ disable()
C section 1
enable()

IH2 ⟶ disable()
CS:2 ⟶ Here CS2 only should be done after CS1
enable()

- We can avoid this by checking the state of IH1.
(or)
- We can avoid it wholly if we directly calculate the new seconds and just call the return function in one function.
- But the problem is that it only works for 32-bit microcontroller.
- A 16-bit processor will need more GPC's which will be effected by interrupts.
- So instead we read the data twice to double check whether the value is effected by interrupt or not.
- But here we can't just use static function we need.

Fig 4.10

```
Static volatile int temp=0
    static int buttonprenod=0
        main()
        {
        while (buttonprenod==0)
```

```
printf ("button was pressed")
return;
}

ITH1
  enterrupt button1
  {
    ++ button pressed;
  }
```

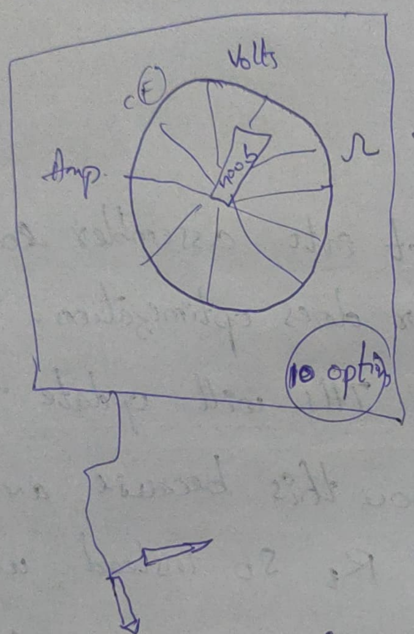
- Here compiler converts it into assembler code but before that the compiler does optimization.
- The problem is that the ITH1 will update the R_1.
- The compiler doesn't know this because and will check the cache but not R_1. So instead we use
- Static Volatile int button pressed = 0 this helps ; will making the compiler check Register instead of cache.

OS Architectures

03/04/2023

1. Round Robin → Sequential fashion
2. Round Roben with Interrupts
3. Function queue scheduling architecture
4. RTOS

Digital meter



→ What OS?

→ simple
→ No time constraints.
→ No DL's
→ No need of interrupts

(1)

main()
while (true)

(Task i = True)

{ if (Task A == True)
if (Task B
(
{ (
(

(10 tasks)

= 100msec

- read
- perform calculations
- Display

→ worst delay
Task has to wait for 10msec

→ Manufacturer make sures the time taken to ~~...~~ move the nob is atleast more than 1sec, so that there is no delay time & response time is low i.e. response is quick

- Digital meter

→ Simple

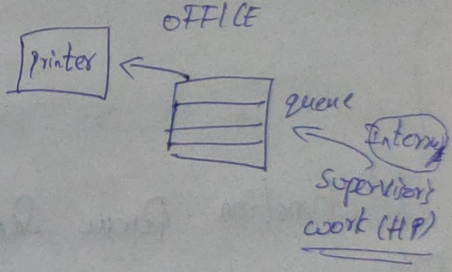
→ Not a RTOS (no timing constraints)

→ No need of interrupts

Printer:

→ Simple → There are deadlines → need of interrupts

- Interrupt use cases : for stopping the print, when paper is jammed
- In case of N/w printer



- Stand alone
- Global variable, all are initialized to FALSE

(2)

```
main()
while (TRUE)
{
   if(A)
   if(B)
   if(C)
}
```

BOOL Task = True

Interrupt subroutine A
" B
" C

{ Task == FALSE
do Job A;

③ Function queue scheduling architecture

· There is nothing in the main function

05/03/2023

RTOS — Why
What's the difference from the above architectures

Applications:

ES
① Round Robin Ex: TDM, Digitalmeter
② " " with Interrupts
Ex: N/W printer

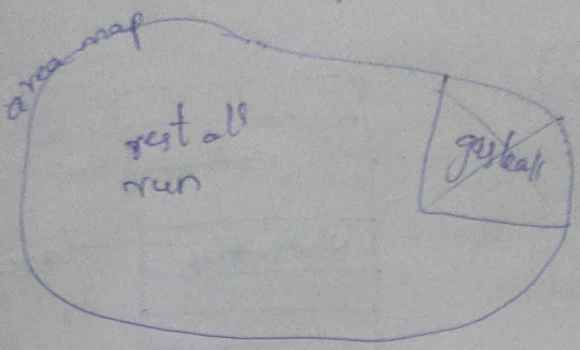
③ Function Queue Scheduling → no deadlines
→ Response time / Latency
Ex: CRO

RTS
④ RTOS Ex: Printer, Bar Code Scanner, Gasoline tank
→ Time critical
→ Deadline

· Latency refers to interrupts

Ex: Factory — gas.

ISP = MC



Interrupt

gas leak()

Store current states to stack

disable()

CR

enable()

restore states

CR = ① Shut down
② Calculations which part
③ Resume normal function (or) remaining part
④ Emergency phone call – sec to min.

(critical section resource)

④ ⇒ delayed

- The period of time it takes to execute any interrupt routines for interrupts that are of higher priority than the one in question.
- Optimization is needed at programming side.
- Solution for P.I (priority inversion), reduce T_{CR}

⇒ reduce latency.

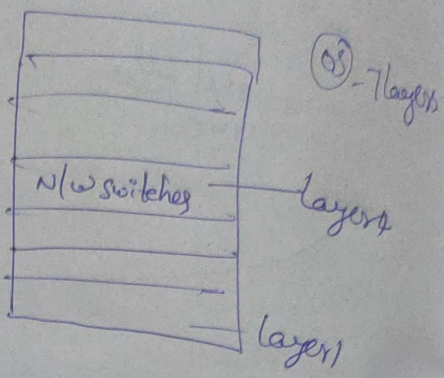
Alternatives for Disabling interrupts

• Round-Robin with Interrupts example : A simple bridge (communication)

→ Routers (layer '4')

→ Bridges

→ ethernet hub



RTOS:

→ no sequential order

→ Based on event/task scheduler & Priority

- Dynamic event scheduler
- Static event scheduler

10/4/2023

• Response time is the key to choose the architecture

• Round-Robin - Worst Response Time

• RTOS - Zero execution time for interrupt routine

Ex: App ⟷ RTOS (data) tightly bounded

• windows, MAC - not tightly bounded

PG no - 13

RTOS examples : Vx.

RTOS

① Linking OS

②

③ Memory part

Ex: App Configuration features, Linking OS

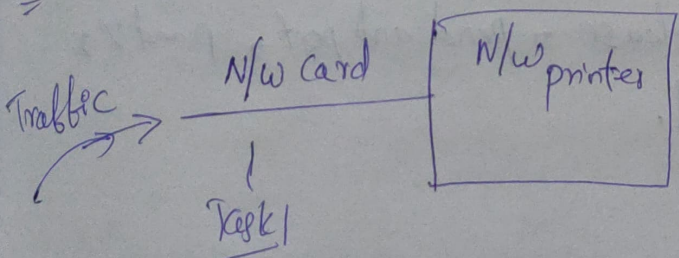
whatsapp Scheduel)

Linking OS

6.1 Task & Task states:

- Blocked state : a particular task waiting for an external event to happen.
~~(other than interrupt)~~

Ex.



blocked state

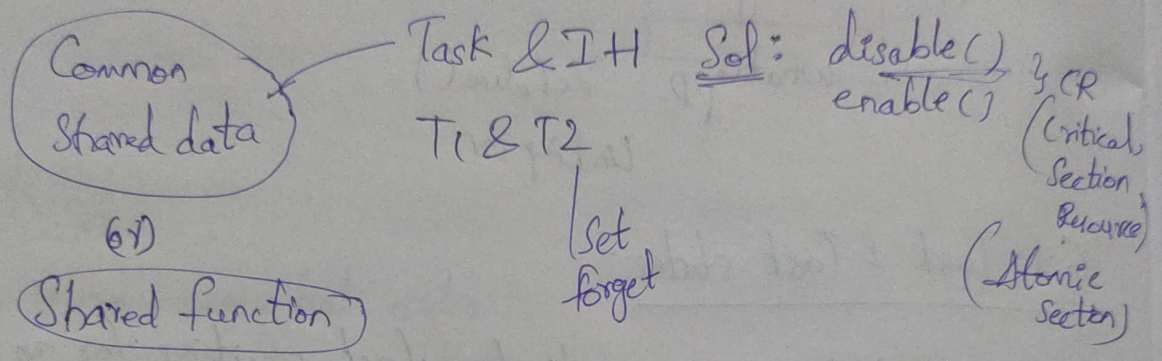
- Ready state :

Ex: Petrol Bunk

Reentrancy:

11/04/2023

- Reentrant functions are functions that can be called by more than one task & that will always work correctly.



- Signalling mechanism – Semaphore – hold, take, release
  → give, take and release → pend and post, pend V, wait and signal.

Semaphore problems:

→ Forgetting to take semaphore, Forgetting to release the semaphore, Taking the wrong semaphore, holding a semaphore for too long.

"Semaphore" A, B, C
↓ ↓
CR1 CR2

Objective
↓
CR execution
↓
min. no. of semaphores

Ex: Nuclear reactor Semaphore. (pg-158)

pg-148 Exampl

→ mutex; b
a

pg-169, 170

- RTOS – OS
- NU – Na
- The fu
with 'N

Ex: The N
(4 semaphore

Message

Ex: Task
Task 2

Tasks
as Task

pg-148 Example: where take Semaphore & releaseSemaphore need to be placed?

→ mutex, binary semaphore & counting semaphore are among most common semaphore variants that RTOS's offer.

pg-169,170 - example problems ☆

- RTOS - OS
- NU - Nucleus - atomic } used exclusively for Reentrant functions
- The functions & data structure whose names begin with 'NU' are those used in an RTOS called Nucleus.

Eg: The NU_SUSPEND parameter to the NU_Obtain_Semaphore

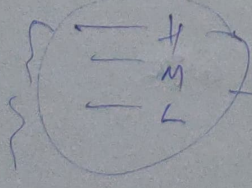
(Semaphore variants - not there in the syllabus)

12/04/2023

Message Queues, mailboxes and pipes: (RTOS)

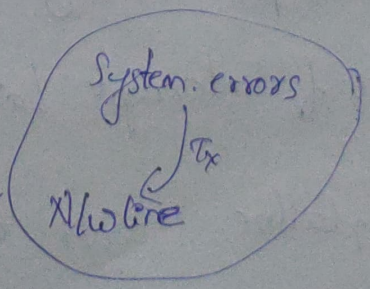
Ex: Task 1 — Calculating petrol tank level

Task 2 — Display



Sub tasks

. By default reentrant



Tasks as Task 6

(*: through N/w line it takes time)

- Inheritant services are provided by the RTOS like Queues, Mailboxes, Pipes.

- Mailboxes: to have large information from 1 task to the other
- Pipes: refer example
- Queues: (just the size is different)
  ?
- Timer functions (7.2).

(7.3, 7.4, 7.5 - covered in nptel lectures)

(7.4 in nptel lect. videos)

- Semaphores
- IH - Nested interrupts (pg-100 in pdf) (120 in book)
- Nu

4 ⇐ Nested interrupts Chapter

RTOS
- Semaphore Nomenclature - give, take, pend, post
  (pg-155 in pdf) (175 in book)

- ✗ No Interrupt Latency - calculation wise
- Definition of Interrupt Latency ✓
- Malloc, Calloc in cpp not supported by RTOS (Memory management)

- for What reasons Semaphores are used?
- Types of Semaphores

K.V.

- Books: Shibu - Introduction to ES

Computer Organization & E.S - Carl Hamacher (in library)

Whisper playgrounds

- Pigal: Bitcoin mining hardware
  Bitcoin wallet hardware

Books: ① A Primer on software for ES ✓ Dard

② Rajib Mall ✓ - RTS - Theory & Practice

Lecture 4: (Cyclic Scheduler) - NPTEL

End exam - 50M

- Post mid → (30-35)
- Pre mid → (15-20)

- EDF, RMA, PIP, Cyclic schedulers
- identify bugs in the program ⇕ (static, volatile) (identify flaws, bugs)
- Pseudocodes (2M)
- Descriptive → 2M
- No 1M questions
- Application is given, come up Architecture
  Ex: Traffic lights, Petrol station
- EDF & RMA algorithms
- PI & Unbounded PIs

Syllabus: (David book)

Chapters: 4, 5, 6, 7 (partial) 7.1, 7.2

Videolec: 1-10, 14, 15 (7.3, 7.4, 7.5)