Embedded Systems & IoT Design.

Unit - 1

8051 MicroController.

MicroController for an Embedded System - 8051 - Architecture - Addressing Modes - Instruction Set - Program & Data Memory - Stacks - Interrupts - Timers/Counters - Serial Ports - Programming.

① MicroController for an Embedded System.

→ Embedded s/m is a Combination of h/w & s/w designed for specific function.

→ It is a Computer s/m which is a Combination of Computer processor, Computer Memory, i/p & o/p peripheral devices that has dedicated fun^{n} within a larger Mechanical or electronic s/m.

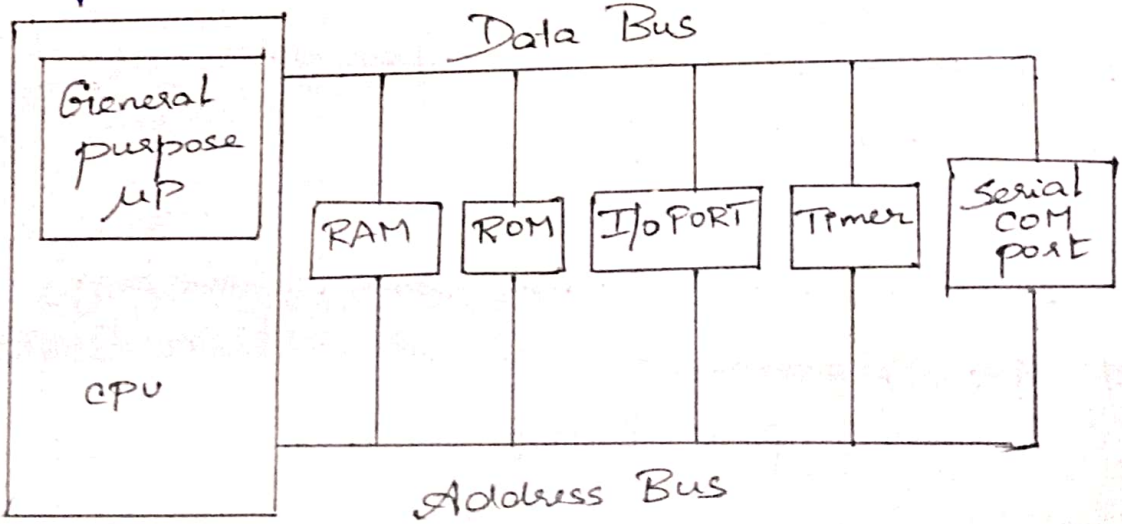
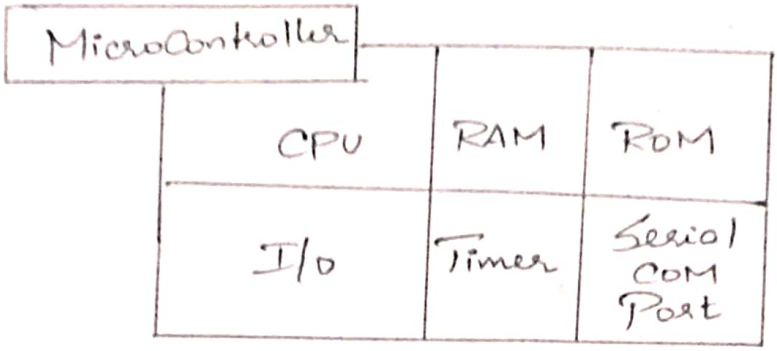


Fig - General-Purpose µP s/m.

→ µP & µC do one & one task only.

→ Only one appl^{n} s/w is typically burned into ROM.



Ex- Printer. (Ex of Embedded s/m - Processor inside performs only one task - getting data & printing.

Applications of Embedded product using µc.

① Home.

→ Telephones, Security s/m, Computers, TVs, Cable Tv tuner, VCR, Remote Controls, Videogames, Cellular phones, Sewing Machines, Camera, toys, etc.

② Office.

→ Telephones, Security s/m, fax Machines, laser printer, Color printer, paging.

③ Auto.

→ Trip Computer, Engine Control, Air Bag, Security s/m, Transmission Control, Climate Control, Cellular phone, Keyless Entry.

→ One of the Critical needs of an Embedded s/m is to decrease power Consumption & space.

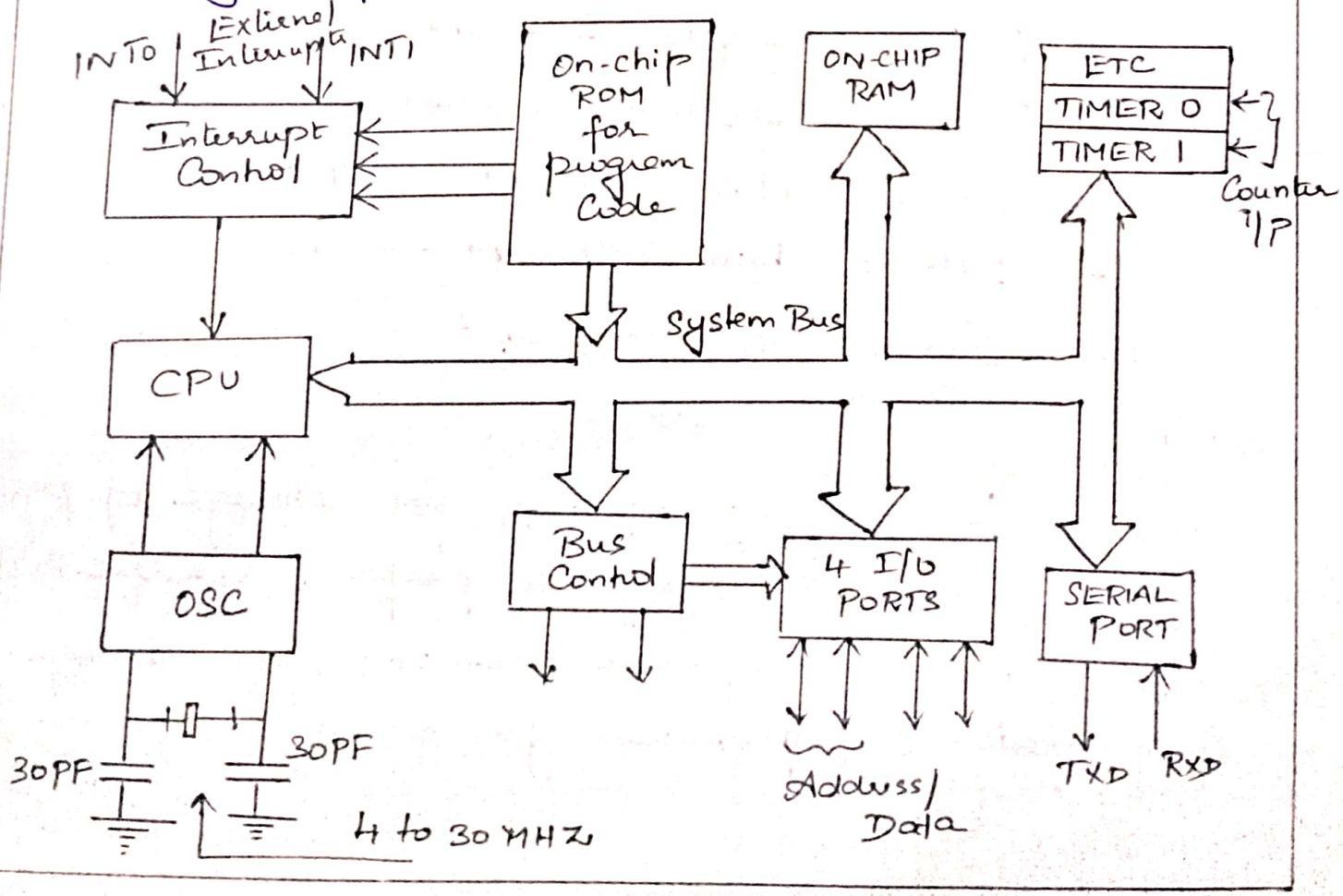
II) 8051 Architecture :-

→ It is 8 bit µc with two pins Dual Inline Package (DIP), 4K byte of on-chip ROM storage, 128 bytes of RAM storage, two 16-bit timers, one Serial port & 4 four ports all on Single chip.

→ It Consists of 4 parallel 8 bit ports which are programmable as well as addressable.

→ All supporting devices are Connected to CPU by using 8/m bus which Consists of 8 bit data bus, 16 bit address bus & bus Control signals.

→ All other devices like program memory, ports, data memory, Serial interface, interrupt Control, timers & CPU are all interfaced together through 8/m bus.



Functional Blocks :-

(i) CPU (Control Processing Unit)

→ It Synchronizes & manages all processes that are carried out in μc.

→ It has no power to control the functioning of CPU.

→ It interprets the program stored in ROM.

→ CPU manages different types of Registers.

(ii) Interrupts:

→ It is a Sub-routine Call that is given by μc when some other pgm with high priority is requesting for acquiring the sys bus.

→ Interrupts delay the current process.

Types of Interrupt :-

(i) Timer 0 Overflow interrupt — TF0

(ii) Timer 1 Overflow interrupt — TF1

(iii) External h/w interrupt — INT0

(iv) External h/w interrupt — INT1

(v) Serial Com° interrupt — RXD/TXD.

(iii) Memory.

→ It uses chip memory for storage of pgm which guides the μc to perform specific tasks.

→ μc also required memory for storage of data & operands for short duration.

(iv) Bus.

→ It is a group of wires which uses as a Communication channel to data transfer.

→ Two types of buses are used in 8051.

1) Address Bus
2) Data Bus.

① Address Bus.

→ It Consists of 16 bit address bus which is used for transferring data from CPU to memory.

② Data Bus.

→ It Consists of 8 bit data bus which is used for transferring data from peripherals to other peripherals.

(v) Oscillator.

→ To perform timer operation inside µc it required externally Connected on-chip Oscillator.

→ 8051 uses 2 16 bit Counters & Timers.

→ Oscillator is used inside µc to perform timer & Counter oper^{n}.

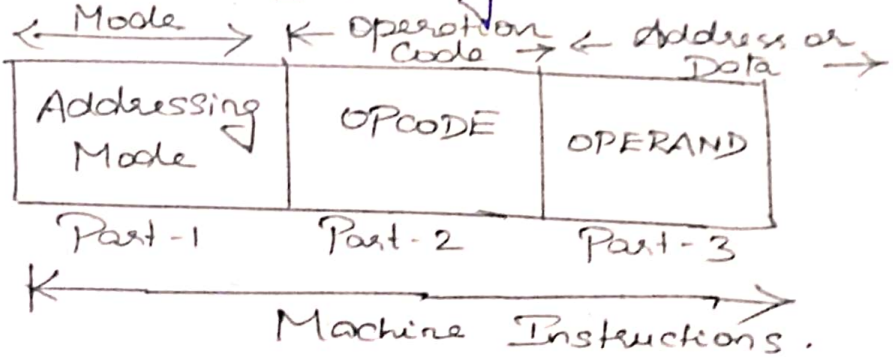
III) Addressing Modes:

→ CPU can access data in various ways - addressing modes.

→ It is a way to address an operand.

→ Operand means source data that is the data operating upon.

→ It can be a direct address of memory (or) register names (or) any numerical data.



→ Each Assembly language is split into opcode & an operand.

→ Opcode → Instruction executed by CPU

→ Operand → Data or memory loc^n used to execute that inst^n.

Types:-

→ Immediate addressing Mode
→ Register direct addressing mode
→ Direct addressing mode
→ Register indirect addressing mode &
→ Indexed addressing mode.

1) Immediate Addressing Mode.

→ In this Mode, Source operand is Constant.

→ The operand comes immediately after opcode.

→ In instructions, #= symbol is used for immediate data.

→ This addressing mode is used to load info into any of registers including DPTR Reg which stands for Data Pointer i.e it points the external data memory loc^{n}.

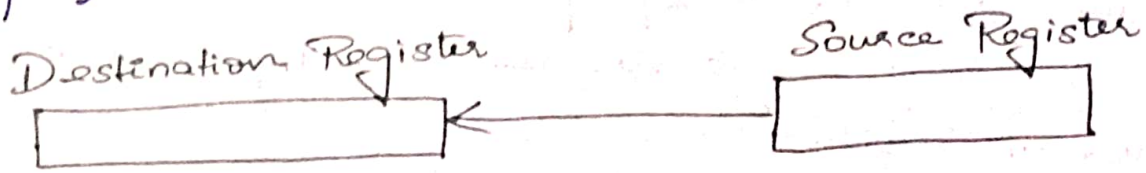
Eg: MOV A, #5511 → Load 5511 into A→ (register)
MOV R4, #62 → Load decimal value 62 into register R4.

2) Register Addressing Mode.

→ In this Mode, the data to be operated is available inside the registers.

→ Register name is used directly as Source operand.

→ Operation is performed within various registers of µP.



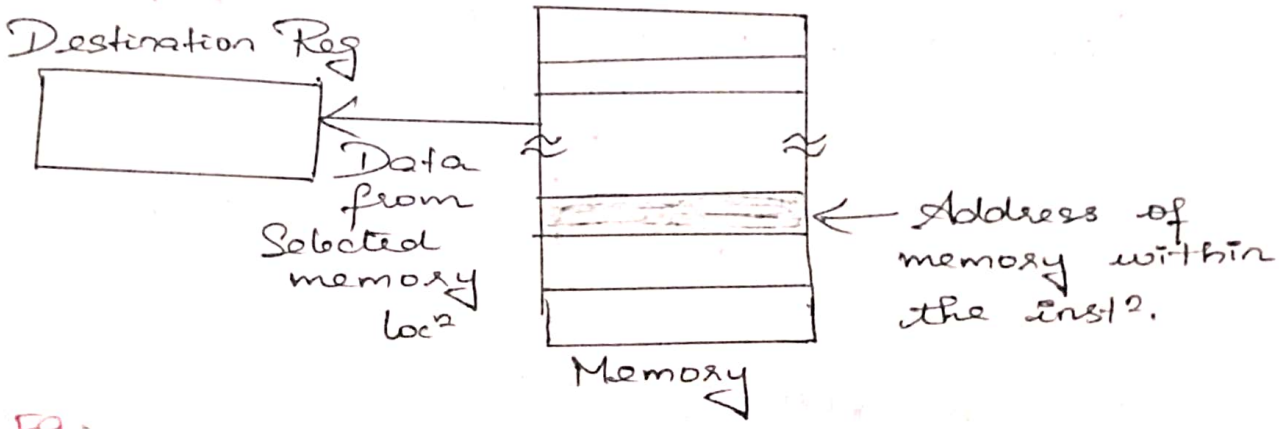
Eg:
MOV A, R0 → Copy the Contents of R0 into A
MOV R2, A → Copy the Contents of A into R2
ADD A, R5 → add the Contents of R5 to Contents of A
ADD A, R7 → add the Contents of R7 to A
MOV R6, A → Save accumulator in R6.

3) Direct Addressing Mode:

→ In this type, address of data loc^n (Source data) is given as an operand.

→ This oper^n is only for internal RAM & special function Registers (SFR) because it provides an address of only 8 bit.

→ RAM has been assigned addresses from 00 to 7FH.



Eg: MOV A, 25H → Data from 25H loc^n given to Reg A
MOV 30H, A → move A reg data to 30H loc^n.

SFR Registers (Special function Register).

→ These are special Registers tied to some special fun^n or status of processor.

→ They might not be directly writable by normal inst^n.

→ SFR are located immediately above 128 bytes of RAM & responsible for oper^n of ALU, timer, Serial port, parallel port & interrupt Control.

→ Some SFR in 8051 are

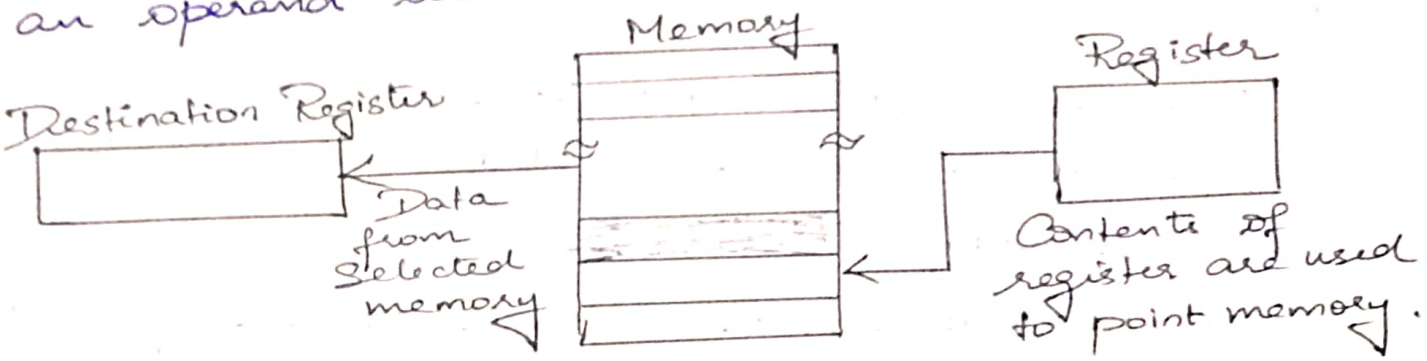
* Accumulator
* B Register
* Program status word
* DPTR
* Stack Pointer

4) Register Indirect Addressing Mode:-

→ In this Mode, a register is used as a pointer to the data.

→ If the data is inside CPU, only registers R0 & R1 are used as pointers & preceded by @ sign.

→ R2-R7 Cannot be used to hold the address of an operand located in RAM.



EX

MOV A, @R0 → Move Contents of RAM loc= whose address is held by R0 into A.

→ Here value inside R0 is Considered as an address which holds the data to be transferred to acaumul?

MOV @R1, B → Move Contents of B into RAM loc= whose address is held by R1.

Advantages:-

→ It access the data dynamic rather than static
→ Efficient use of memory
→ Flexibility in addressing
→ Easy Implementation.

Limitations:-

→ R0 & R1 are only registers that Can be used as pointers in registers.

→ R0 & R1 are 8bit wide & limited to access any info in internal RAM.

5) Indexed Addressing Mode:-

→ It is widely used in accessing data elements of look-up table entries located in ROM space

→ The destination operand is always the register A.

→ Either DPTR or PC can be used as index register.

Ex: MOVC A, @A+DPTR

→ 16 bit register DPTR & register A are added to form the address of data element stored in on-chip ROM.

→ Instruction MOVC is used instead of MOV & C means Code.

MOVC A, @A+PC.

→ Here Data inside PC is added with accumulator to obtain target address.

IV) Instruction Set:-

→ The instructions in 8051 can be classified into 5 groups.

1) Data transfer inst^n
2) Arithmetic inst^n
3) Logical inst^n
4) Program branch inst^n
5) Boolean or Bit Manipulation inst^n.

① Data transfer Instructions

Mnemonic	Description	Operation
MOV A, Rn	A ← Rn	Move Register to A
MOV A, Direct	A ← (addr)	Move Direct byte to A
MOV A, @Ri	A ← (Ri)	Move Indirect RAM to A
MOV A, #data	A ← Data	Move immediate data to accumulator
MOV Rn, A	Rn ← A	Move accumulator to Register
MOV Rn, direct	Rn ← (addr)	Move Direct byte to register.
MOV Rn, #data	Rn ← data	Move immediate data to register.
MOV direct, A	(addr) ← A	Move accumulator to direct byte
MOV direct, Rn	(addr) ← Rn	Move Register to direct byte
MOV direct, @Ri	(addr) ← (Ri)	Move Indirect RAM to direct byte.
MOV, @Ri, A	(Ri) ← A	Move accumulator to indirect RAM
MOV @Ri, direct	(Ri) ← (addr)	Move direct byte into indirect RAM.
MOV DPTR, #data16	DPTR ← data 16	load data pointer with 16 bit Constant.
MOVC A, @A+PC	A ← (A+PC)	Move Code byte relative to PC to accumulator
MOV x A, @Ri	A ← (Ri)^	Move External RAM
MOV x @Ri, A	(Ri)^ ← A	Move accumulator to External RAM.

Mnemonic	Description	Operation
PUSH Direct	(SP) ← ADDR	Push direct byte onto stack
POP direct	(addr) ← (SP)	POP direct byte from stack
XCH A, Rn	A ↔ Rn	Exchange Register with A
XCH A, Direct	A ↔ (addr)	Exchange direct byte with accumulator
XCH A, @Ri	A ↔ (Ri)	Exchange indirect RAM with accumulator

Rn → Any of 8 registers
Ri → Either of pointing registers
addr → Address of internal RAM
↓ → Least Significant Nibble
^ → External Memory loc^n
( ) → Content of loc^n inside parenthesis.

2) Arithmetic Instructions:-

Mnemonic	Description	Operation
ADD A, Rn	A ← A+Rn	Add Register to accumu^r
ADD A, Direct	A ← A+(addr)	Add Direct byte to A
ADD A, @Ri	A ← A+(Ri)	Add indirect RAM to A
ADD A, #data	A ← A+data	Add immediate data to A
ADDC A, Rn	A ← A+Rn+C	Add Register to A with Carry
ADDC A, @Ri	A ← A+(Ri)+C	Add indirect RAM to accumulator with Carry
ADDC A, #data	A ← A+data	Add immediate data to accumulator with Carry.

Mnemonic	Description	Operation
SUBB A, Rn	A ← A-Rn-C	Subtract Register from accumulator with borrow.
SUBB A, Direct	A ← A-(addr)-C	Subtract Direct byte from accumulator with borrow
SUBB A, @Ri	A ← A-(Ri)-C	Subtract indirect RAM from accumulator with borrow.
INC A	A ← A+1	Increment accumulator
INC Rn	Rn ← Rn+1	Increment Register
INC Direct	(addr) ← (addr)+1	Increment Direct byte
INC @Ri	(Ri) ← (Ri)+1	Increment indirect RAM
INC DPTR	DPTR ← DPTR+1	Increment Data Pointer
DEC A	A ← A-1	Decrement Accumulator
DEC Rn	Rn ← Rn-1	Decrement Register
DEC Direct	(addr) ← (addr)-	Decrement Direct byte
DEC @Ri	(Ri) ← (Ri)-1	Decrement indirect RAM
MUL AB	AB ← AXB	Multiply A & B
DIV AB	AB ← A/B	Divide A by B.

Ex:

```
MOV A, #+5
MOV R0, #+3
ADD A, R0
```

```
  0000  0101
  0000  0011
  ----------
A = 0000  1000
```

Ans ⇒ 08

3) Logical Instructions:-

Mnemonic	Description	Operation
ANL A, #data	AND immediate data to accumul:	ANL <dest-byte>, <src-byte>
ANL A, Rn	AND (A) ← (A) ∧ (Rn)	AND registe to accumulator
ANL A, direct	(A) ← (A) ∧ (Direct)	AND direct byte to accumulator.
ANL A, @Ri	(A) ← (A) ∧ (Ri)	AND indirect RAM to accumulator
ANL Direct, #data	(direct) ← (direct) ∧ (#data)	AND immediate data to direct byte.
ANL direct, A	(direct) ← (direct) ∧ (A)	AND accumulator to direct byte.
ORL A, #data	(A) ← (A) ∨ (#data)	OR immediate data to accumulator
ORL A, Rn	(A) ← (A) ∨ (Rn)	OR Registe to accumulator
ORL A, direct	(A) ← (A) ∨ (direct byte)	OR Direct byte to accumulator
ORL A, @Rn	(A) ← (A) ∨ (Ri)	OR Indirect RAM to accumulator
ORL direct, #data	(direct byte) ← (direct byte) ∨ (#data)	OR immediate data to direct byte
ORL direct, A	(direct) ← (direct) ∨ (A)	OR accumulator to direct byte
XRL A, #data	(A) ← (A) ⊕ (#data)	Exclusive OR immediate data to accumulator.

XRL A, Rn	(A) ← (A) ⊕ (Rn)	Exclusive OR Reg to accumulator
XRL A, direct	(A) ← (A) ⊕ (Direct)	Exclusive OR direct byte to accumulator
XRL A, @Rn	(A) ← (A) ⊕ ((Rn))	Exclusive OR indirect RAM to accumulator.
CLR A	A ← 0	Clear Accumulator
CLR bit	CLEAR P1.7 (P1.7 = 0)	Clear Bit
CPL A	A ← $\bar{A}$	Complement Accumulator
RL A	(Dn+1) ← (Dn)	Rotate Accumulator left
RLC A	(Dn+1) ← (Dn) (D0) ← (CY) (CY) ← (D7)	Rotate Accumulator left through Carry
RR A	(Dn) ← (Dn+1)	Rotate Accum: Right
RRC A	(Dn) ← (Dn+1) (D7) ← (CY) (CY) ← D0	Rotate Accumulator Right through Carry
SWAP A	(D3-D0) ↔ (D7-D4)	Swap Nibbles within accum:

4) Boolean or Bit Manipulation Inst^{n}:

Mnemonic	Operation	Description
CLR C	(cy) ← 0	Clear Carry flag
CLR bit	(bit) ← 0	Clear Direct bit
SETB C	(cy) ← 1	Set Carry flag
SETB bit	(bit) ← 1	Set Direct bit
ANL C, bit	(c) ∧ (bit)	AND direct bit to Carry flag
ANL C, /bit	(c) ∧ ($\overline{bit}$)	AND Complement of direct bit to Carry
ORL C, bit	(c) ∨ (bit)	OR direct bit to Carry flag
ORL C, /bit	(c) ∨ ($\overline{bit}$)	OR Complement of direct bit to Carry
MOV C, bit	(c) ← (bit)	Move Direct bit to Carry flag
MOV bit, C	(bit) ← (c)	Move Carry flag to direct bit

5) Branch Instructions:-

→ It is used to change the sequence of inst^{n} execution which controls the flow of program logic.

→ There are 3 types of branching inst².

(i) Jump Instructions
(ii) Call Instructions
(iii) Return Instructions.

(i) CALL Instructions:

→ It is used to call a subroutine.
→ Subroutines are often used to perform tasks that need to be performed frequently.

* 2 types of CALL Inst².
  (i) LCALL (long call)
  (ii) ACALL (Absolute Call)

ACALL addr11 → Absolute Subroutine Call
LCALL addr16 → long Subroutine Call.

(i) ACALL → Absolute CALL.
→ 11 bits are used for target Subroutine add.

(ii) LCALL → 3 byte instruction
↳ 1 is opcode & 2 bytes are 16 bit addrs.
→ LCALL is used to call subroutines located anywhere within 64K byte.
→ When a subroutine is called, PC register is pushed onto the stack, & SP is ↑ by 2.

(SP) ← (SP) + 2.

→ After finishing the execution of subroutine, RET is executed & PC is popped off the stack.

(ii) Return Instruction:

a) RET → Return from Subroutine.

→ This inst^n is used to return from a Subroutine which is previously entered by Inst^n LCALL (or) ACALL.

→ SP is ↓ by 2.

(SP) ← (SP) − 2.

b) RETI → Return from Interrupt.

→ This is used at end of interrupt Service routine.

→ Pc & SP is ↓ by 2.

(PC) ← (PC) − 2

(SP) ← (SP) − 2.

(iii) JUMP Instructions:-

→ Transfers the program Sequence to the memory address given in the operand based on specified flag.

→ 2 types of JUMP inst^n are

(i) Unconditional Jump Inst^n → Transfers the program Sequence to the discribed memory address.

(ii) Conditional Jump Inst^n → Transfers the program sequence to the discribed memory address only if Cond^n is satisfied.

AJMP addr11 → Absolute Jump.

→ Transfers Pgm execut^n to the target address unconditionally.

Mnemonic	Operand	Description
LJMP	addr 16	long Jump
S JMP	rel	short Jump
JMP	@A+DPTR	Jump indirect relative to DPTR.
JZ	rel	Jump if accumulator is Zero.
JNZ	rel	Jump if accumu^2 is $\neq 0$
JC	rel	Jump if Carryflag is Set
JNC	rel	Jump if Carry flag is not Set
JB	bit, rel	Jump if Direct bit is Set.
CJNE	A, direct, rel	Compare direct byte to A & Jump if not equal.
CJNE	Rn, #data, rel	Compare immediate to register & jump if not equal.
DJNI	Rn, rel	to register & Jump if not Zero
DJNZ	direct, rel	to direct byte & Jump if not zero

V) Program & Data Memory:

i) Program Memory (ROM):

→ 8 bit μc by Intel has 4KB of internal ROM.

→ 8052 Series - 8KB of internal ROM in form of flash memory & provides an option of reprogramming the memory.

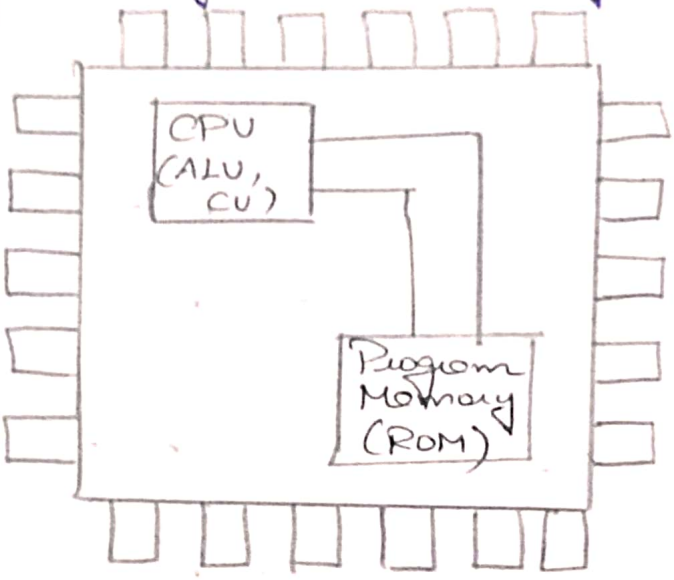
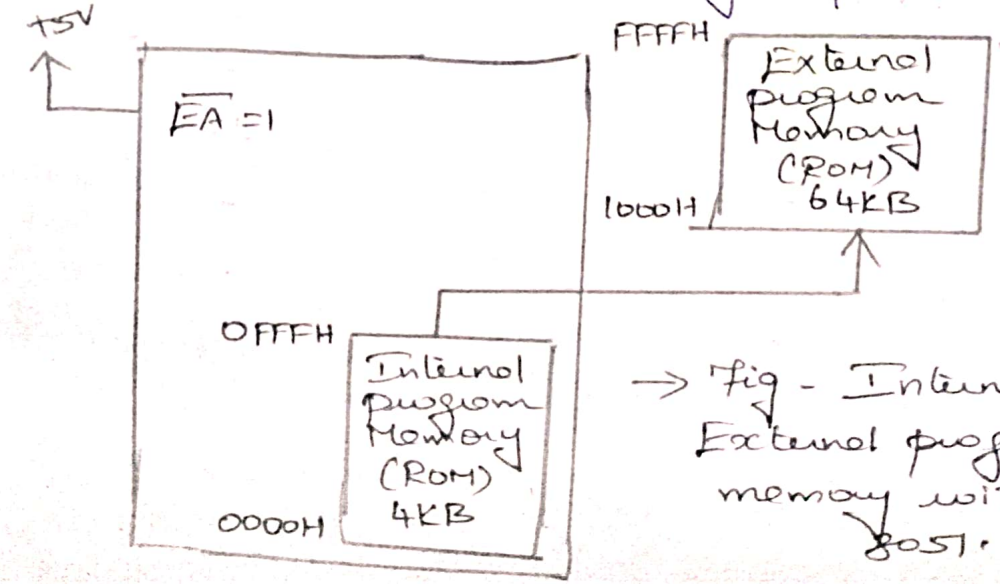


fig - 8051 program memory in μc.

→ When External Access ($\overline{EA} = 1$) is high, CPU first fetches inst^{n} from 4KB of internal (on-chip) ROM in address range of 0000H to 0FFFH.



→ fig - Internal & External program memory with 8051.

→ When all instⁿ are fetch only from an external pgm memory, then EA pin must be connected to GND ($\overline{EA}=0$) & memory address will be from 0000H to FFFFH.

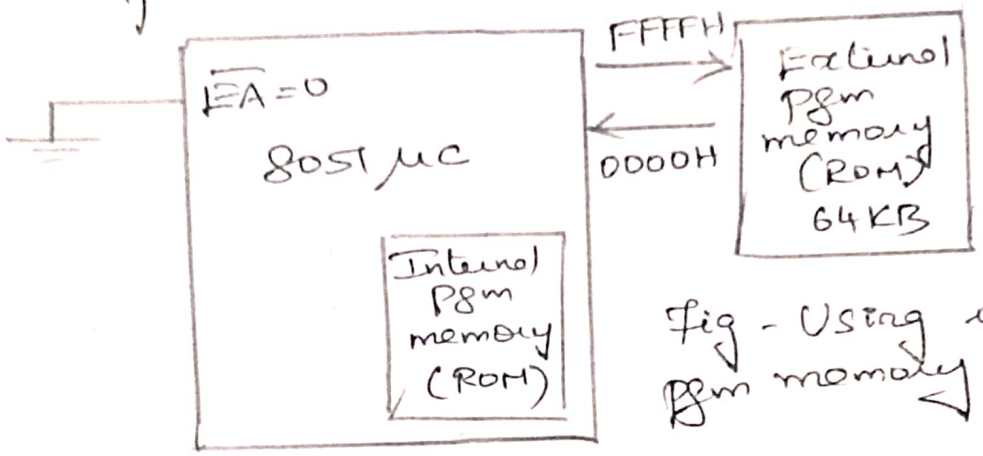


Fig - Using only External Pgm memory with 8051.

(ii) Data Memory (RAM) :-

→ Stores temporary data & intermediate results are used during normal operⁿ of μC.

→ Original Intel 8051 μC → had 128B of internal RAM.

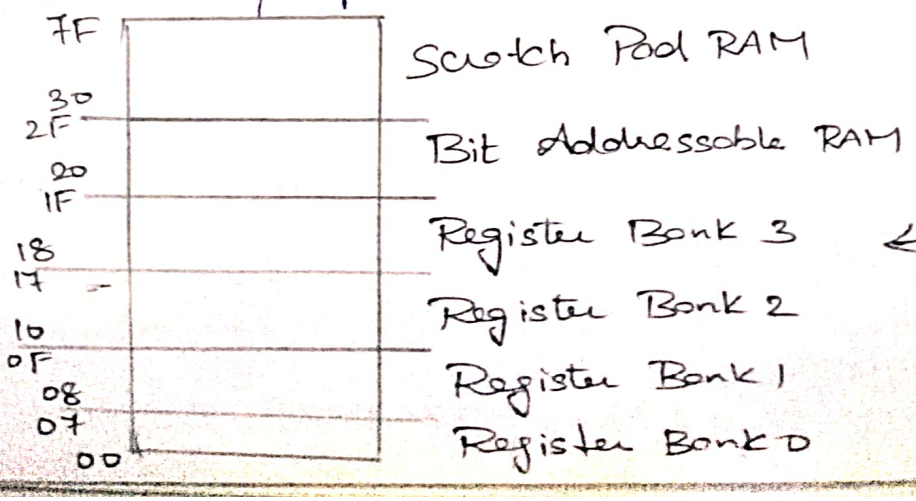
Internal Data Memory :-

↳ 128 bytes of RAM & assigned address 00 to 7FH.

(i) Register Banks

(ii) Bit - addressable RAM

(iii) General purpose RAM



8051 Register Banks

→ In 128B of RAM (from 00H to 7FH), first 32B (i.e memory from address 00H to 1FH consists of 32 working registers. that are organized as 4 banks with 8 registers in each Bank.

Bank0		Bank1		Bank2		Bank 3	
7	R7	F	R7	17	R7	1F	R7
6	R6	E	R6	16	R6	1E	R6
5	R5	D	R5	15	R5	1D	R5
4	R4	C	R4	14	R4	1C	R4
3	R3	B	R3	13	R3	1B	R3
2	R2	A	R2	12	R2	1A	R2
1	R1	9	R1	11	R1	19	R1
0	R0	8	R0	10	R0	18	R0

fig - 8051 Register Banks.

→ 8051 uc is powered up with Register Bank 0.

→ Based on possible Combⁿ of bits RS1 & RS0 of PSW, register bank is changed accordingly.

RS1 (PSW:4)	RS0 (PSW:3)	Register Bank	Address
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

fig - PSW bits bank Setⁿ.

VI) Stacks:-

→ It is a section of internal RAM used by CPU to store the info temporarily.

→ This info could be data or address.

Stack Pointer (SP):-

→ The register used to access the stack is called stack pointer (SP).

→ SP is a small register used to point the stack. When push oper is performed then SP gets increased.

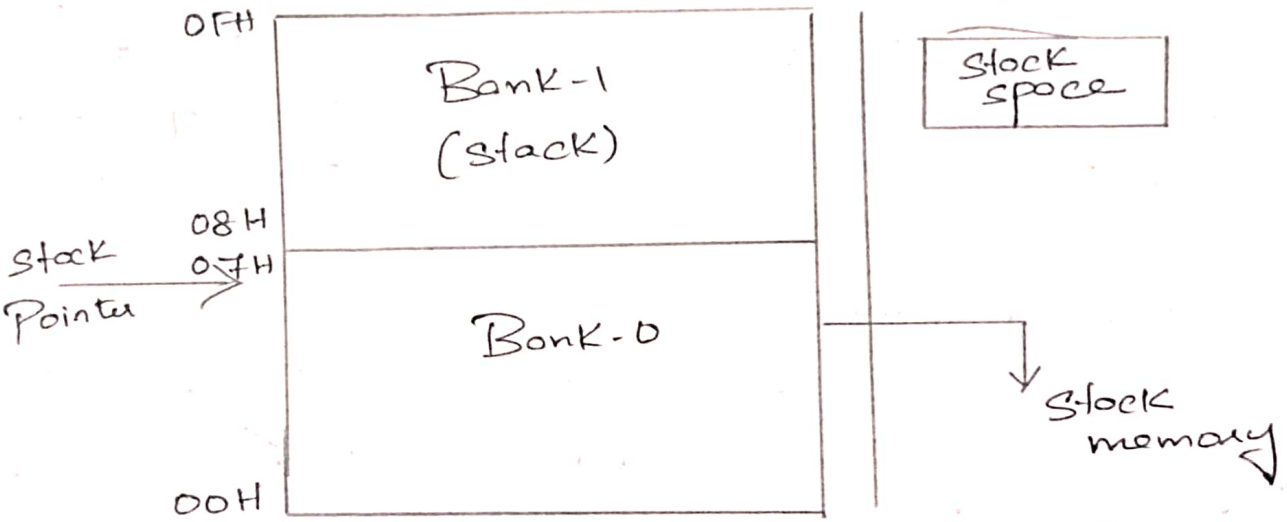


Fig - Stack Memory Alloc in 8051.

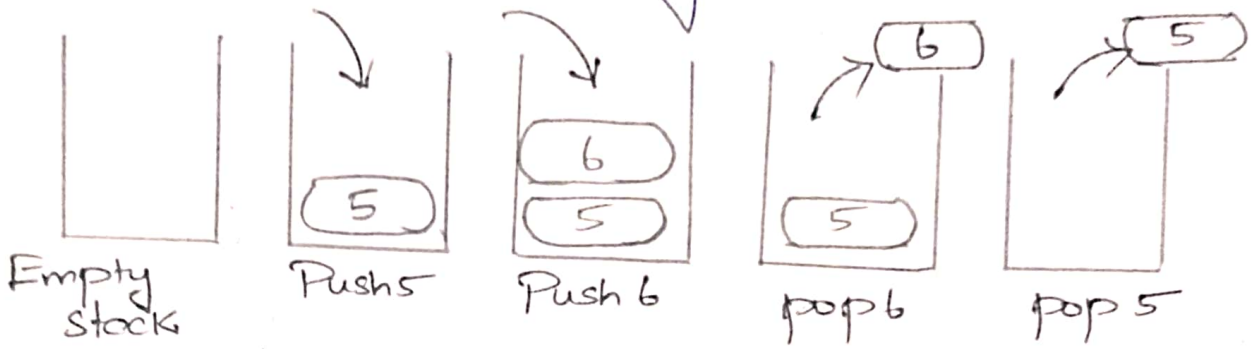
→ SP in 8051 is 8 bit wide & can take a value of 00H to FFH.

→ When 8051 is initialized, SP register contains the value 07H.

→ This means RAM loc 08H is the first loc used for stack.

PUSH & POP operⁿ :-

→ Storing operⁿ of CPU register in stack is known as PUSH & getting Contents from stack back into CPU register Called POP.



1) PUSH operation :-

→ SP points to last used locⁿ of stack.

→ When PUSH is executed, Contents of registe are saved on the stack & SP is ↑ by 1.

→ To push the registers onto stack, we use RAM address.

2) POP operation.

→ Popping the Contents of stack back into given register is opposite to process of pushing.

→ With every pop operⁿ, top byte of stack is Copied to register specified by instⁿ & SP is ↓ by once.

Other Instructions.

→ Other Instⁿ of 8051 that affect the stack & SP are ACALL, LCALL, RET & RETI.

→ CPU also use the stack to save the address of an instⁿ just below CALL instⁿ.

VII) 8051 Interrupts.

→ An interrupt is an external or internal event that interrupts the μc to inform that a device needs its service.

Interrupts:

→ Whenever any device needs its service, then device notifies the μc by sending it an interrupt signal

→ After receiving interrupt signal, μc interrupts whatever it is doing.

→ The pgm associated with interrupt is called interrupt service Routine (ISR).

→ ISR is a sub-routine calls which is given to μc.

→ Interrupts provide a method to postpone or delay the current process, thereby performs a sub-routine task & then restart the std pgm again.

Polling:-

→ Polling can monitor the status of several devices & serve each of them as certain conds are met.

ISR:-

→ Group of memory locs which set separately to hold the address of ISR is called Interrupt Vector Table.

Interrupt	ROM loc^n / Interrupt Vector address	Pin	Flag Clearing
Reset	0000H	9	Auto
External h/w interrupt 0	0003H	P3.2(12)	Auto
Timer 0 interrupt	000BH		Auto
External h/w interrupt 1	0013H	P3.2(13)	Auto
Timer 1 interrupt	001BH		Auto

Steps in Executing Interrupt:-

→ It finishes the inst^n which is executing & saves the address of next (PC) on stack

→ Saves the Current status of all interrupts internally.

→ It jumps to fixed loc^n in memory which is called interrupt vector table.

→ Mc gets address of ISR from Interrupt vector table.

→ Interrupt Service Subroutine is executed until it reaches the last inst^n of Subroutine which is RETI.

→ After executing RETI inst^n, then Mc returns to place where it gets interrupted.

Types of Interrupts:

(i) Reset
(ii) Internal Interrupt
(iii) External H/w Interrupt
(iv) Serial Interrupt

Enabling & Disabling an Interrupt:

D7	6	5	4	3	2	1	D0
EA	–	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	It disables all interrupts. EA=0 → No interrupt is acknowledged.
–	IE.6	Reserved for future use
ET2	IE.5	Enables or disables timer 2 overflow interrupt
ES	IE.4	Enables (ES=1) or Disables (ES=0) the Serial port interrupt.
ET1	IE.3	Enables (ETI=1) or disables (ET1=0) timer 1 overflow interrupt.
EX1	IE.2	Enables (EX1=1) or disables (EX1=0) external interrupt 1.
ET0	IE.1	Enables (ET0=1) or disables (ET0=0) Timer 0 overflow interrupt.
EX0	IE.0	Enables (EX0=1) or disables (EX0=0) external interrupt 0

VIII) Timers :-

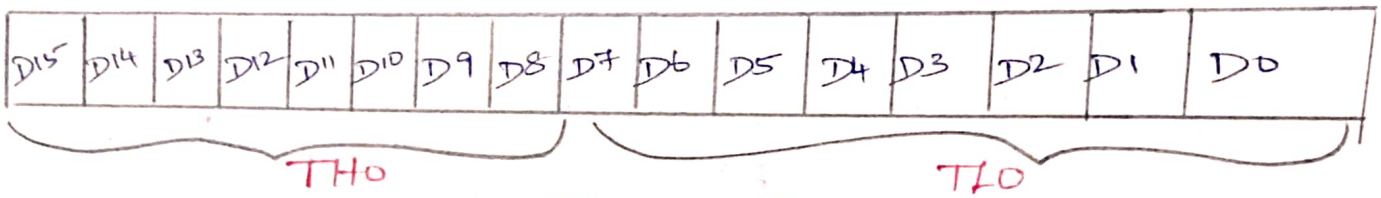
→ Timers are used to generate time delay or as event Counters to Count events happening outside the µc.

→ Both Timer 0 & Timer 1 are 16 bits wide.

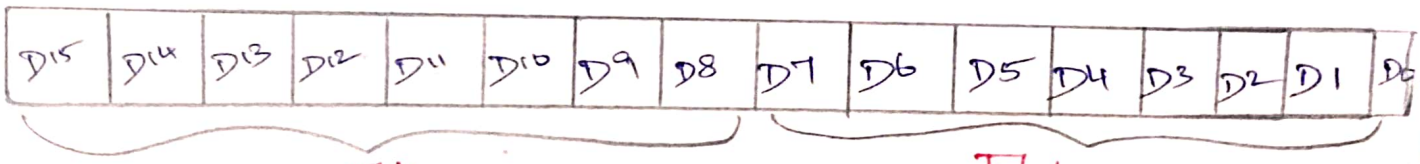
→ Each 16 bit timer is accessed as 2 separate registers of low byte & high byte.

→ low byte register → TL0 / TL1

→ High byte register → TH0 / TH1.



a) Timer 0 Registers.



b) Timer 1 Registers.

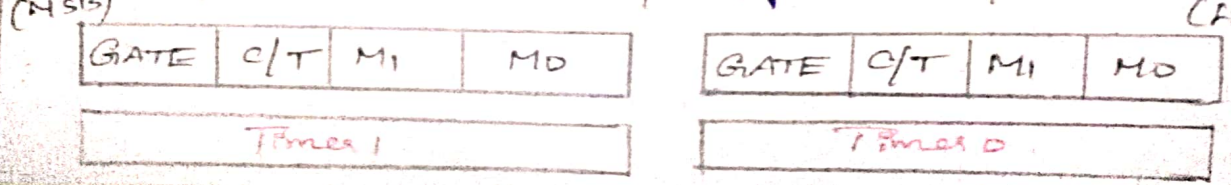
Structure of TMOD Register.

→ Both timer 0 & 1 uses the same register Called TMOD - used to set various timer oper^{n} modes.

→ TMOD - 8 bit register in which lower 4 bits are for Timer 0 & upper four bits are Timer 1.

→ Lower 2 bits (M1 & M0) → Set timer Mode

→ Upper 2 bits → Specify the oper^{n}.



M1	M0	Mode	Operating Mode
0	0	0	13-bit timer mode
0	1	1	16 bit timer mode
1	0	2	8 bit auto reload
1	1	3	split timer Mode.

(i) Mode Bit-1 (M1) & Mode bit-0 (M0).

Mode 0 - 13 bit timer

Mode 1 - 16 bit timer

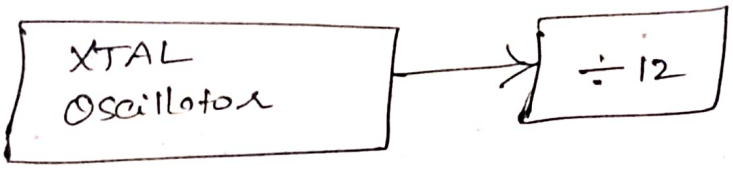
Mode 2 - 8 bit timer.

(ii) C/T (clock/Timer)

C/T = 0 → Used as timer

C/T = 1 → Used as Counter

(iii) Clock Source for Timer.



→ frequency of Timer is always 1/12th the frequency of crystal attached to 8051.

(iv) GATE

a) GATE = 0 → Software is used.

→ SETB inst^n starts & stopped by CLR inst^n.

b) GATE = 1 → h/w is used.

TCON (Timer/Counter) Register:-

→ Controls timer/counter oper^{n}.

→ lower 4 bits → Interrupt fun^{n}

Upper 4 bits → Timer oper^{n}.

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

BIT	SYMBOL	FUNCTION
TCON.7	TF1	Timer 1 overflow flag
TCON.6	TR1	Timer 1 run Control flag
TCON.5	TF0	Timer 0 overflow flag
TCON.4	TR0	Timer 0 run Control flag

Fig - TCON Register.

Bit 3: IE1

→ An external interrupt 1 edge flag is Set by h/w when interrupt on INT1 pin occured & cleared by h/w when interrupt get processed.

Bit 3: IT1

→ This bit Selects external interrupt event type on INT1 pin

1 = Sets interrupt on falling edge

0 = Sets interrupt on low level.

Bit 1: IE0

→ Interrupt 0 edge flag, set by h/w when interrupt on INT0 pin occured & cleared by h/w when interrupt is processed.

Bit 0: IT0

→ 1 = Sets interrupt on falling edge

0 = " " on low level.

Counters :-

→ Timers can also be used as Counters Counting events happening outside 8051.

C/T bit in TMOD Register :-

→ The C/T bit in TMOD registers decides the source of the clock for the timer.

If C/T = 0 → Timer gets pulses from crystal

C/T = 1 → Timer is used as a Counter & gets its pulse from outside 8051.

Pin	Port Pin	Functⁿ	Description
14	P3.4	T0	Timer/Counter 0 external i/p
15	P3.5	T1	Timer/Counter 1 external i/p

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

Fig - Port 3 pins used for Timers 0 & 1.

→ Port 3 pins are used for Timers 0 & 1.

→ In case of Timer 0, C/T = 1, pin P3.4 provides the clock pulse & Counter Counts up for each clock pulse.

→ For Timer 1, when C/T = 1 → Each clock pulse coming in from pin P3.5 that makes the Counter Count up.

1) Counter 0 in Mode 1:-

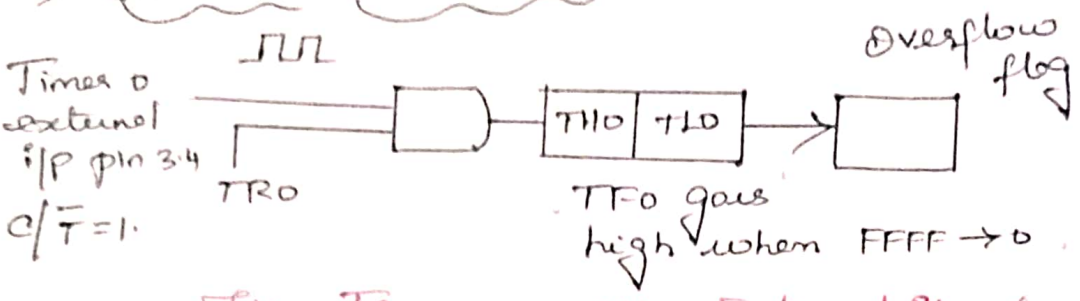


Fig - Timer 0 with External i/p (mode 1)

→ To operate Counter 0 in mode 1,

(i) C/T̄ = 1 → allow Counter mode operⁿ

(ii) M1:M0 bits are set to 01 to select mode 1.

(iii) When GATE = 0 & TR0 is set to 1 to start Counter

(iv) When GATE = 1, Counter will run only if TR0 is set to 1 & logic signal on external interrupt pin INT0 is high.

2) Counter 1 in Mode 1:-

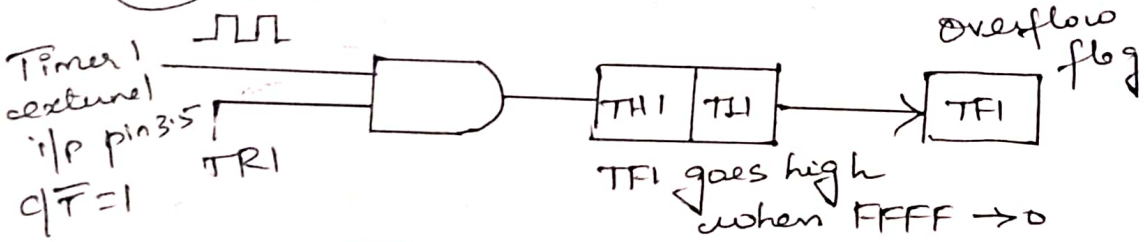


Fig - Timer 1 with External i/p (mode 1)

To operate Counter 1 in mode 1,

(i) C/T̄ = 1 → allow Counter mode operⁿ

(ii) M1:M0 bits are set to 10 to select mode 1

(iii) When GATE = 0 & TR1 is set to 1 to start Counter

(iv) " GATE = 1, Counter will run only if TR1 is set 1 & logic signal on external interrupt pin INT1 is set high.

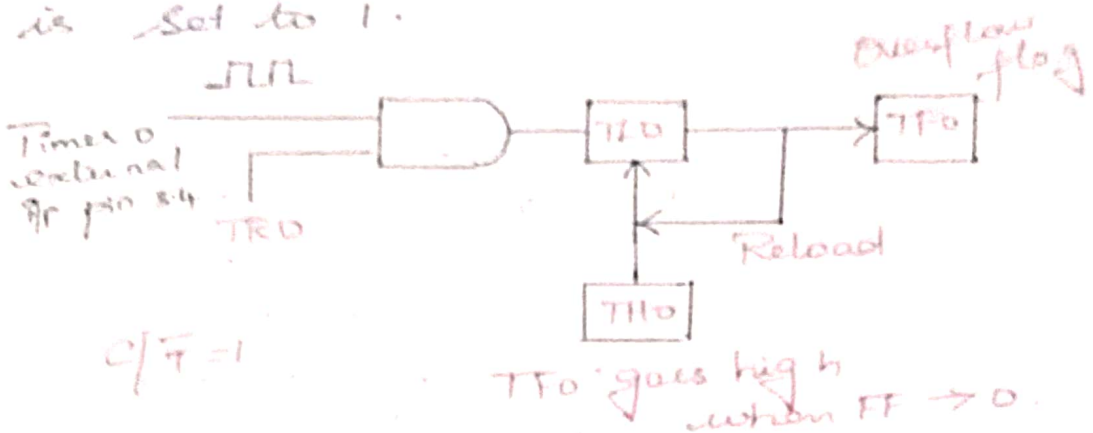
3) Counter 0 in Mode 2:

(i) C/T̄ = 1 → allow Counter mode operⁿ

(ii) M1:M0 bits are set to 10 to select mode 2.

→ When GATE = 0 & TR0 is Set to 1 → to start the Counter.

→ When GATE = 1 → Counter will run only if TR0 is Set to 1.



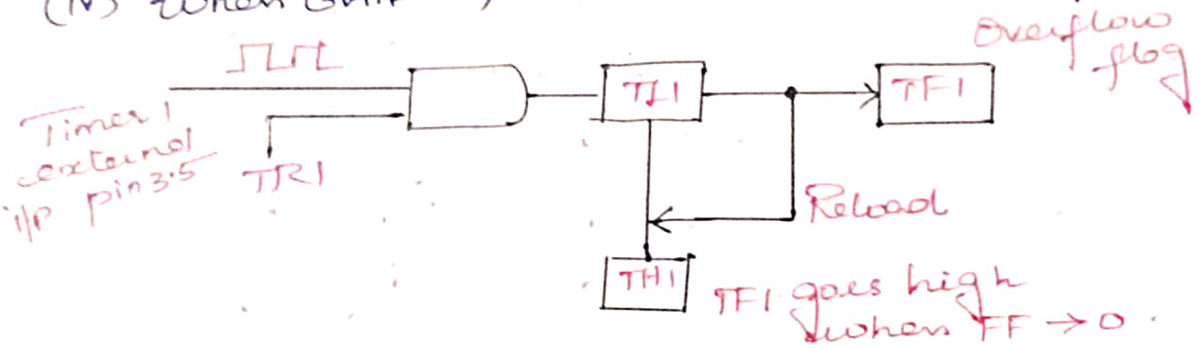
(iv) Counter 1 in Mode 2.

(i) C/T = 1 → allow Counter mode operation

(ii) M1:M0 bits are Set to 10 to Select mode 2

(iii) When GATE = 0 & TR1 is Set to 1 – to start Counter

(iv) When GATE = 1, Counter will run only if TR1 is Set 1.



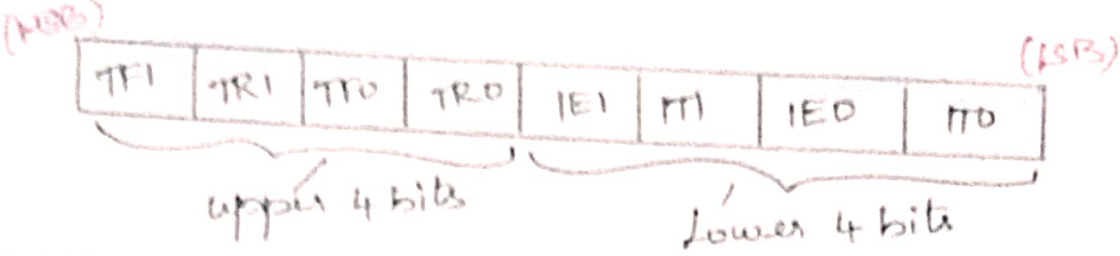
2) TCON Register : Timer/Counter Register :-

→ TCON is a 8 bit register

→ Upper four bits are used to store the TF & TR bits of both timer 0 & timer 1.

→ Lower 4 bits → used for Controlling the interrupt bits

→ TR0 & TR1 flags in TCON register are used to turn ON or off the timers.



Symbol	Position	Name & Significance
TF1	TCON.7	Timer 1 overflow flag. Set by h/w on timer/counter overflow.
TR1	TCON.6	Timer 1 Run Control bit. * Set/cleared by s/w to turn timer/counter -on/off.
TF0	TCON.5	Timer 0 overflow flag. * Set by hardware.
TR0	TCON.4	Timer 0 run Control bit * Set/cleared by s/w
IE1	TCON.3	Interrupt Edge flag. * Set by h/w when external interrupt edge detected.
IT1	TCON.2	Interrupt 1 Control bit * Set/cleared by s/w.
IE0	TCON.1	Interrupt 0 Edge flag. * Set by h/w when External interrupt edge detected.
IT0	TCON.0	Interrupt 0 type Control bit.

Fig - TCON Register

8051 Serial Port:

→ RS 232 is a std Comm^n port for Connecting the Computers & their peripheral devices to enable Serial data exchange.

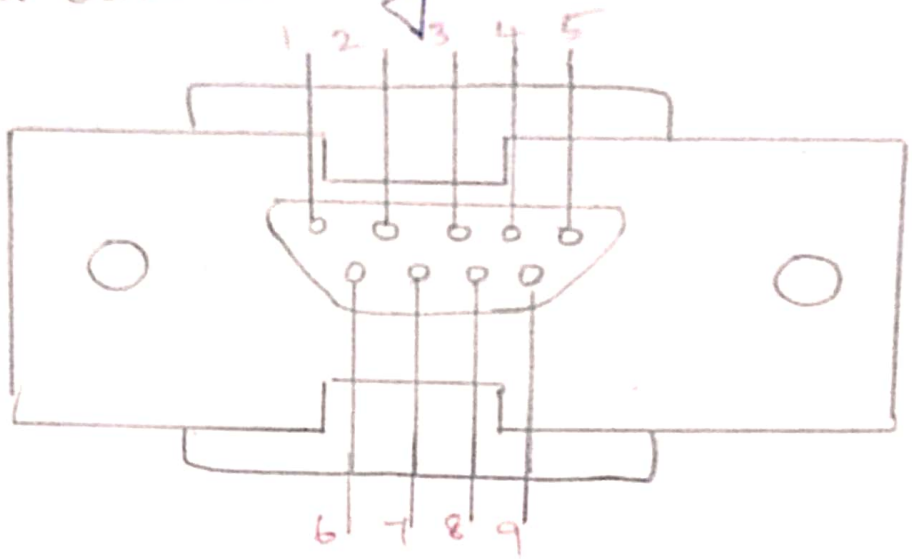


Fig - RS 232 Connector DB-9.

Pin	Description
1	Data Carrier Detect (DCD)
2	Received Data (RxD)
3	Transmitted data (TXD)
4	Data terminal Ready (DTR)
5	Signal ground (GND)
6	Data Set Ready (DSR)
7	Request to Send (RTS)
8	Clear to Send (CTS)
9	Ring Indicator (RI)

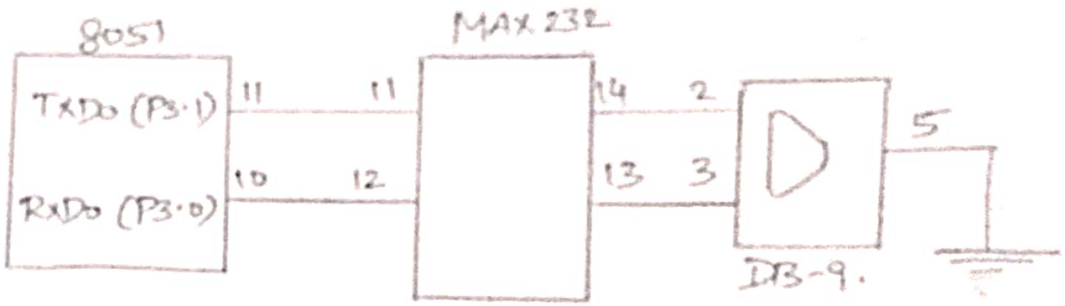
① RxD & TxD Pins.

→ 8051 has 2 pins that are used for transferring & receiving data Serially.

→ Pin 11 → assigned to TxD
Pin 10 → " to RxD.

MAX 232

→ RS 232 is not TTL Compatib. Hence MAX 232 Chip is used to Convert RS 232 voltage levels to TTL levels & vice versa.



Baud Rate in 8051:

→ Allow data transfer b/w PC & 8051 s/m without any error.

→ Baud Rate of 8051 s/m matches baud rate of PC.

→ 8051 transfers & Receives data serially

→ 8051 divides the Crystal freq by 12 in order to get machine cycle freq.

→ UART Circuit divides machine cycle freq of 921.6KHz by 32 before used by timer 1.

→ ∴ 921.6 ÷ 32 gives 28,800 Hz.

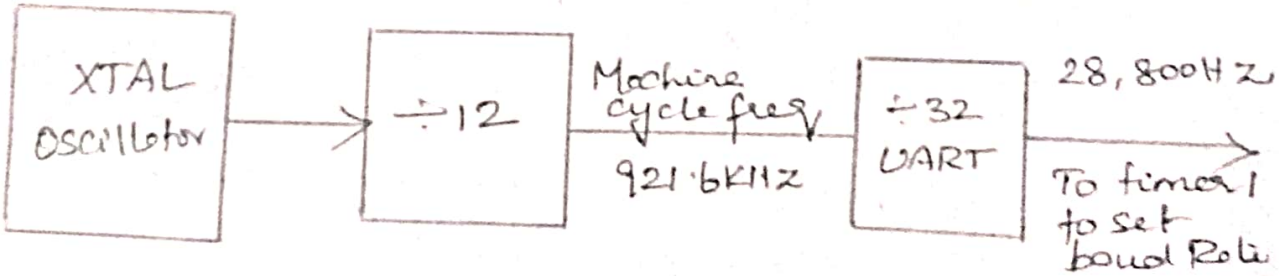


fig - Baud Rate in 8051.

SBUF Register:

→ It is 8 bit register used for Serial Comm².

→ SBUF holds byte of data when it is received by 8051 RxD line.

SCON Register:

→ It is 8 bit register used to program the start bit, stop bit & data bits of data framing.

D7	6	5	4	3	2	1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol	Position	Significance
SM0	SCON.7	Serial port mode Control bit 0 which is set/cleared by s/w
SM1	SCON.6	Serial port mode Control bit 1 which is set/cleared by s/w
SM2	SCON.5	Serial port mode Control bit 2 used for µP Comm²
REN	SCON.4	Receiver Enable Control bit. Set/cleared by s/w to enable/ disable serial data reception
TB8	SCON.3	Transmit Bit 8. Not widely used.
RB8	SCON.2	Receive Bit 8. Not widely used
TI	SCON.1	Transmit Interrupt flag. Set by h/w & cleared by s/w
RI	SCON.0	Receive Interrupt flag. Set by h/w & cleared by s/w

Fig- SCON Serial port Control Register.

Unit 2
Embedded Systems

Embedded system Design Process - Model train Controller - ARM Processor - Instruction set Preliminaries - CPU - Programming Input & o/p - Supervisor Mode - Exceptions & traps - Models for programs - Assembly, Linking & loading - Compilation Techniques - Program level Performance analysis.

1) Embedded Sys Design Process:

→ Major steps involved in Embd Sys are
1) Top down Design - Begin with abstract descriptⁿ & Concludes with Concrete details.
2). Bottom-up Design - start from small components to build a large sys.

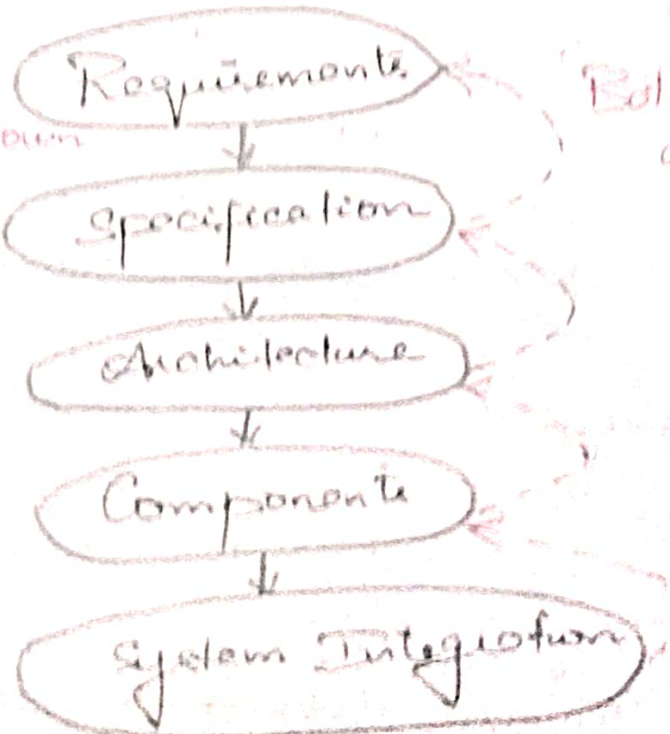


Fig- Major levels of Abstraction in design process

Top Down View:
→ Start with s/m requirements & specification
→ Bottom-up design info help to refine the s/m & given by dashed-line arrows.

(i) Requirements:
→ Gather an informal discription from the Customers known as requirements &
→ Refine the requirements into specific? to begin the designing of s/m architecture.

(i) Requirements Vs Specification:
→ Requirement is a plain language discription of what the user wants & expects to get.
→ Requirements may be
* Functional or
* Non-functional.

Non-functional Requirements:
→ It includes
(i) Performance (Comb? of so,1 performance metrics such as approximate time to perform user level fu° & hard deadlines)
(ii) Cost → (Target cost or purchase price of s/m is a consideration).
* Manufacturing Cost → Includes cost of comp & assembly
* Non Recuring Eng → Include personnel & other costs.
(iii) Physical Size & Weight
(iv) Power Consumption. (Battery powered).

(ii) Validating Requirements :-

→ It requires understanding both (what people want & how they communicate those needs)

→ Give the Customer a good idea that how s/w will be used & how user can react to it.

Simple Requirements form:

→ Capturing a relatively small amount of info in a clear & simple format is a good start.

→ Requirement form is used as a Checklist.

Name	GPS moving map
→ Purpose → Inputs → Outputs → Functions → Performance → Cost → Power → Physical size & weight.	

Specification :-

→ Serves as a Contract b/w Customer & architects.

→ Specifⁿ is essential in Creating the working s/w with minimum designer effort.

→ Specifⁿ should be understandable.

Eg. → Specifⁿ of GPS s/w would include Several Components

→ Data Rcd from GPS Satellite
→ Map data
→ User Interface
→ Operⁿ performed.

Architecture Design:-

→ Architecture is a plan for overall structure of s/m that will be used to design the Components.

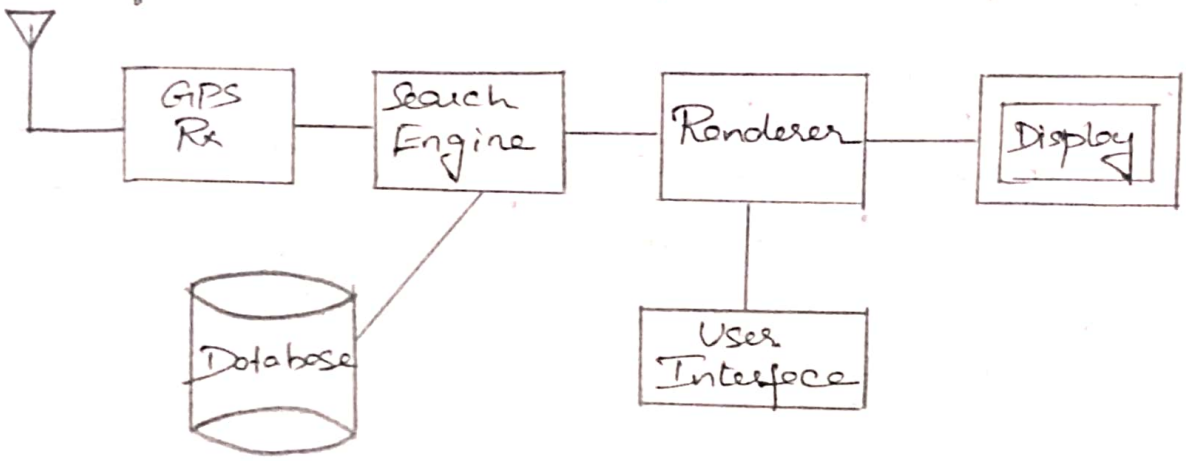


fig - Block Diag for Moving Map.

→ The above diag gives an idea that helps in implementing the fu².

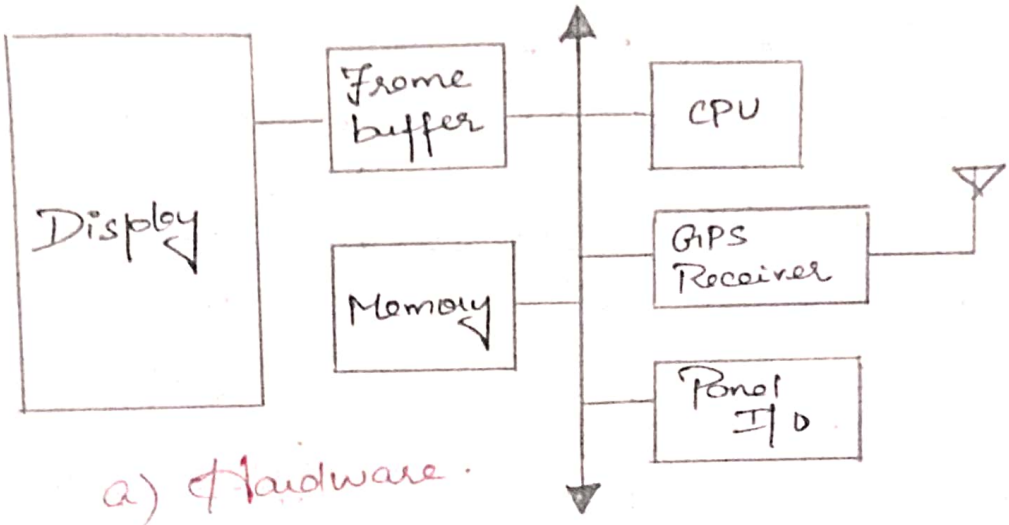
→ Refine the s/m block Diag into 2
  (i) One for h/w &
  (ii) one for s/w

→ h/w shows one Central CPU Surrounded by memory & I/o devices.

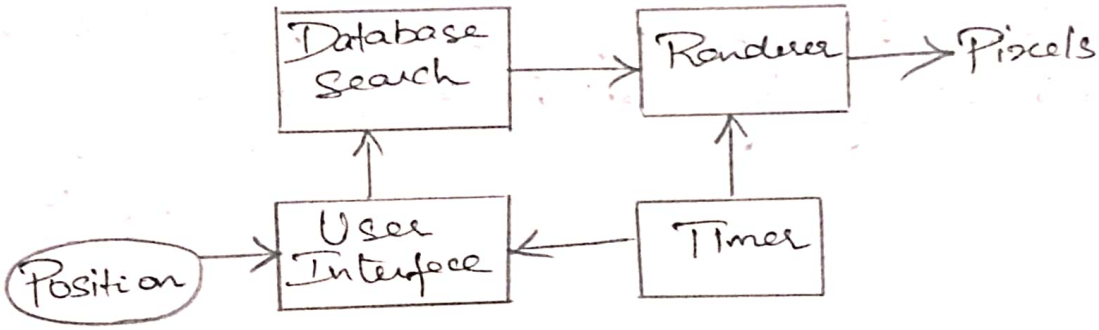
→ 2 memories are chosen
  frame buffer - for pixels
  Separate pgm/data memory - for use of CPU.

→ Architectural Descriptions must be designed to satisfy both functional & non functional requirements.

→ s/w architecture is refined into h/w & s/w architectures.



a) Hardware.



b) Software.

System Integration:-

→ After all components are built, they are integrated together to check whether the s/m is working or not.

→ Bugs are found during this s/m integration.

Formalism of s/m Design:-

→ UML (Unified Modeling Language) is an object oriented modeling language which is designed to be useful at many levels of abstraction.

* Object Oriented Design — Design described as a no. of interacting objects rather than large monolithic block of code.

→ Use UML to model the outside world.

Object - Oriented Specifications:-

→ Allows s/m to closely model the real world objects.

→ Provides a basic set of primitives that can be used to describe s/m with particular attributes.

II) Model Train Controller:-

→ The user sends messages to the train with a control box attached to the tracks.

→ Control box have familiar controls such as throttle, emergency stop button & so on.

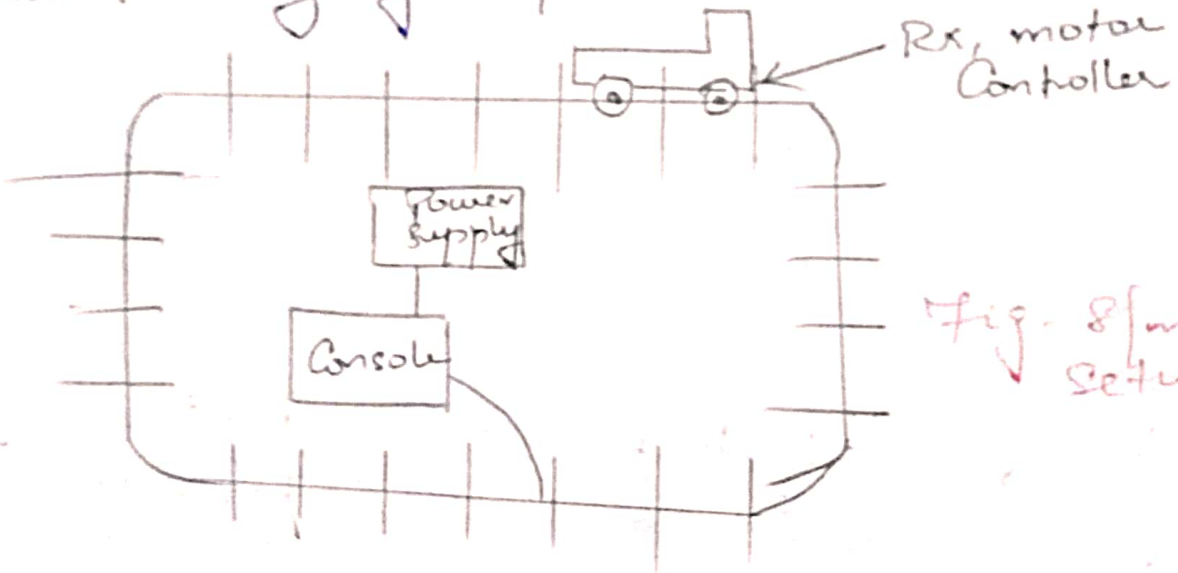


Fig. s/m setup.

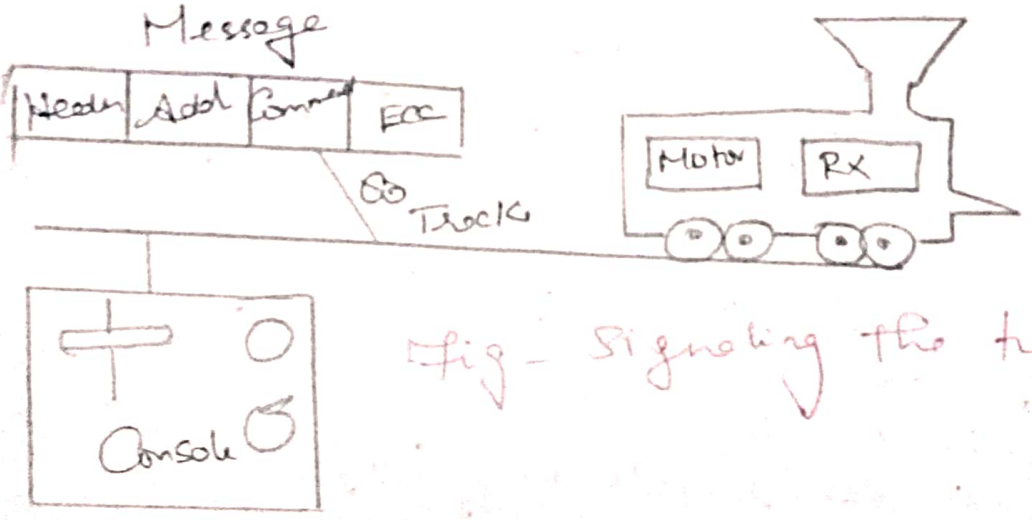


Fig - Signaling the train

→ The train Rx its electrical power from 2 rails of track the Command Control box can send signals to the train over the tracks by modulating power supply Vtg.

→ The Control panel sends packet over the tracks to the Rx on train.

→ The train includes analog electronics to sense the bits being Txd & a Control Sfm to set train motor speed & dirn based on those Commands

→ Each packet includes an address so that a Console can Control several trains on same tracks & packet includes Error Correction Code (ECC) to guard against Txn errors.

→ This is one way Commn Sfm. So model train Cannot send Commands back to user.

Requirements

Name	Model Train Controller
Purpose	Control speed upto 8 model trains
Inputs	Throttle, emergency stop, train no
Outputs	Train Control Signals
Functions	Set engine speed based on inertia settings.
Performance	Can update train speed at least 10 times per sec.
Manufacturing Cost	$50
Power	10W

→ The Console shall be able to Control upto 8 eights on Single track.
→ The speed of each train shall be Controllable by throttle to at least 63 diff levels.
→ There shall be Inertia Control – allow user to adjust the responsiveness of train
→ There shall be emergency stop button.
→ Error detection scheme - used to transmit message.

Conceptual Specification:-

→ The Conceptual specific^n helps us to write a detailed specif^n that needs to be given to s/m.
→ A train Control s/m turns Commands into packets.

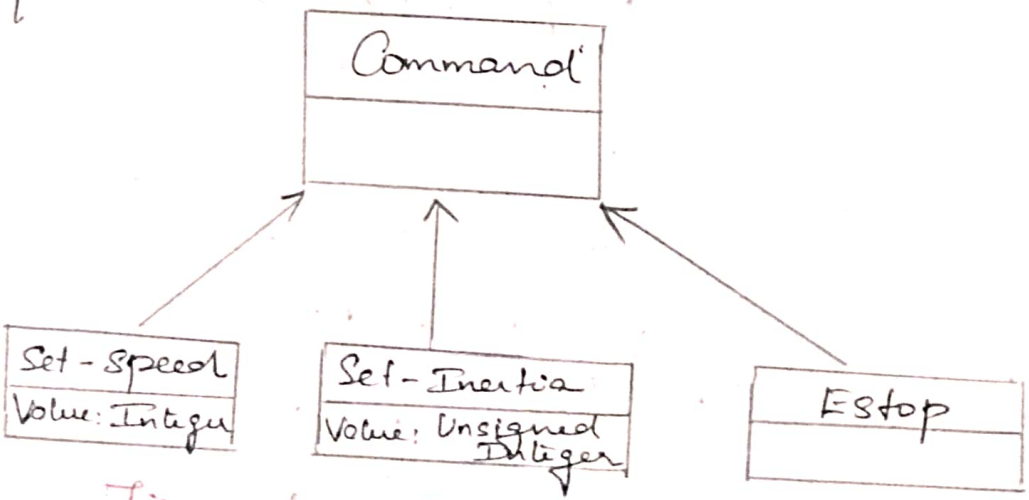


Fig - Class Diag for train Controller Commands

① Subsystems:

→ There are 2 major Sub s/m
(i) Command unit
(ii) Train-board Component (Rx)

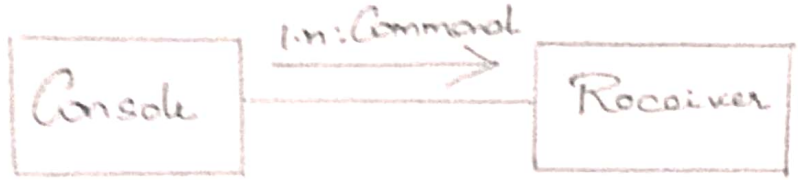


Fig - UML Collaboration diag for major sub sfm.

→ Console sends all message which are numbered the assoo message 1..n.

→ The Console needs to peform the following 3 fun^s.

→ Read the state of front panel of Command unit

→ Front message &

→ Transmit message.

→ Train Rx must also peform 3 major fun^s.

→ Receive the message

→ Interpret the message

→ Control the motor.

UML Class Diagram:-

→ UML class diagram shows the Console class using 3 classes.

(i) Panel class → Describes the Command unit front panel which contains analog knobs & h/w to interface.

(ii) Formatter class → includes behaviors that know how to read the panel knobs & creates bit stream.

(iii) Transmitter class → Interface to analog electronics to send the message along the track.

→ Train makes use of 3 other classes that define its components.

* Receiver
* Controller
* Motor Interface

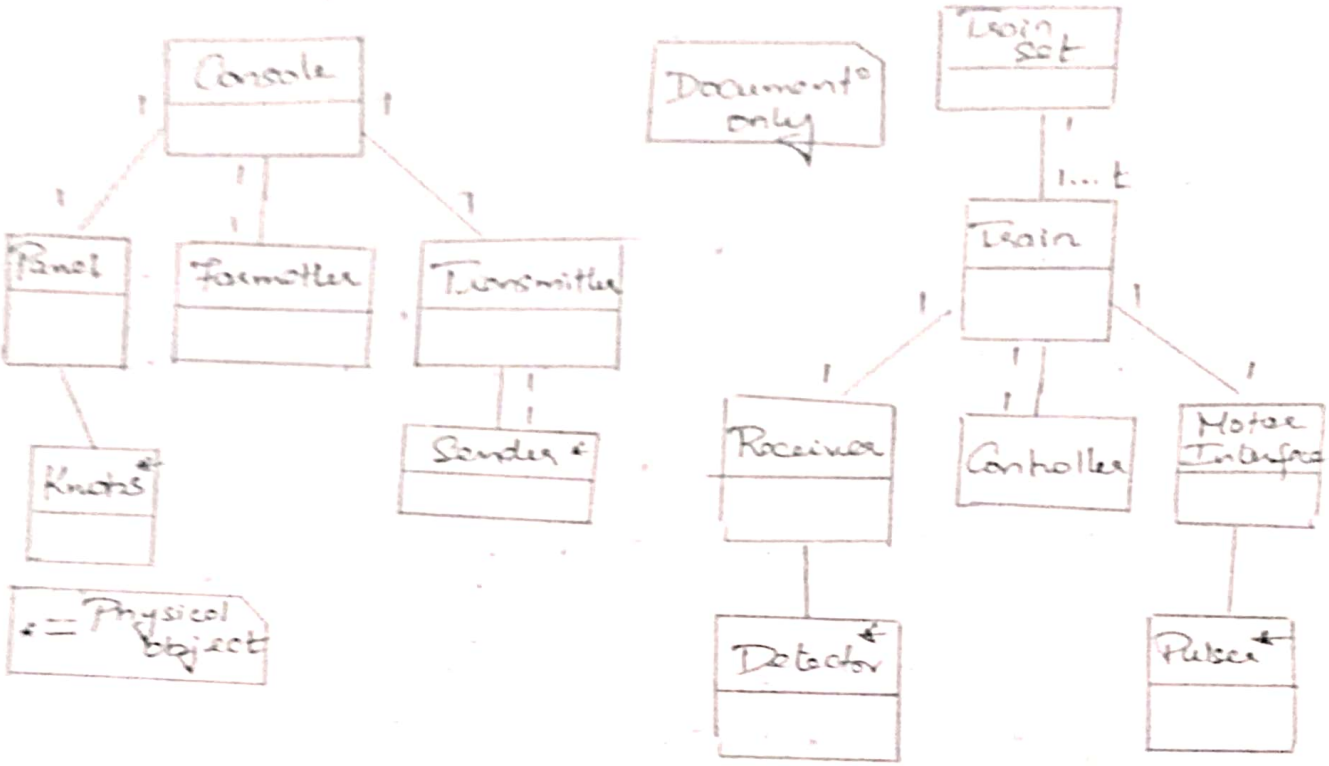


Fig - UML class Diag for train Controller.

→ Two classes to represent analog comp.

(i) Detector → detects analog signals on track & converts into digital form.

(ii) Pulser → Turns digital commands into an analog signals.

Detailed Specification.

→ The Panel has 3 Knobs

(i) Train no

(ii) Speed

(iii) Inertia.

Knobs*
train-Knob: integer Speed-Knob: Integer inlaia-Knob: unsigned Integer emergency-stop: Boolean

Pulsar*
Pulse-width: Unsigned Integer direction: Boolean

Sender*
Send-bit()

Detector*
<integer> read-bit(): integer

Fig - Classes describing analog physical objects

→ The panel also has one button for emergency stop.

→ When we change the train no setting, we want to reset the other Controls to proper values.

→ Knobs* → provide Set Knobs behaviar that allows rest of s/m to modify Knob settings.

→ The following fig shows the classes for panel & motar interfaces.

Panel
panel-active(): boolean train no(): integer speed(): integer inertia(): integer estop(): boolean newSettings().

Motar-Interface
Speed: integer

→ TX → Provides a distinct behavior for each type of message that can be sent.
→ RX → Provides a read-cmd behavior to read a message
→ Send-Command → Utility fn that serves as interface to transmitter.
→ operate → Performs basic actions for object.
→ Panel-active → Returns true whenever the panel values do not correspond to current values.
→ The role of formatter shows 2 changes to knob settings.

* speed
* inertia &
* emergency stop.

→ The panel is called periodically by the formatter to determine whether any control settings have changed.
→ If a setting has changed for the current train, then the formatter decides to send a command, by issuing a Send-Command to TX.
→ If the train number has changed, the formatter must cause the knob settings to get reset to proper values for new train.
→ The Controllers operate behavior must execute several behaviors in order to determine nature of message.

ARM Processor:

→ Advanced RISC Machine (ARM) processor is one of CPU based RISC (Reduced Inst^{n} Set Computer) architecture for Compiler processors.

→ ARM processor are used in music players, smart phones, wearables & other Consumer e^{c} devices.

→ ARM inst^{n} are written one per line.

→ Comments begin with semicolon & Continue to end of line.

→ Label → gives name to memory loc^{n}.

```
        LDR r0, [r8]    ; a Comment.
label   ADD r4, r0, r1
```

Processor & Memory Org^{n}:-

→ ARM7 is a Von Neumann architecture machine & ARM9 – Harvard arch.

→ ARM arch supports 2 types of data
- standard ARM word in 32 bit long &
- Word may be ÷ into 4 8 bytes.

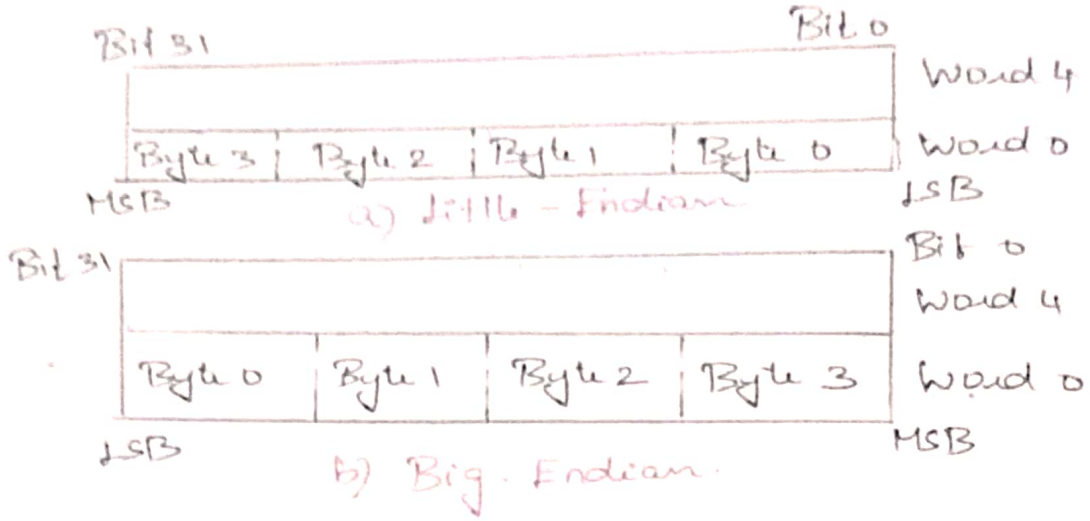
→ ARM7 allows addresses to be 32 bits long & address refers to byte not a word.

→ Word 0 → loc^{n} 0.

→ ARM processor can be Configured in a word either little endian or big endian mode.

→ Big Endian → stores MSB
little Endian → stores LSB.

→ PC is ↑ by 4.



Data Operations:-

→ In ARM processor, arithmetic & logical operⁿ cannot be performed directly on memory locⁿ.

→ ARM is a load-store architecture in which the data operands must first be loaded into CPU & then stored to main memory to save results.

ARM Programming Model:-

→ ARM has 16 general purpose registers, R0 - R15

→ R15 register has some capabilities as other registers but also used as Program Counter (PC)

CPSR:- (Current Program status Register). which is used by ARM Core to monitor & control internal fun.

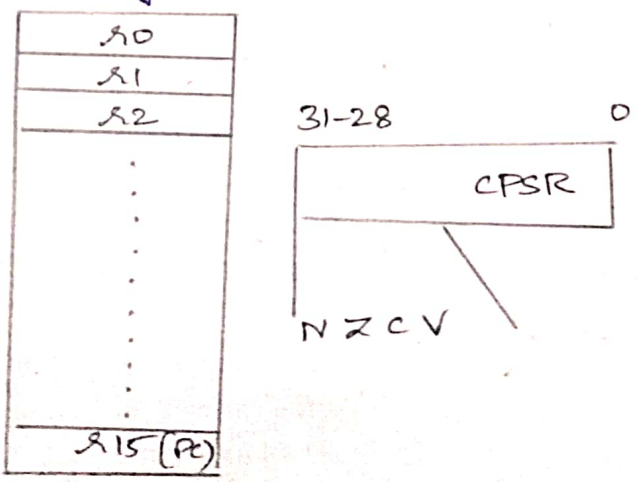


Fig - ARM prog model.

Instruction Set:

- → Arithmetic Inst°
- → Multiply Inst°
- → Logical Inst°
- → Shift & Rotate Inst°
- → Comparison Inst°
- → load & store Inst°.

① Arithmetic Inst°:-

ADD	Add two 32 bit values	$R_d = R_n + N$
ADC	Add 2 32-bit with Carry	$R_d = R_n + N + Carry$
SUB	Subtract 2 32 bit values	$R_d = R_n - N$
SBC	Subtract with Carry of two 32 bit	$R_d = R_n - N - 1$
RSB	Reverse subtract of 2 32bit	$R_d = N - R_n$
RSC	Reverse subtract with Carry of two 32 bit values.	$R_d = N - R_n - 1$

② Multiplication Inst°:

MLA	Multiply & Accumulate	$R_d = (R_m * R_s) + R_n$
MUL	Multiply	$R_d = R_m * R_s$

R_s → Source Register

Q MLA R_4, R_3, R_2, R_1 @ $R_4 = R_3 \times R_2 + R_1$.

3) Logical Instⁿ :-

AND	Logical bitwise AND of 2 32 bit	Rd = Rn AND N
ORR	Logical bitwise OR of two 32 bit	Rd = Rn ORR N
EOR	Logical Exclusive OR of two 32bit	Rd = Rn EOR N
BIC	Logical bit clear (AND NOT)	Rd = Rn AND NOT N.

Eg.

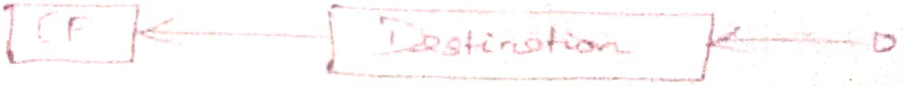
AND R0, R1, R2 @ R0 = R1 AND R2
ORR R0, R1 R2 @ R0 = R1 OR R2
EOR R0, R1, R2 @ R0 = R1 XOR R2
BIC R0, R1, R2 @ R0 = R1 AND (~R2).

4) Shift & Rotate Instⁿ :-

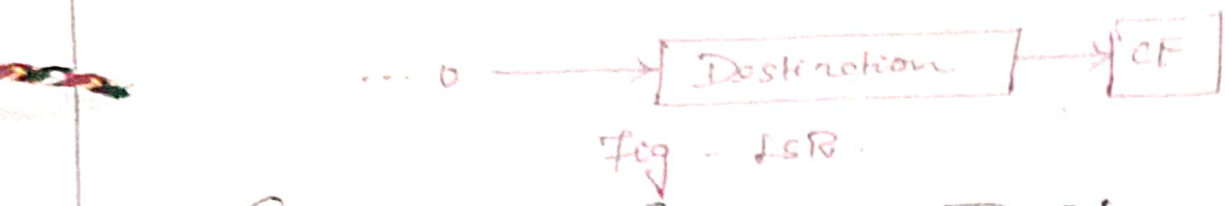
LSL	Logical shift left
LSR	Logical shift Right
ASL	Arithmetic shift left
ASR	Arithmetic shift Right
ROR	Rotate Right
RRX	Rotate Right extended with C.

→ Shift Modifier is always applied to second Source operand.

→ LSL & LSR → left & Right logical shifts, filling LSB of operands with Zeros.



This - LSL



Ex: LSR

5) Compare (or) Comparison Inst^n:-

→ Compare Inst^n are used to Compare or test a register with 32 bit value.

→ They update CPSR flag bits, but do not affect other registers.

→ After bits have been set, the inf^n can be used to change pgm flow by using Cond^n execut^n

CMP	Compare	Flags set as a result of $R_n - N$
CMN	Negated Compare	Flags set as a result of $R_n + N$
TST	Bit-wise test	Flags set as a result of R_n AND N
TEQ	Bit-wise negated test	Flags set as a result of R_n EOR N.

Ex:
CMP R1, R2 @ R1 − R2
CMN R1, R2 @ R1 + R2
TST R1, R2 @ R1 AND R2
TEQ R1, R2 @ R1 EOR R2.

6) Move Inst^n:-

→ It Copies the value (N) of second operand into Rd (Dest^n register)

N → Register or immediate Value.

MOV →	Move	Move 32 bit Value into Register
MVN →	Move Negated	Move NOT of 32 bit into Register

Eg. MOV R0, R1.

7) Load & Store Instⁿ:-

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.

Ex

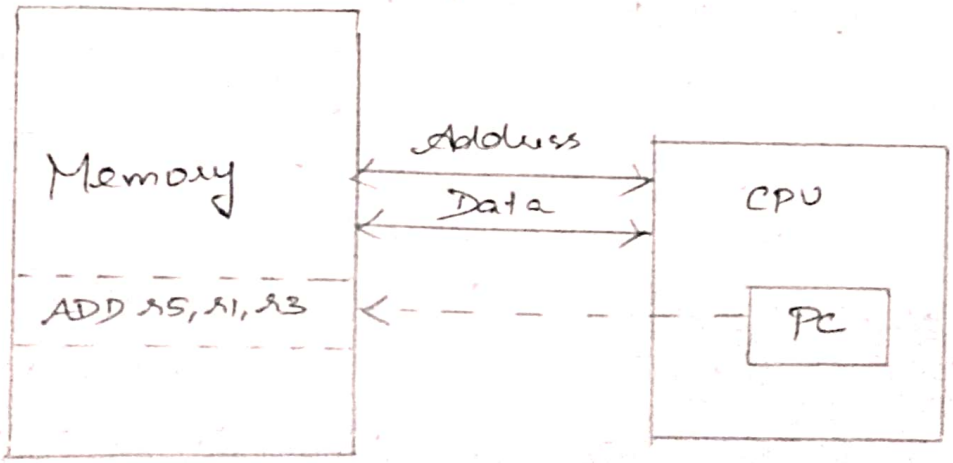
LDR Ra, [Rb]

STR Ra, [Rb]

Preliminaries :-

① Computer Architecture Taxonomy :-

Von Neumann Architecture.



→ Computing sys consists of CPU & memory.
→ A Computer whose memory holds both data & Instr is known as Von Neumann machine
→ CPU has several internal registers that store values used internally.
→ PC is a register which holds the address in memory of an instr.

Harvard Architecture :-

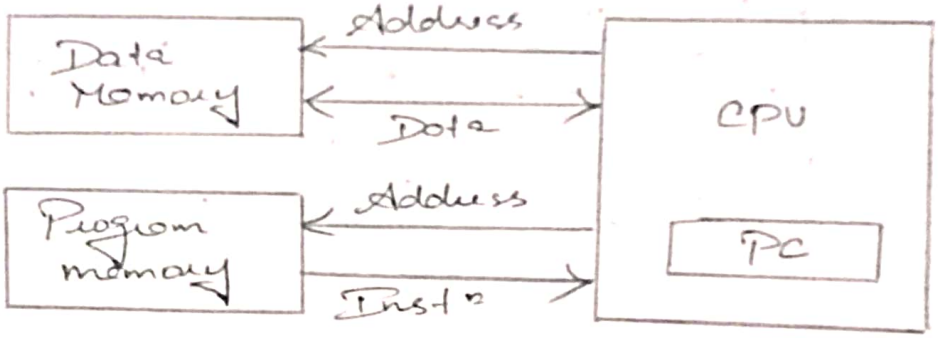


Fig - Harvard Architecture

→ It has separate memories for data & program.
→ PC points the program memory, not data memory.
→ Due to separ of program & data memories, it provides higher performance for digital signal processing.
→ Data sets that arrive continuously & periodically are called streaming data.
→ Two memories with separate parts provide higher memory bandwidth.

RISC Vs CISC :-

CISC → Complex Inst^n Set Computers

→ It provides a variety of Inst^s that may perform very Complex tasks & used a no. of diff inst^c formats of varying lengths.

→ A single inst^n do all loading, evaluating & storing operations.

RISC → Reduced Inst^r Set Computers.

→ It uses fewer & simpler inst^s.

→ a simplify h/w is obtained by using an inst^s set composed of few basic steps for loading, evaluating & storing oper^s just like a load command will load data, store command will store the data.

Instructions Set Characteristics :-

→ Inst^s Set of Computer defines the interface b/w software modules & underlying h/w.

→ Inst^s define what the h/w will do under Certain circumstances.

→ Characteristics are as follows

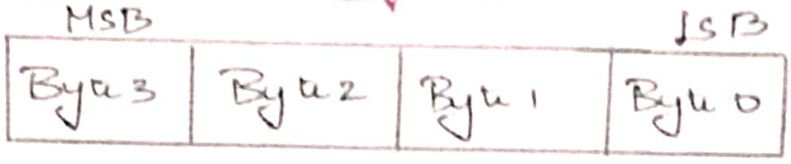
- Fixed Vs Variable length
- Addressing modes
- No. of operands
- Types of oper^2.

Word Length :-

→ Word length includes 4 bit, 8 bit, 16 bit, 32 bit & so on.

→ Length of data word, an inst^r & an address are same.

6) Little-endian Vs Big-endian



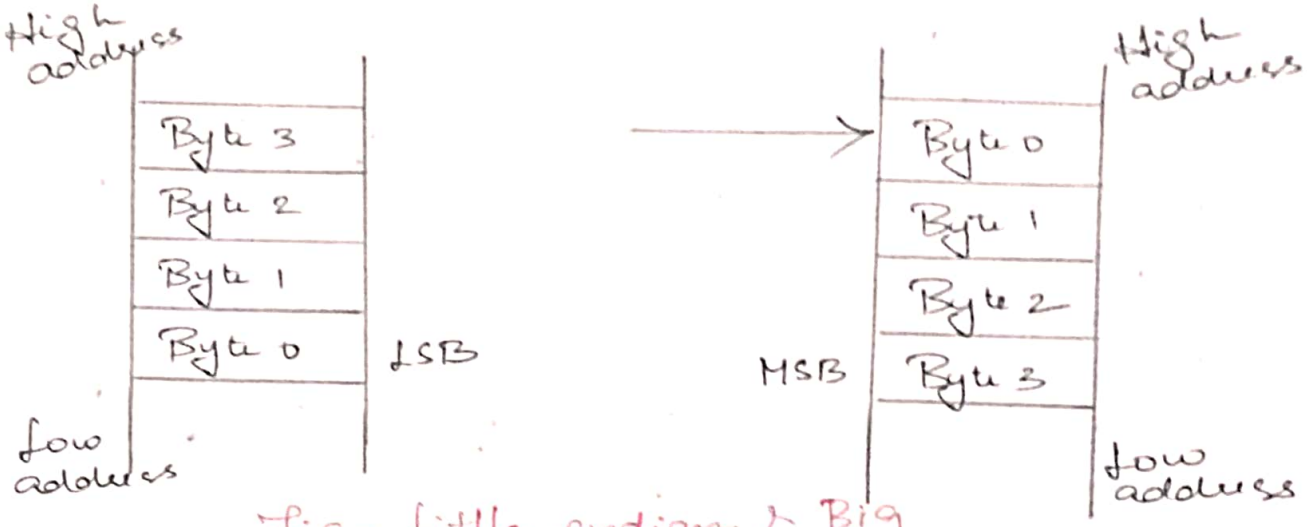


Fig- little endian & Big endian.

→ Big endian s/m stores → MSB of word at smallest memory address & LSB at largest.

VI) CPU:-

Programming i/p & o/p:-

① Input & o/p Devices:

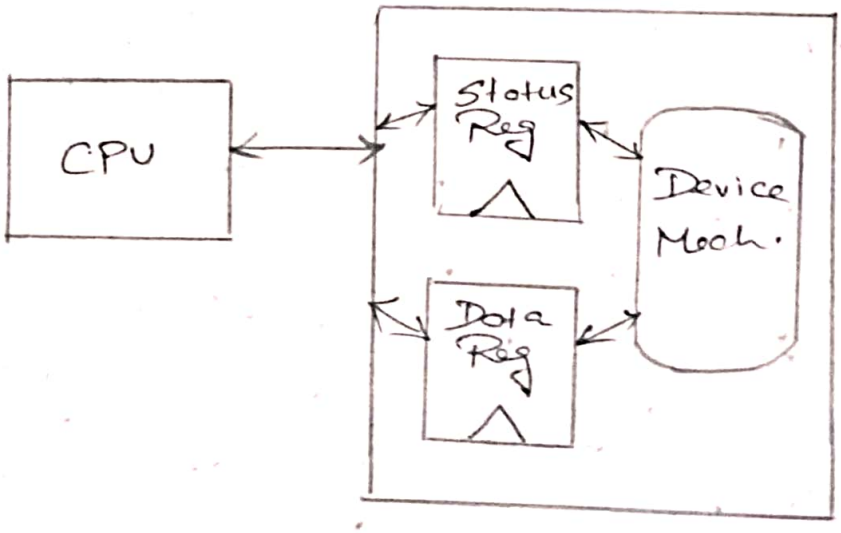


Fig- Structure of Typical I/O device.

→ Interface b/w CPU & device internals is a set of registers.

(i) Data Registers – hold values that are treated as data by the device such as data read or written by disk.

(ii) Status Registers – Provide info about device oper^{n} such as whether current trans^{n} has completed.

(ii) Input & Output Primitives:

→ There are 2 methods of interfacing I/P & O/P

(i) Memory mapped I/O

(ii) I/O mapped I/O.

(i) Memory mapped I/O:-

→ In memory mapped, i/P / o/p devices are mapped to the memory address space of µP.

→ This means I/O devices are treated like memory loc^{n} & can be accessed using same read & write inst^{n} as memory.

(ii) I/O mapped I/O.

→ The I/P & O/P devices are mapped to a separate I/O address space which is different from memory address space.

→ The µP uses special inst^{n} to access the I/O devices using I/O address signals which are separate from memory address signals.

(iii) Interrupts:-

→ The interrupt mechanism allows devices to signal the CPU & to force execut° of particular piece of code.

→ When interrupt occurs, the PC changed to point to interrupt handler routine that takes care of device: writing next data, reading data & so on.

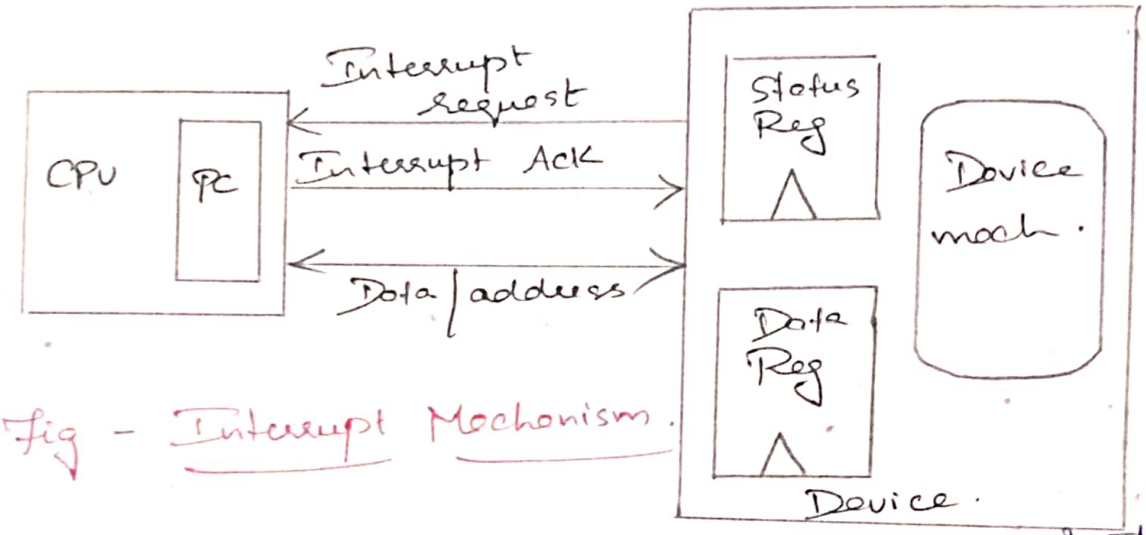


fig - Interrupt Mechanism.

→ Above fig shows interface b/w CPU & I/O devices.

(i) I/O device asserts interrupt request signal when it wants service from CPU

(ii) CPU asserts interrupt acknowledge signal when it is ready to handle I/O device request.

→ I/O device logic decides when to interrupt.

Priorities & Vectors:-

→ Some mechanism are followed for allowing multiple device to interrupt.

→ There are 2 ways to handle multiple devices

(i) Interrupt Priorities → allow CPU to recognize some interrupts as important as others

(ii) Interrupt Vectors allow interrupting device to specify its handler.

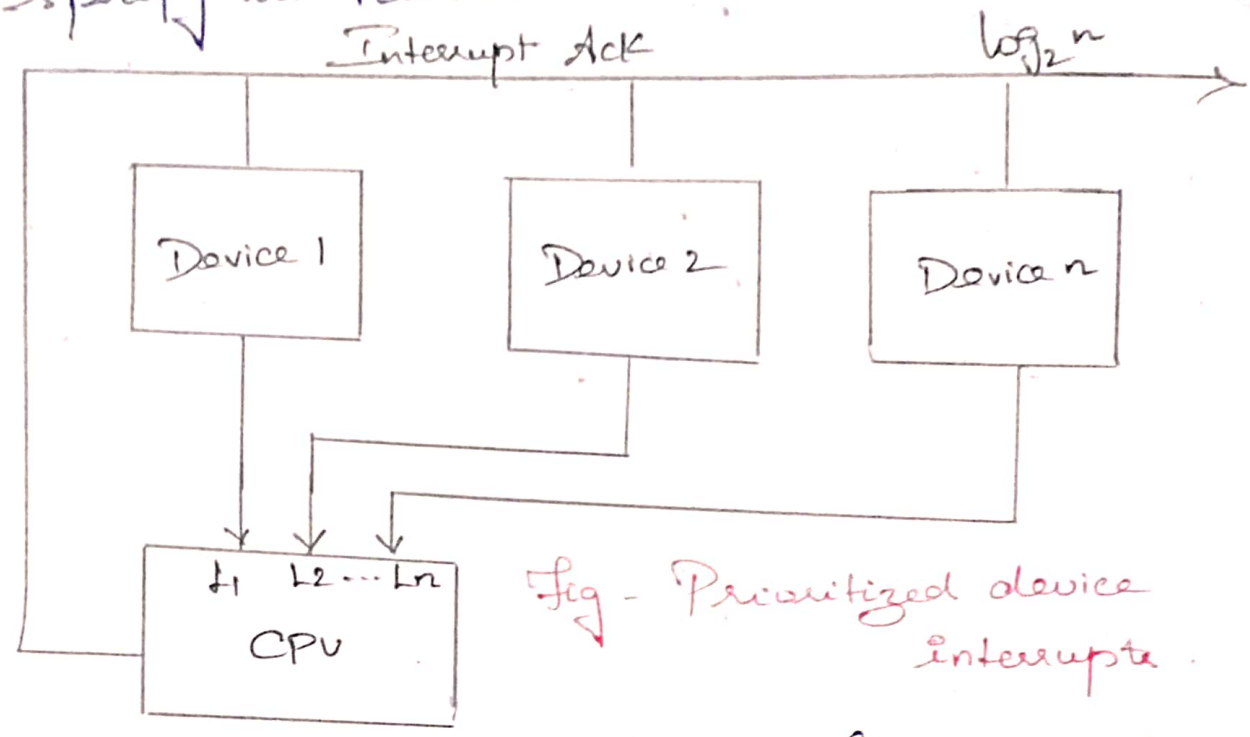


Fig - Prioritized device interrupts.

→ Lower numbered interrupt lines are given higher priority.

Masking:

→ The priority mechanism must ensure that lower-priority interrupt does not occur when higher-priority interrupt is being handled, then the decision process is known masking.

Interrupt Vectors:-

→ It provide the ability to define the interrupt handler that should service a request from a device which is more flexibility.

→ Interrupt vector lines run from device to CPU.

→ CPU then uses the vector no as a index in a table stored in memory as shown in fig below.

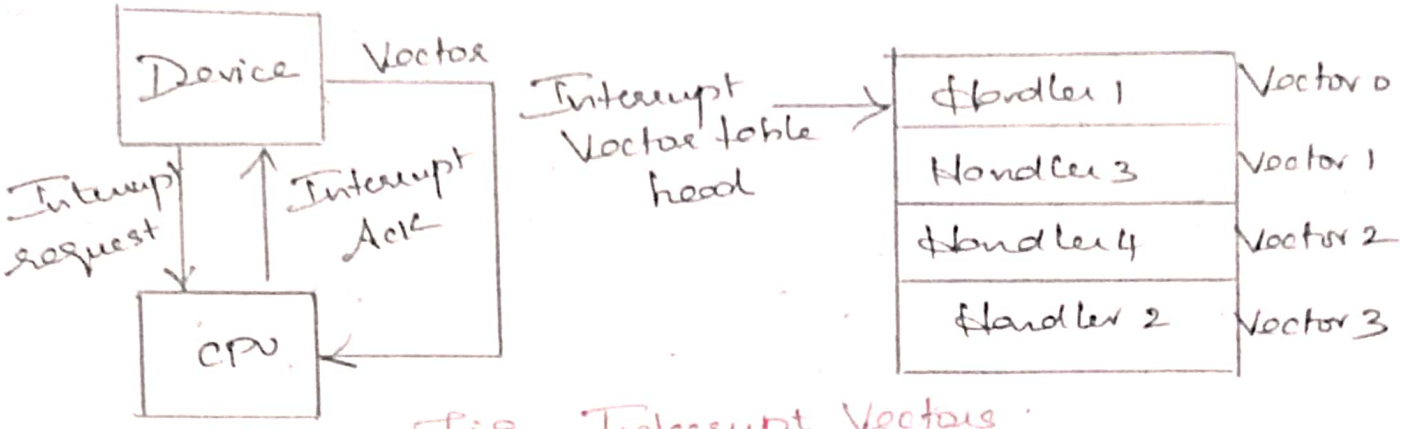


Fig- Interrupt Vectors

Interrupt Overhead:

→ Once a device requests an interrupt, the following steps are performed.

(i) CPU :-
→ CPU checks the pending interrupts at the beginning of inst^n to find highest-priority interrupt.

(ii) Device :-
→ The device which receives the acknowledgement sends its interrupt vector to CPU

(iii) CPU :-
→ CPU looks up the device handler address in interrupt vector table using this vector as index.

(iv) Software :-
→ The device driver may save additional CPU state, then it performs the required oper^n on the device which restores any saved state & execute the interrupt return inst^n.

(v) CPU :-
→ The interrupt return inst^n restores the PC & other automatically saved states.

Supervisor Mode:

→ It is an automatic selected mode when a Computer is powered on (ii) processor enters supervisor mode on reset.

→ When a program is in user mode, wants to execute a privileged task, for eg allocating more memory, it needs to make s/m call.

Mode	Privileged	Purpose.
User	No	Normal operating mode for most pgm
Fast Interrupt Request	yes	Used to handle a high priority Interrupt
Interrupt Request	yes	Used to handle low priority interrupt
Supervisor	yes	Used when processor is reset
Abort	yes	Used to handle memory acess violations
Undefined	yes	Used to handle undefined or unimplemented inst?
System	yes	Uses same registers as user mode

Features:-

→ It handles different types of Commands but mostly deals with privileged inst?.

→ The OS selects supervisor mode for low level tasks that require Complete acess to s/m.

→ It Can Create the memory address spaces as well as updating them.

→ In ARM, Software Interrupt (SWI) inst^n puts the CPU in Supervisor mode.

SWI CODE_1.

→ In Supervisor mode, bottom 5 bits of CPSR are all set to 1 to indicate that CPU is in Supervisor mode.

Exceptions :-

→ An exception is an internally detected error.

→ It provides a way for pgm to react for an unexpected event.

Eg: → Resetting ARM Core
→ Failure of fetching inst^n
→ Illegal memory acces.
↳ Externally generated interrupts.

Exceptions & Modes :-

Exception	Mode	Purpose.
Fast Interrupt Req	FIQ	Fast interrupt handling
Interrupt Req	IRQ	Normal interrupt handling
SWI & RESET	SVC	Protected mode for OS
Pre fetch or data abort	ABT	Memory protect^n handling
Undefined Inst^n	UND	SW emulation of HW Coprocessors.

→ Exceptions require both prioritization & vectoring.
→ It must be prioritized because single oper^n may generate more than one except^n.
→ Vectoring provides a way for the user to specify the handler for exception cond^n.

Traps:-

→ A trap is an inst^n that generate an exception cond^n which is also known as software interrupt.
→ The entry into supervisor mode must be controlled to maintain security.
→ The ARM provides SWI interrupt for software interrupts. This inst^n causes CPU to enter into supervisor mode.

Models of Programs:-

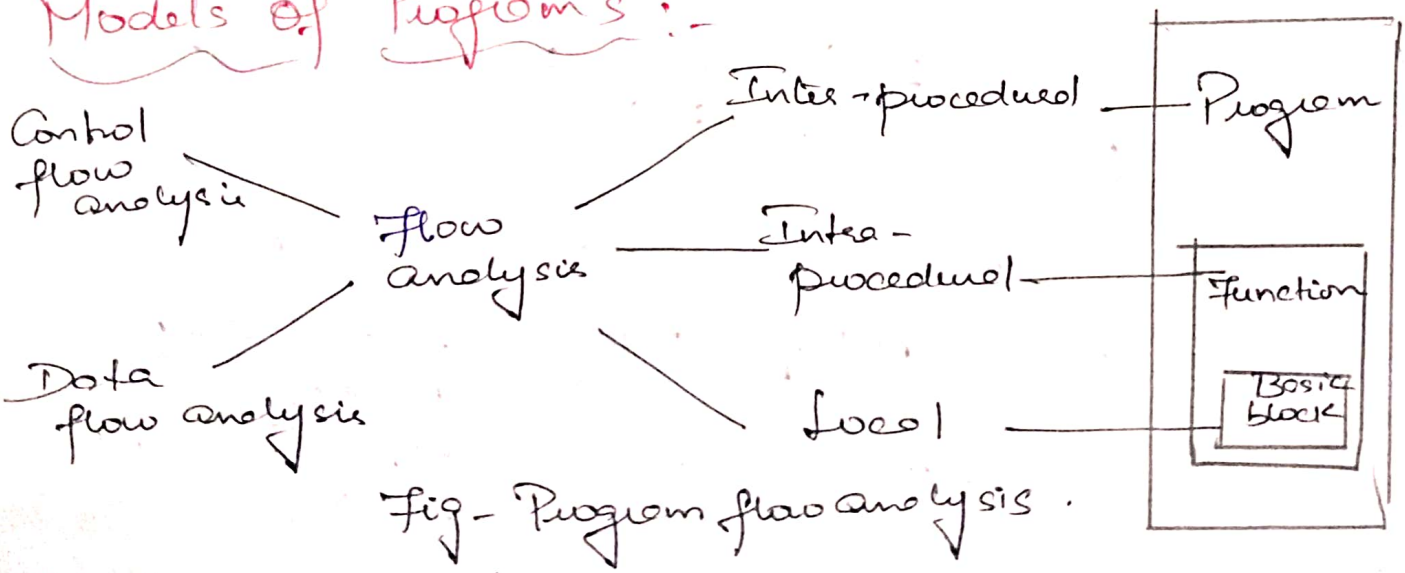


Fig- Program flow analysis.

① Control flow Analysis:-
It determines the control structure of a pgm & builds control flow graphs.

(ii) Data flow Analysis.

→ It determines the flow of data values & builds Data flow graphs (DFGs)

DFG:

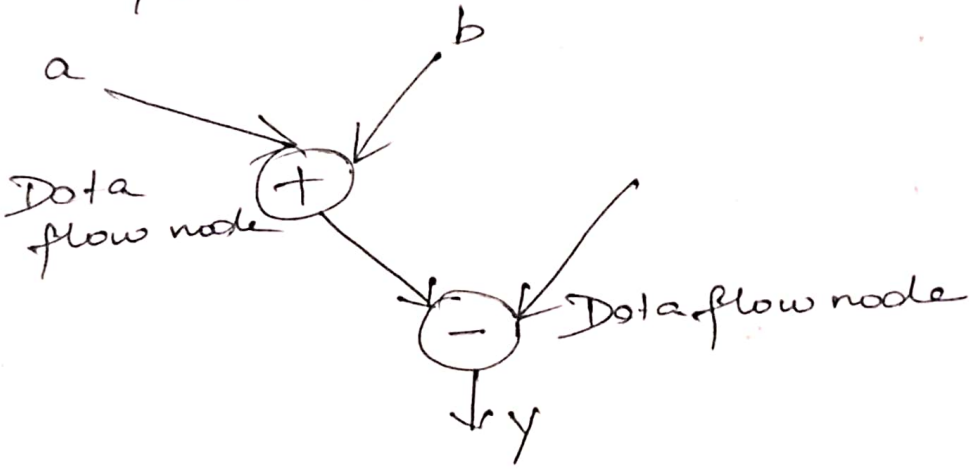
→ DFG model translate data processing requirements into graph.

→ DFG is a visual model in which operation on data is represented using a block & data flow is represented using arrows.

→ Inward arrow – Input data
Outward arrow – Output data.

→ A Data path is the data flow path from i/p to an o/p.

eg
```
x = a+b;
y = x-c;
```



Control flow graph (CDFG):-

→ It Construct the model for both data operation & Control operation.

→ CDFG have 2 type of nodes
(i) Decision Node
(ii) Data flow Node.

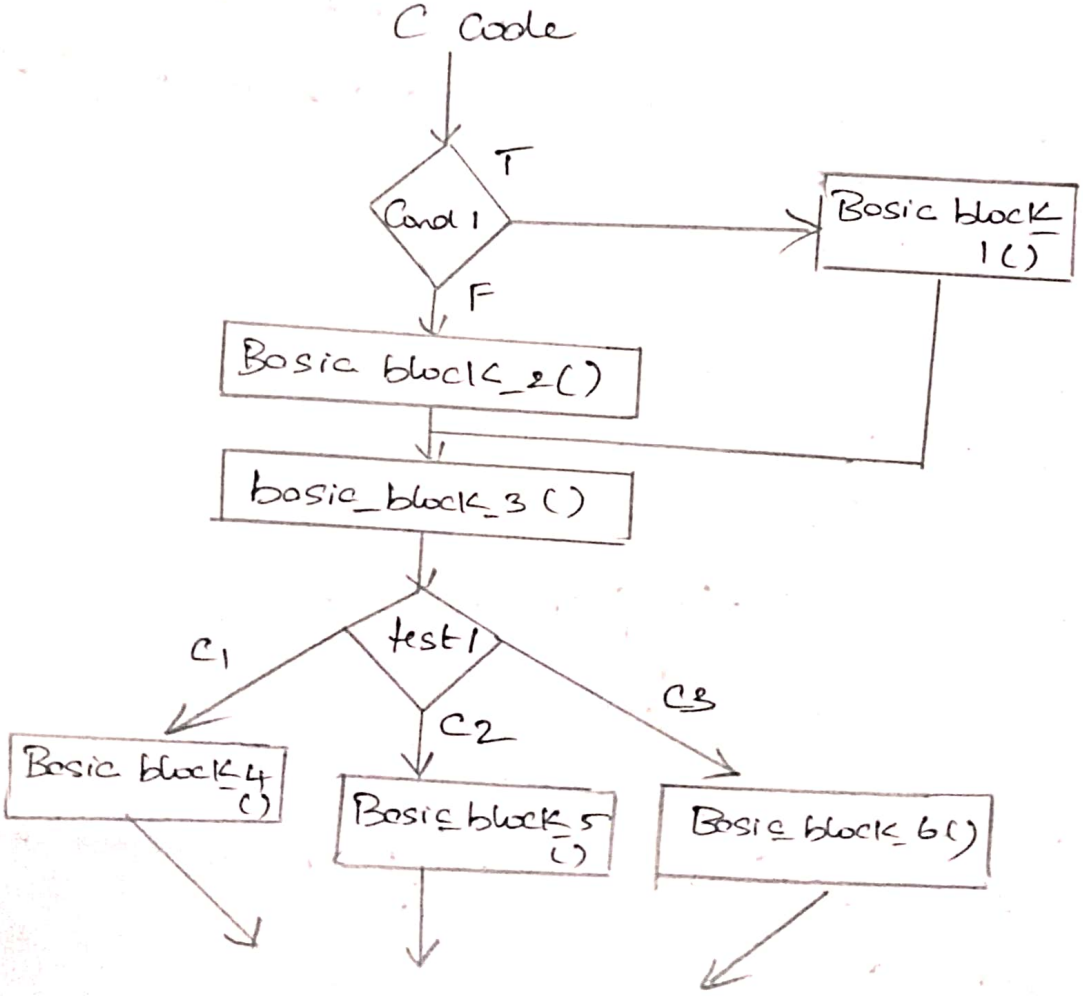
→ Data flow Node encapsulate a Complete data flow graph to represent basic block & a

decision node is used to describe all types of control in sequential pgm.

→ The rectangular nodes in graph represent basic blocks.

→ The Diamond shape nodes represent the conditionals.

→ Node's condition is given by label & edges are labeled with possible outcomes of evaluating the cond².

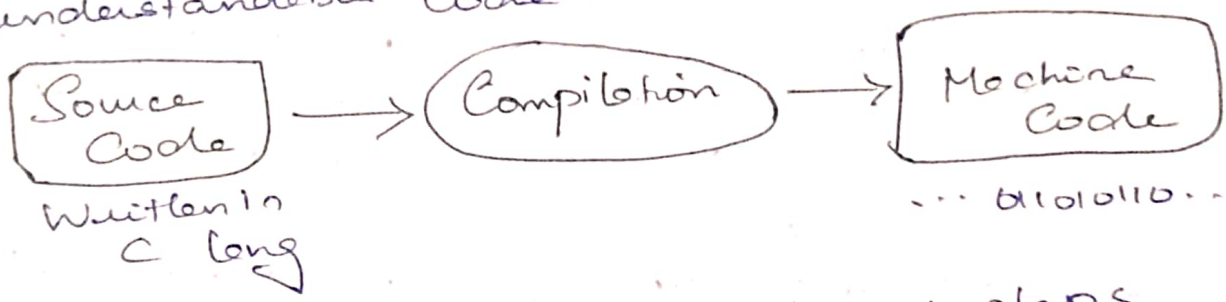


CDFG.

fig - CDFG

Assembly, Linking & Loading:-

→ The Compilation process is converting an understandable human code into machine understandable code.



Compilation process in C involves 4 steps

(i) Preprocessing
(ii) Compiling
(iii) Assembling
(iv) Linking.

Assembler:

→ The assembler must translate opcodes & format the bits in each inst^n & then translate the labels into address.

→ Labels make the assembly process more complex, but they are most impt abstraction provided by assembler.

(i) 1st pass → Scans the code to determine the address of each label

2nd pass → assembles the inst^n using label values computed in 1st pass.

Symbol Table:

PLC → xx
44

add R0, R1, R2
add R3, R4, R5
Cmp R0, R3
Sub R5, R6, R7

xx 0X8
44 0X10

→ The symbol table is built by scanning from the 1st instr to the last.
→ During Scanning, the current loc in memory is kept in PLC.
→ At start of 1st pass → PLC is set to program starting address & assembler points to 1st line.
→ After exoming the line, assembler updates PLC to next loc & looks at next instr.
→ During 2nd pass - when label name is found, label is looked up in symbol table & its value substituted into appropriate place in instr.

Linking

→ A linker is a s/w tool that plays a crucial role in Compiling process of program.
→ It takes object code generated by an assembler & combines with other libraries.
→ Linker operates on object files created by assembler & modifies assembled code to make necessary links b/w files.

```
label1  LDR R0,[R1]
        ...
        ADR a
        ...
        B label2
        ...
var1    y. 1
```

```
label2  ADR var1
        ...
        B label3
        ...
x       y. 1
y       y. 1
a       y. 10
```

External References	Entry points
a	label1
label2	var1

External References	Entry points
var1	label2
label3	x
	y
	a.

Compilation Techniques

→ Compilation is a process of converting the source code into object code.
→ It is done with help of compiler & an assembler.
→ The compiler checks the source code for syntacial or structural errors.
→ This assembly code is then converted into object code by using assembler.

Compilation = Translation + optimization.

→ Compilation begins with high level language code such as C or C++.

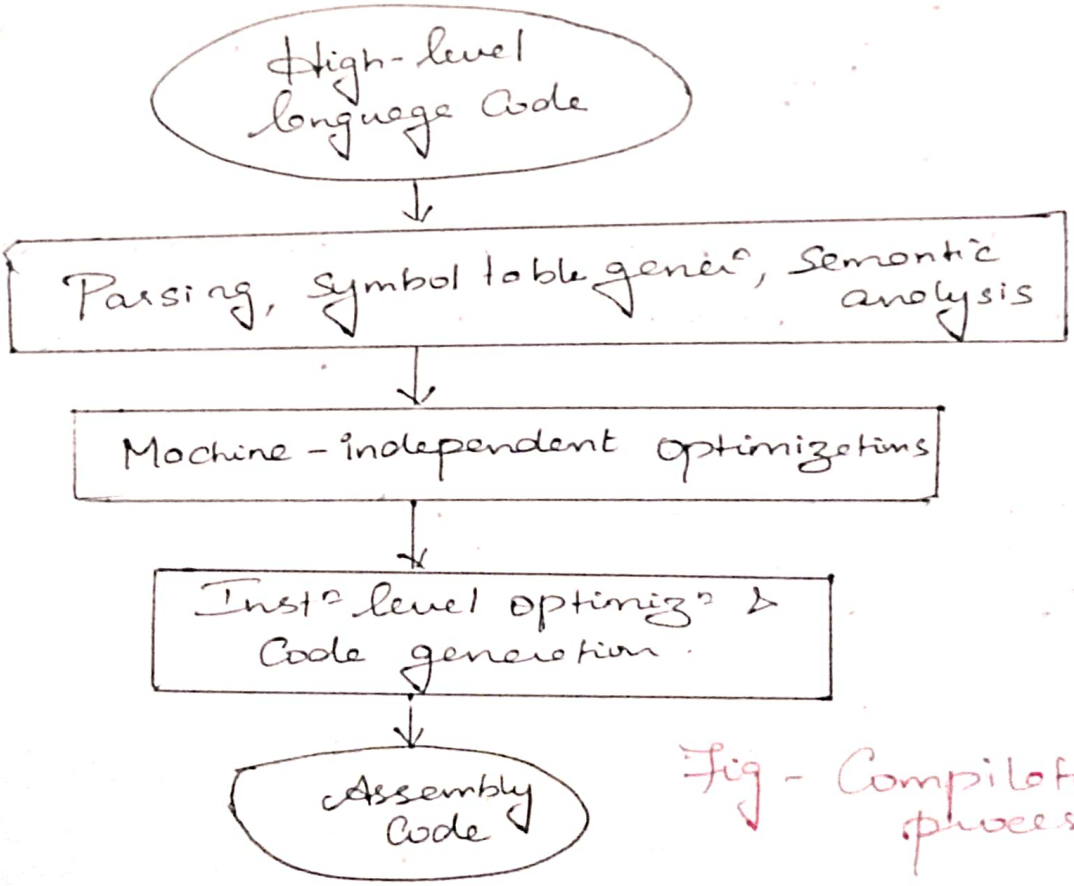


Fig - Compilation process.

→ High level language pgm is parsed to break it into statements & expressions.

Basic Compilation Methods

① Procedures:-

→ Procedures are known as functions in c that requires a specialized code.

→ The info for a call to procedure is known as a frame.

→ The frames are stored in a stack to keep track of the order in which the procedures have been called.

→ Procedure stacks are typically built to grow down from high addresses.

→ A stack pointer (SP) defines the end of current frame while frame pointer (FP) defines the end of last frame.

② Data Structures:-

→ The Compiler must also translate references to data structures into references to raw memories.

→ Address of an array element must be computed at run time because the array index may change.

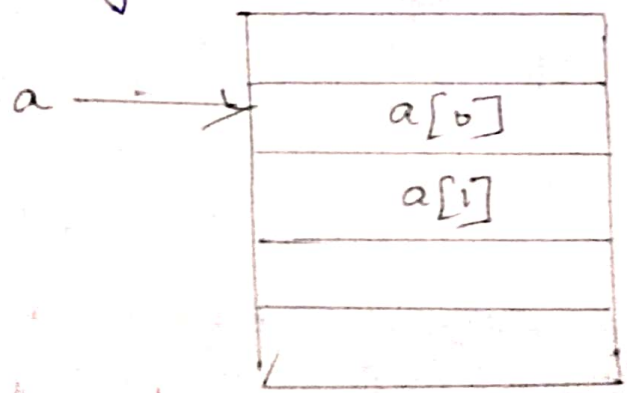


fig - Layout of one dimensional array in memory

→ We use pointer aptr for reading a[1] as
*(aptr+1)

→ a[] → array with size of M×N.
a[i,j]

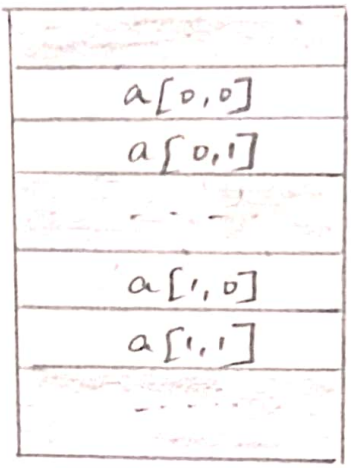


Fig. Memory layout for 2D arrays.

Compiler optimizations :-

→ Basic Compilation techniques can generate inefficient code.

Loop Transform : Loop optimiz Techniques :-

→ Loop optimization is the process of Trg an ~~execution~~ execution speed & reducing the overheads associated with loops.

Loop Unrolling :-

→ loop unrolling is a loop transf technique that help to optimize the execution time of pgm.
→ It increases the pgm speed by eliminating loop control inst & loop test inst.

Loop fusion :

loop fusion is combining two or more loops in a single loop which reduces the loop overhead & also the time taken to compile many loops.

Loop Distribution:

→ It is a Compiler optimiz^n tech in which a loop is broken into multiple loops over the same index range with each taking only a part of original loop body.

Loop Tiling:-

→ It breaks up a loop into a set of nested loops & each inner loop performing the oper^n on subset of data.

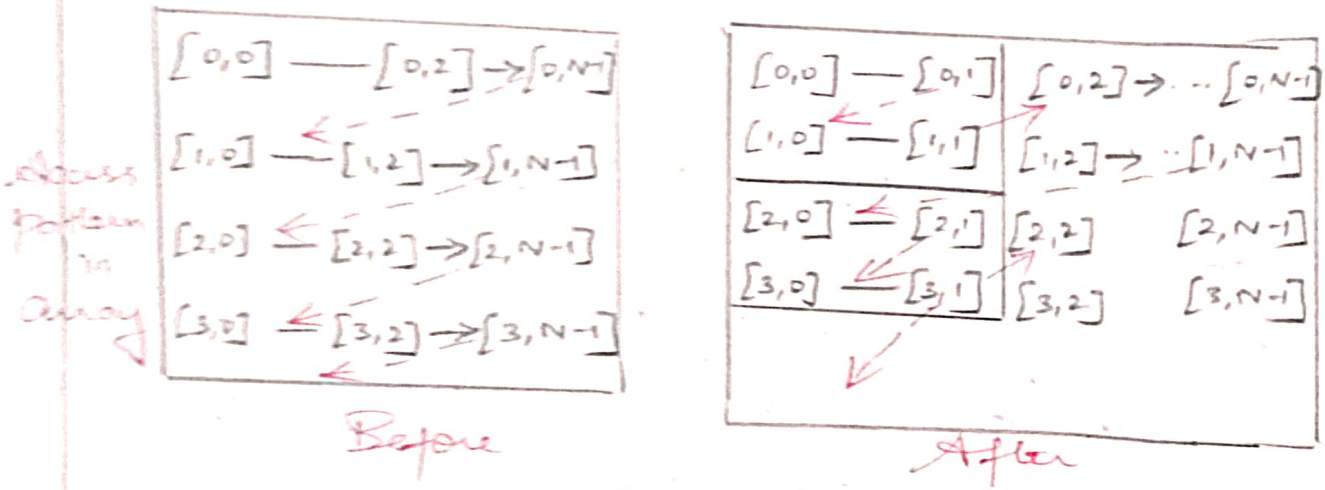


Fig - Loop Tiling

Dead Code Elimination:

→ Dead Code is the Code that Can never executed.

→ It Can be identified by reachability analysis. (i.e) finding other statements or instr^s from which it Can be reached.

Register allocation:

→ It is a final phase of Compiler.

→ Registers are faster to access than Cache memory.

Scheduling

Instruction Scheduling:

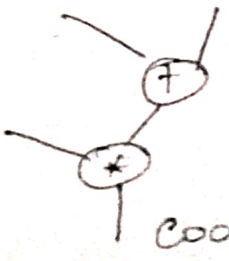
→ It is a process of mapping a series of inst° into execut° of resources.
→ It decides when & where inst° is executed.
→ Rows – represent inst° execution time slots
Col° – represent resources that must be scheduled

Time	Resource A	Resource B
t	X	
t+1	X	X
t+2	X	
t+3		X

Before Scheduling – check the reservation table to determine whether all resources needed by inst° are available at that time.

Inst° Selection:-

→ Selecting the inst° to implement each oper° is a difficult task.
→ Using one inst° for one part of Pgm may affect the inst° that can be used in adjacent code.
→ One useful technique for generating code is <u>template matching</u>.



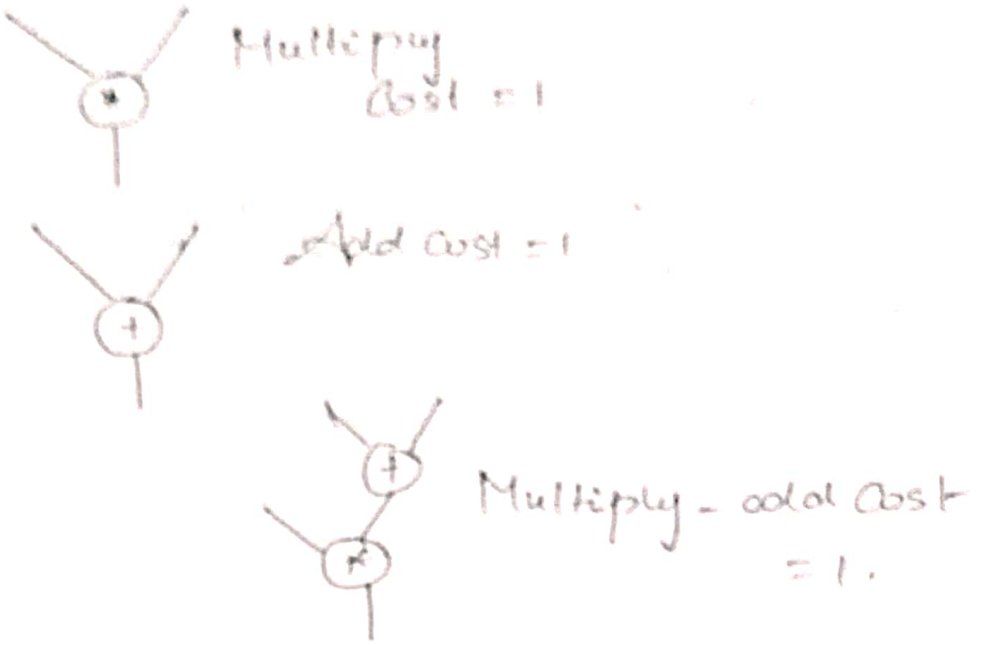


Fig. Code generation by template matching

Program Level Performance Analysis:

→ The technique used to analyze program execution time are helpful in analyzing properties such as power consumption.

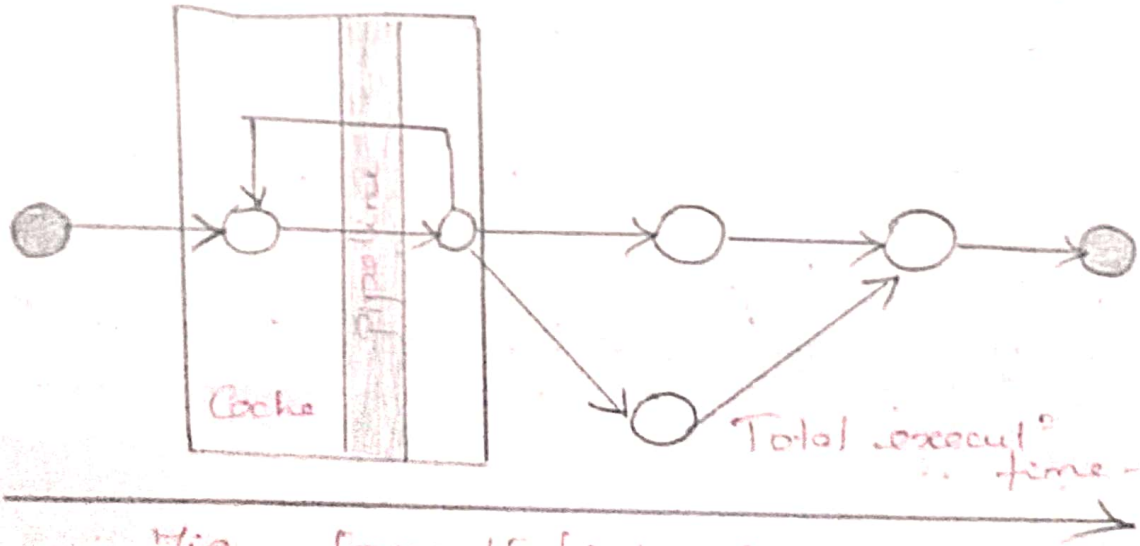


Fig - Execution time of a pgm.

→ Consider the above fig, the CPU pipeline & Cache act as windows into our pgm.

→ The pipeline & Cache influence execut° time, but execut° time is a global property of pgm.

→ It is difficult to Calculate execut° time of pgm because

(i) The execut° time of pgm often varies with i/p data values because those values select diff execut° paths.

(ii) The Cache has a major effect on pgm performance

Measuring Execution Speed :-

→ 3 diff types of performance measures on

(i) Average Case Execution time :-

→ This execut° time would expect for typical data. First, we clearly define the typical i/p.

(ii) Waist Case Execution time :

→ This is longest time that the pgm can spend on any i/p sequence.

(iii) Best-Case Execution time :-

→ It is the shortest time that the pgm can spend on any i/p sequence.

→ It is impl in multirate real time s/m.

Execution time = Program path + Instruction timing.

Unit 3

Processes & Operating Systems.

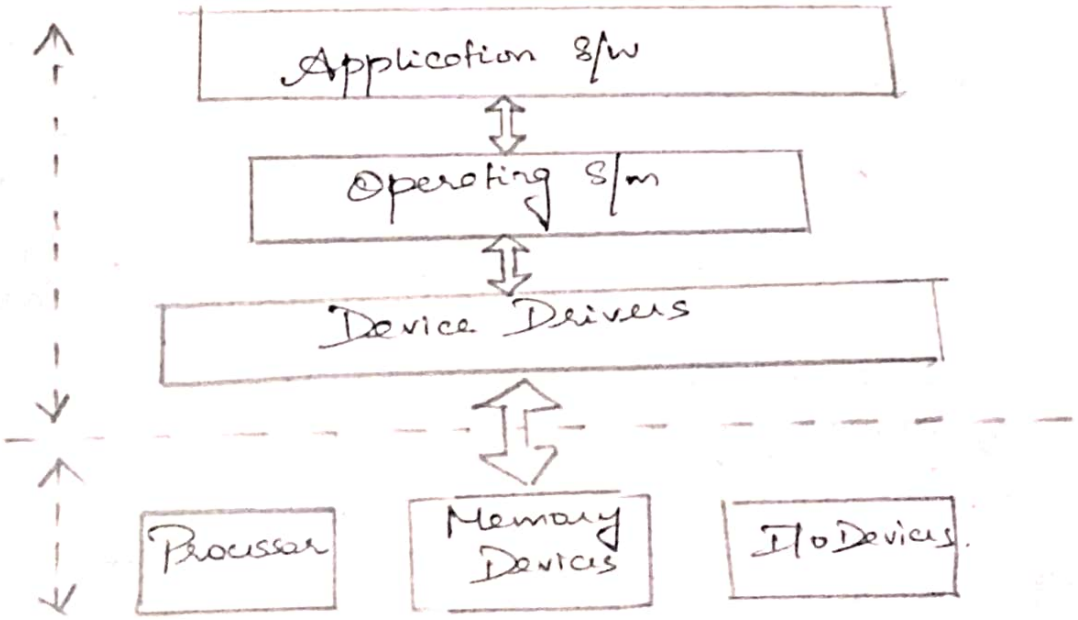
Structure of a real-time system - Task Assignment & Scheduling - Multiple tasks & Multiple processes - Multirate systems - Pre emptive real-time operating systems - Priority based Scheduling - Interprocess Communication System - Distributed Embedded s/m - MPSoC & shared Memory multiprocessors - Design Example - Audioplayer - Engine Control Unit & Video Accelerator.

1) Structure of Real time system:

Real time system is a system which is used for performing specific tasks.

→ These specific tasks are mainly related to time Constraints.

Structure of Embedded s/m:-



- The lower layer is embedded h/w which consists of
  - The processor
  - Memory Devices
  - I/O Devices.

→ Upper layer is the embedded s/w.
— Device drivers which are s/w written to directly control the embedded h/w & provide an Appl° Programming Interface (API) to upper Software layers.
— Appl° s/w is the s/w that performs the main fun° that the s/m was built for.

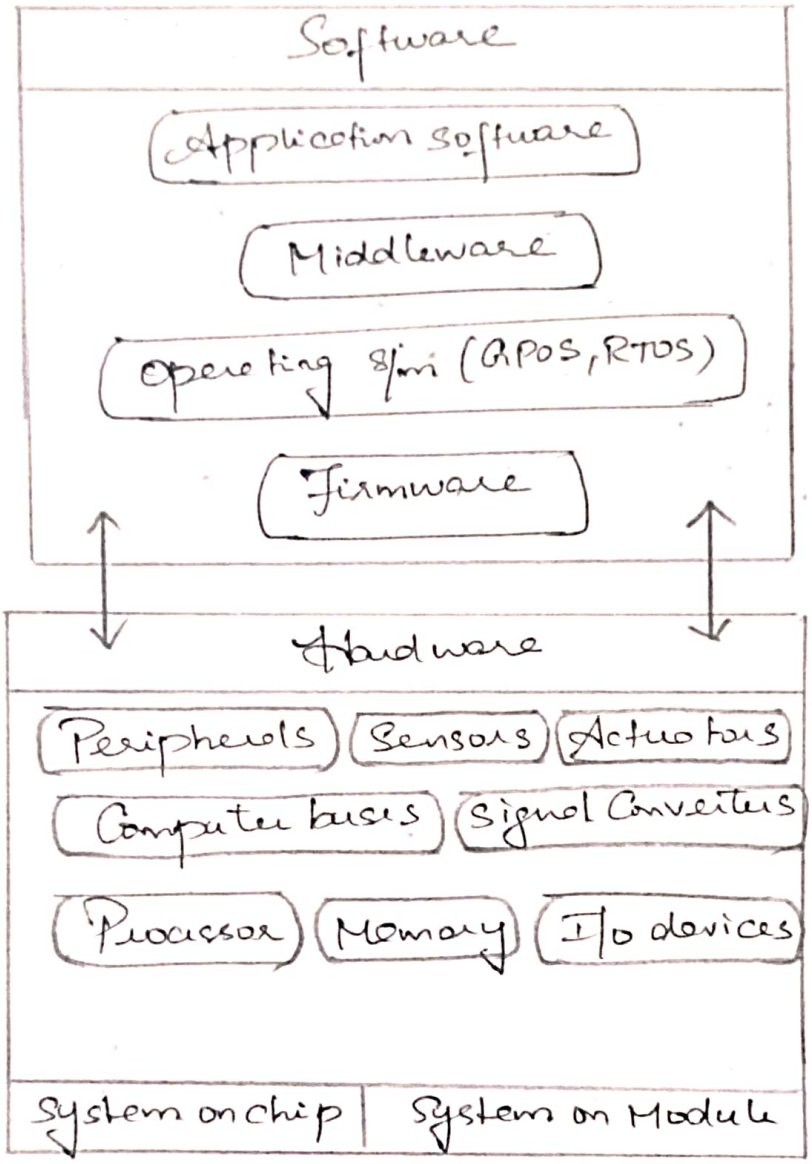


Fig - Embedded system structure.

Structure of Embedded Real time s/m:-

→ It includes various h/w & s/w devices.
→ Following fig represents the structure of Embedded Real time- s/m.

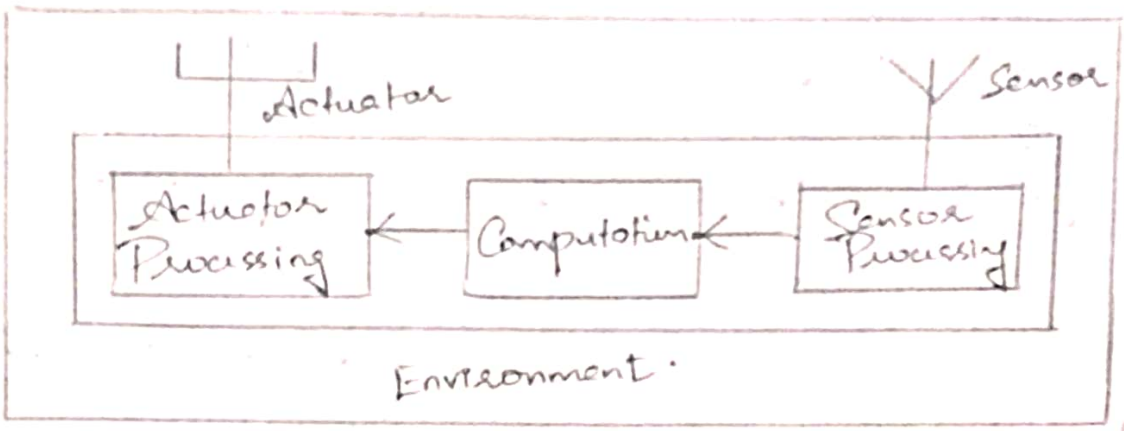


Fig - Structure of Embd Real time s/m.

① Sensor:
→ It is used to sense the environment periodically
→ It is used for Conversion of some physical events into elec signals.
→ The sensed data from environment is processed to determine the necessary Corrective actions.

(ii) Sensor Processing:
→ When data is sensed from environment, it makes data ready for Computation process.
→ It involves both Conditioning & Other processes.

(iii) Computation:
→ It is a procss of Calc & operations needed for task to be Completed.
→ It takes data i/p from Sensor & gives output to actuator of real time s/m.

(iv) Actuator Processing:
→ It takes i/p from s/m & gives this to an actuator of s/m.
→ This is basically used to make o/p Compatible with environment.

(v) Actuator:
→ It is reverse device of sensor.
→ It is used to Convert elec signals into physical events or characteristics.

11) Task assignment & Scheduling:

Task assignment:

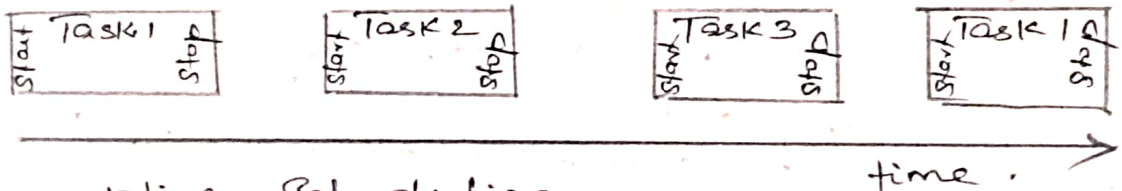
→ It is the process of allocating specific tasks or responsibilities to individuals or teams within an organiz°.
→ It ensures that work is distributed efficiently & effectively among team members
→ It allows for better utiliz° of individual skills
→ It helps in balancing workload & avoiding bottlenecks.
→ It promotes accountability & clarity reg responsibilities

When assigning tasks,

→ Individual skills, knowledge & expertise required
→ Availability & workload of team members
→ Deadlines & priority of tasks
→ Comm° & Collaboration requirements
→ Balancing workload & avoiding overburden.

Scheduling:

"The time allocated b/w tasks is termed as Scheduling". Scheduler is the s/w that determines which task should run next.



Pre-emptive Scheduling

→ It is commonly used scheduling alg.
→ Here tasks are prioritized & task with highest priority among all other tasks get CPU time.

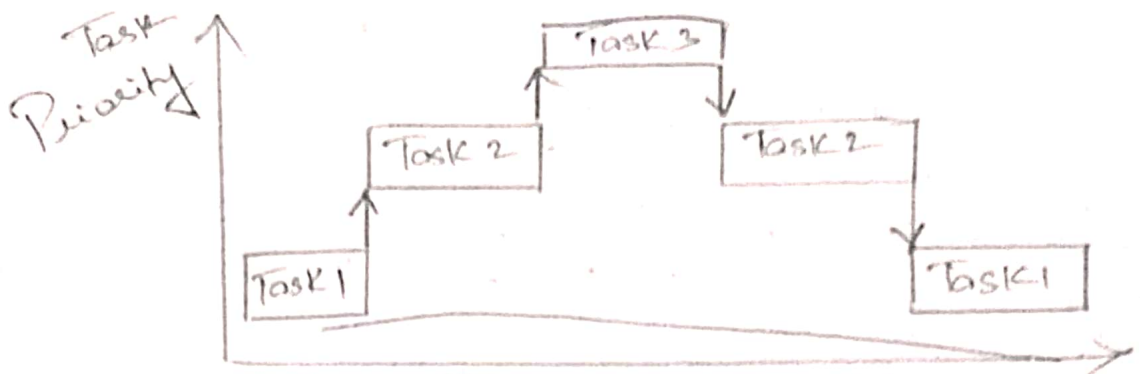


Fig. Preemptive Scheduling

Non Preemptive Scheduling:-

→ Even if the highest priority is allocated to the task, it needs to wait until the Completion of Current task.

→ This task Can be either slow or lower priority which Can lead to longer wait.

→ Better approach is designed by Combining both preemptive & non preemptive scheduling which is Called time based interrupts in priority based system.

Classification of Task Scheduling Algorithms:-

In static priority algorithms, task priority does not change with a mode.

In dynamic priority Algorithms, priority Can change with time.

(i) Static Table-Driven approaches:

→ It perform a static analysis associated with Scheduling & Capture the schedules that Can point out a task with which the execut⁻ must be started at run time.

(ii) Static Priority Driven Preemptive approaches:
→ These type of Algorithms also use static analysis of Scheduling.
→ It is a useful way of assigning priorities among various tasks in preemptive Scheduling.

(iii) Dynamic planning based approaches.
→ The feasible Schedules are identified dynamically.
→ It carries certain fixed time interval & a process is executed if & only if satisfies the time Constraint.

(iv) Dynamic Best effort approaches:-
→ The task is aborted if its deadline is reached
→ This approach is widely used in real time s/m.

(III) Multiple Tasks & Multiple Processes:

→ Multiple tasking - refers to the ability to effectively carry out several tasks or processes at once.
→ It enables an embedded s/m to manage multiple tasks Concurrently, as opposed to single-tasking s/m.
→ RTOS (Real time operating s/m) allows us to run several pgm Concurrently which helps build Complex s/m using several pgm that run Concurrently.

Tasks & Processes:-

Tasks:
→ A task is a unit of execut° or unit of work in a s/w appl°.
→ Task execut° in an embd processor is managed by OS.

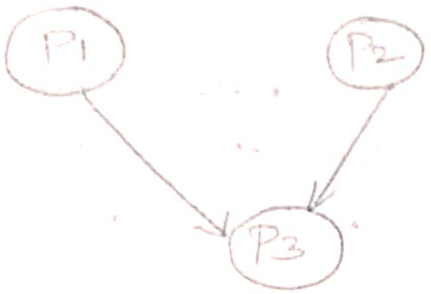


Fig - A task made of 3 subtasks.

→ Fig shows a task that consists of 3 subtasks & arrows in the task graph show data dependencies.

Process:-

→ A process is a single execution of a program.
→ If we run the same program two diff times, then we have 2 diff. processes.
→ Each process has its own state that include both its registers as well as its memory.
→ In some OS, the memory management unit is used to keep each process in separate address space.
→ In lightweight RTOS, the processes run in same address space.

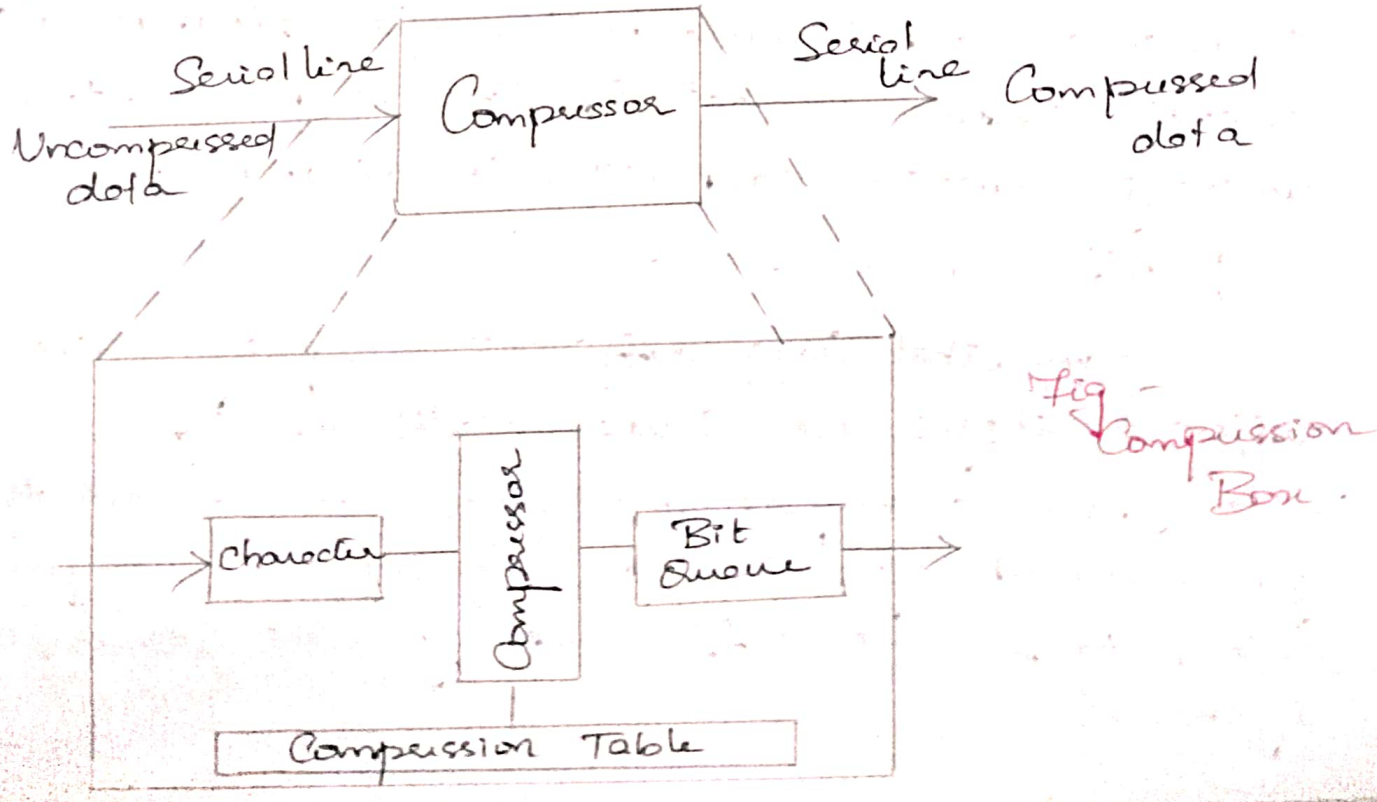


Fig - Compression Box.

'Threads'

→ Processes that share the same address space are often called 'threads'.

→ The i/p to the box is an uncompressed stream of bytes.

→ The box emits a compressed string of bits on the o/p serial line based on a predefined compression table.

(i) Variable Data Rates

→ There is a need to receive & send data at different rates.

→ Eg - The pgm may omit two bits for the 1st byte & then 4 bits for the 2nd byte.

→ This will be reflected in structure of code.

(ii) Asynchronous Input:

→ Asynchronous data means it is 'rcd & received at variable time intervals.

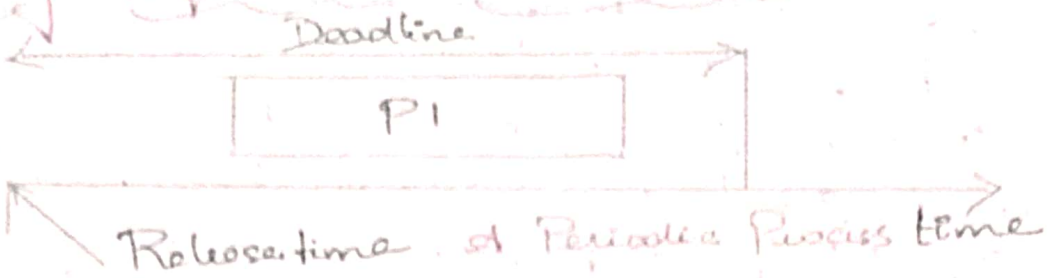
→ Ex → Control panel of compression box may include a compr mode button that disables or enables compr so that i/p text is passed through unchanged when compression is disabled.

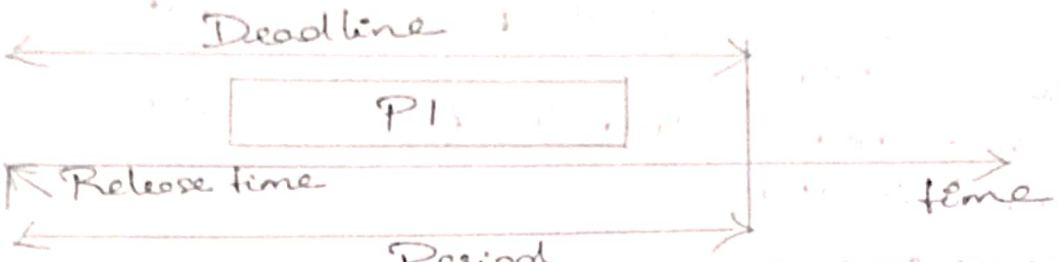
(iv) Multirate systems:-

→ S/m that use different sampling rate at different stages are called Multirate S/m.

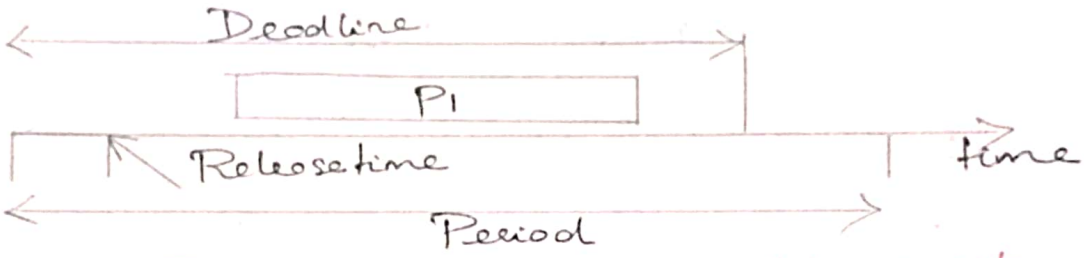
→ These techniques are used to convert the given sampling rate to the desired sampling rate & to provide diff sampling rate through system.

Timing Requirements on Processes:





Periodic Process initiated at start of period



Periodic process released by event.

→ The timing requirements on a set of processes strongly influence the type of scheduling.
→ Scheduling policy define the timing requirements that uses to determine whether schedule is valid.
→ 2 important requirements on processes
1) Initiation time
2) Deadline.

① Initiation Time:
↳ It is the time at which the process goes from the waiting to ready state which is also known as release time.
→ An aperiodic process is the process which is initiated by an event, such as an external data arriving or data computed by another process.

Deadline:

→ A deadline specifies when a computation must be finished.

→ Deadline for an aperiodic process is generally measured from the initiation time.

→ Deadline for a periodic process may in general occur at some time other than end of period.

→ Process rate - Each process executes at its own distinct rate.

→ Fig below illustrates process execution in a sys with 4 CPUs.

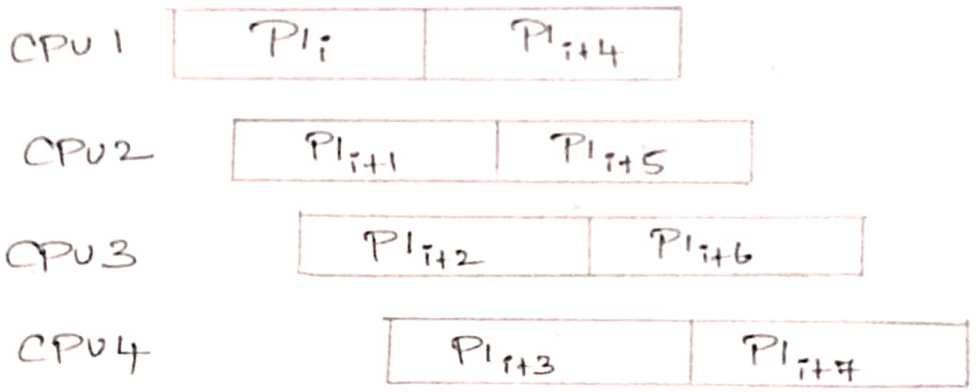


Fig - Sequence of processes with a high initiation rate.

Jitter:-

→ Jitter of a task means it is allowable variation in the completion of task.

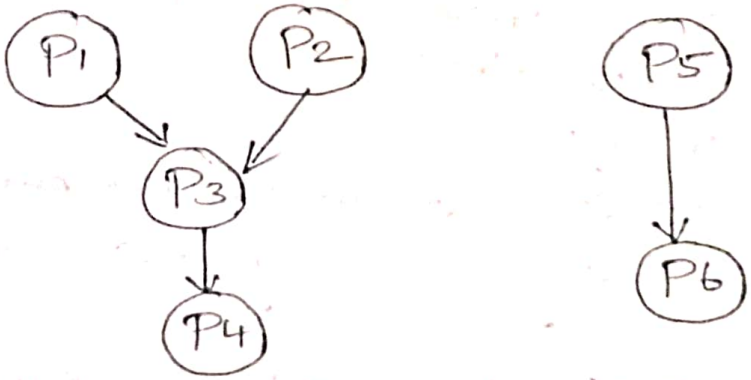


Fig - Data Dependencies among processes.

→ Before a process can become ready, all the processes on which it depends must complete & send their data to it.

Data Dependencies define a partial ordering on process execution.

→ In above case, P1 & P2 can execute in any order but must both complete before P3 & P3 must complete before P4.

CPU Usage Metrics:

→ Basic measure of % of CPU is utilization (U).

→ It is defined as "Ratio of CPU time that is being used for useful computations to the total available CPU time.

$$U = \frac{\text{CPU time for useful Work}}{\text{Total available CPU time}}$$

→ This ratio ranges b/w 0 to 1.

→ Utilization is often expressed as a percentage.

Process state & Scheduling:-

→ The work of choosing the order of running processes is known as Scheduling.

→ There are 3 basic scheduling states

(i) Waiting
(ii) Ready
(iii) Executing.

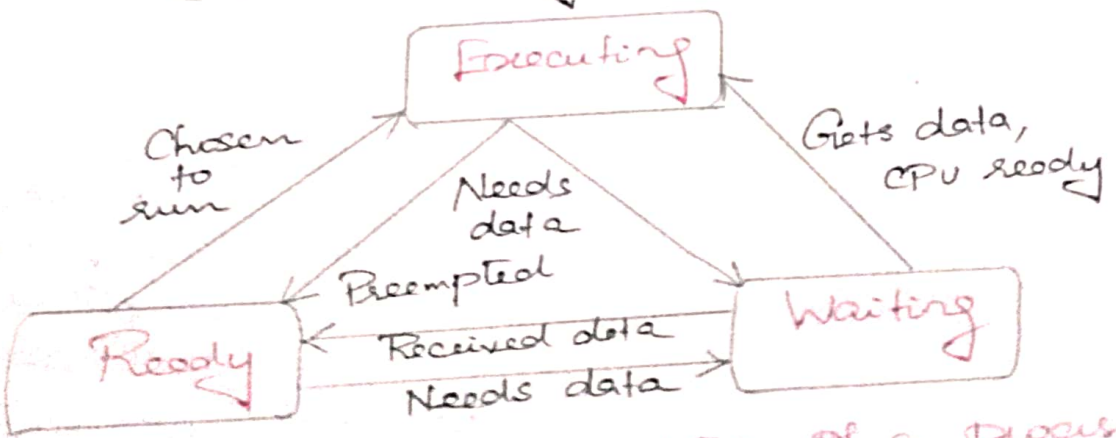


Fig - Scheduling states of a process.

→ A process goes into waiting state when it needs data that it has not yet received or when it has finished all its work for current period.
→ A process goes into ready state when its required data (or) when it enters a new period.
→ A process can go into executing state only when it has all its data & it is ready to run.

Scheduling Policy:
→ It defines how processes are selected for promotion from the ready state to running state.

(V) Preemptive Real time Operating systems:-
→ It executes the processes based upon timing requirements provided by s/m designer.

Two Basic Concepts:
→ Preemption as an alternative to c fn° calls as a way to control the execution.
→ Priority based scheduling as a way for the progremmer to control the order in which processes run.

Preemption:-

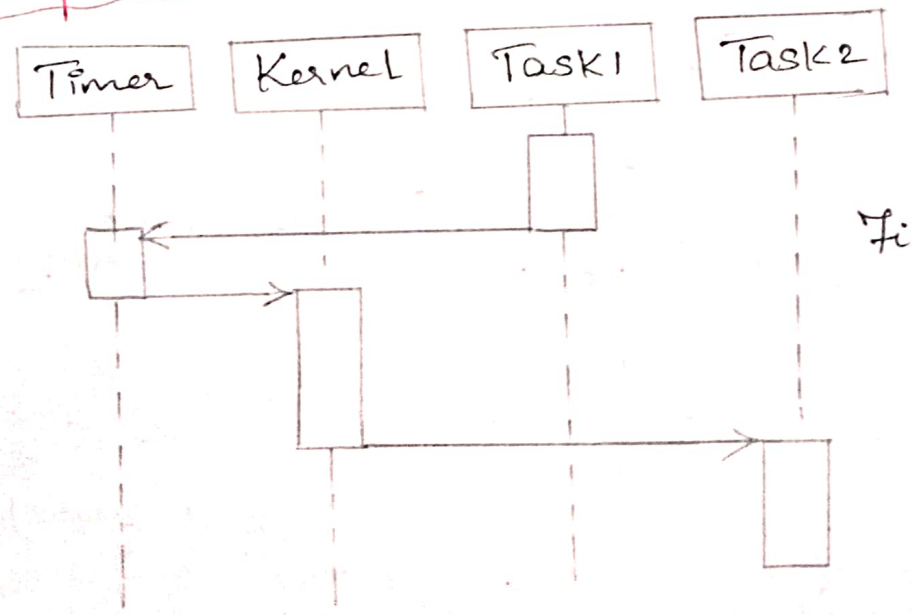


Fig - Sequence Diagram for preemptive execution.

→ Jumping from one subroutine to another at any point is allowed in the program.
→ Together with the timer, allow us to move b/w fun^{n} whenever necessary based on s/m timing constraints

Kernel:-
→ It is a part of OS that determines what process is running which is activated periodically by timer.

Time Quantum:-
→ The length of the timer period is known as time quantum which is smallest ↑ in that we can control CPU actively.
→ The Kernel determines what process will run next & causes that process to run.
→ On the next timer interrupt, the Kernel may pick the same process or another process to run.

2) Context switching:
→ Set of Registers that defines a process is known as its context & switching from one process register set to another is known as context switching.

3) Process Priorities:
→ Kernel can simply look at the processes & their priorities, & then select the highest priority process that is ready to run.

Processes & Context:
→ A process is known as a task in freeRTOS.org
→ The Diag shows the appl^{n} tasks, the H/w timer & all fun^{s} in Kernel that are involved in Context switch are

Fig - Seq Diag for preemptive context switch.

- vPreemptive Tick () → Called Timer ticks
- SIG_OUTPUT_COMPARE1A → Responds to timer Interrupt & uses portSAVE_CONTEXT() to swap out current task Context
- vTaskIncrementTick() → updates time & VTaskSwitchContext chooses new task
- portRESTORE_CONTEXT() → Swaps new Context.

Processes :- Object Oriented Design :- UML

Active Objects

→ UML often refers to processes as active objects (ie) Objects that have independent threads of Control.

→ The class that defines an active object is known as active class.

processclass1
myAttributes
myOperations ()
Signals Start Resume

Fig - Active class in UML.

→ Fig shows an example of UML active class.
→ It has all normal characteristics of class including name, attributes & operations.
→ It also provides a set of signals that can be used to communicate with the process.
→ A signal is an object that is passed b/w processes for asynchronous Comm^{n}.

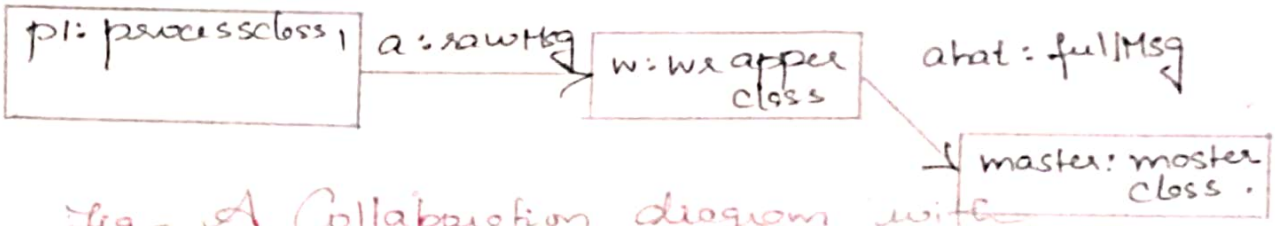


Fig - A Collaboration diagram with active & normal objects.

VI) Priority Based Scheduling:-

→ A task can be stopped or suspended in order to allow a task of higher priority to run.
→ In case of two tasks of same high priority each is given an equal time slice.

(i) Round-Robin Scheduling:

→ Here all processes are kept on a list & scheduled one after the other.
→ It provides a chance for all processes to execute but it does not guarantee the completion time of any task.
→ As no of processes ↑, then the response time of all processes get ↑.

(ii) Process Priorities.

→ The next executing process in RTOS is choosed based on process priorities.

→ Each process is assigned a priority which is an integer valued no.

Process	Priority	Execution time
P1	1	10
P2	2	20
P3	3	30

Fig - Process Priority table

Rate Monotonic Scheduling (RMS):-

→ The priority is decided according to the cycle time of the processes that are involved.
→ The process with small job duration has assigned a highest priority.
→ RMS is a static scheduling policy because it assigns fixed priorities to processes.
→ RMS uses RMA (Rate Monotonic Analysis) which is summarized below.

- All processes run periodically on single CPU
- Context Switching time is ignored
- No Data dependencies b/w processes
- Execute time for process is constant
- All deadlines are at ends of their periods.

Response time:
→ It is the time at which process finishes.

Critical Instant:
→ It is defined as instant during execute at which task has largest response time.

CPU Utilization:

→ It does not allow system to use 100% of available CPU cycles.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$

Where

$n \rightarrow$ No. of processes in process set
$C_i \rightarrow$ Computation time of process
$T_i \rightarrow$ Time period for process to run
$U \rightarrow$ Processor Utilization.

For 'n' tasks, max. processor utiliz^n is

$$\boxed{U = n\left(2^{1/n} - 1\right)}$$

Advantage:

→ Easy to Implement
→ Can meet the deadlines
→ Consist of Calculated copy of time periods.

Drawbacks:

→ Very Difficult to support aperiodic tasks under RMA.
→ RMA is not optimal when the task period & deadline differ.

Earliest Deadline first (EDF) Scheduling:

→ It is an optimal dynamic priority scheduling algorithm used in real time s/m.
→ EDF uses priorities for scheduling which assigns priorities to the task according to absolute deadline.
→ The task whose deadline is closest gets

RMS	EDF
→ Achieves low CPU Utilization	→ Higher CPU Utilization
→ static priority Scheduling	→ Dynamic priority Scheduling.
→ Not Expensive to use in practice	→ Expensive to use in practice.
→ Shortest period process gets highest priority	→ Process closest to its deadline has highest priority.

(vii) Interprocess Communication s/m :-

→ Process Can send a Comm^n in one of 2 ways

1) Blocking
2) Non Blocking.

→ After sending in a blocking Comm^n, the process goes into waiting state until it receives a response.

→ Non Blocking Com^n allows the process to Continue execut^n after sending the Comm^n.

→ There are 2 modes through which the processes Can Communicate with each other.

(i) Shared Memory
(ii) Message passing.

① Shared Memory Comm^n :-

→ It is bus based s/m.

→ Here CPU & I/O device is Communicated through shared memory loc^n.

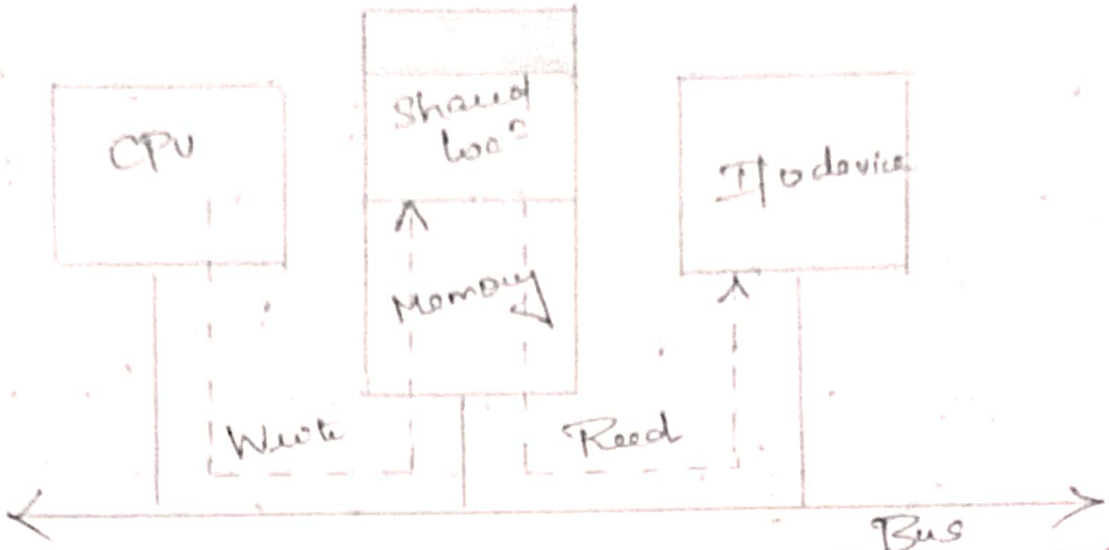


Fig - Shared memory Comm^n implemented on Bus.

→ If CPU wants to send data to the device, it first writes to shared loc^n.
→ I/O device then reads the data from that loc^n.

Message Passing :-

→ In the message passing mode, processes interact with each other through messages with assistance from the underlying OS.

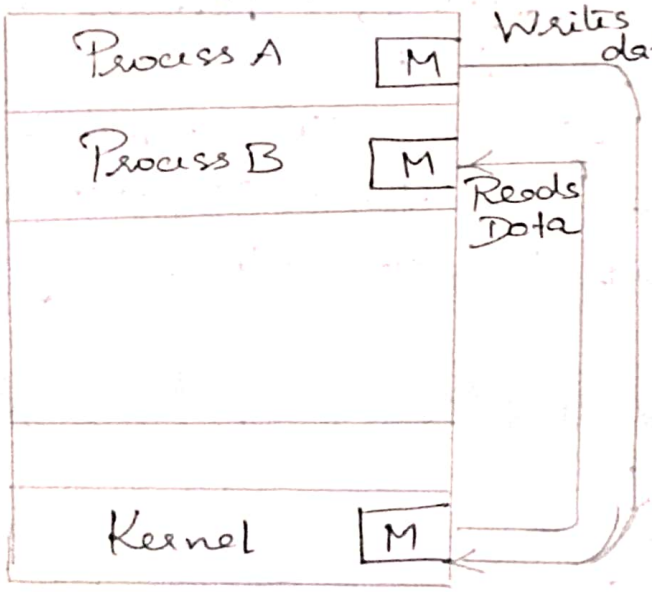


Fig - Message passing Comm^n.

→ Two processes A & B are communicating with each other through message passing.

Queues:

→ A Queue is a common form of message passing.

→ The Queue uses a FIFO discipline & holds records that represent the messages.

→ FreeRTOS.org s/m provides a set a queue fu² which allows queues to be created & deleted so that s/m may have as many queues as necessary.

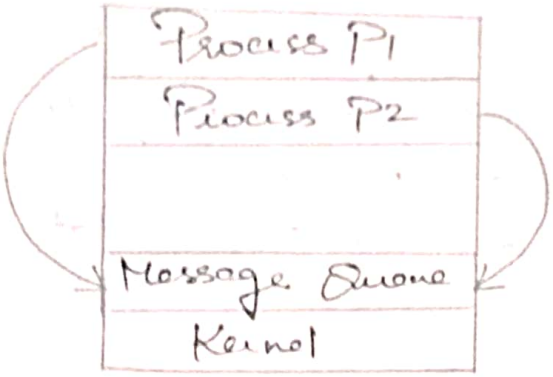


Fig - Message Queue.

Signals:

→ Signal - does not pass data.

→ Signal - Notif^c to a process indicating the occurrence of event.

→ Signal is also called s/w interrupt & it is not predictable to know its occurance.

→ Hence it is also called asynchronous event.

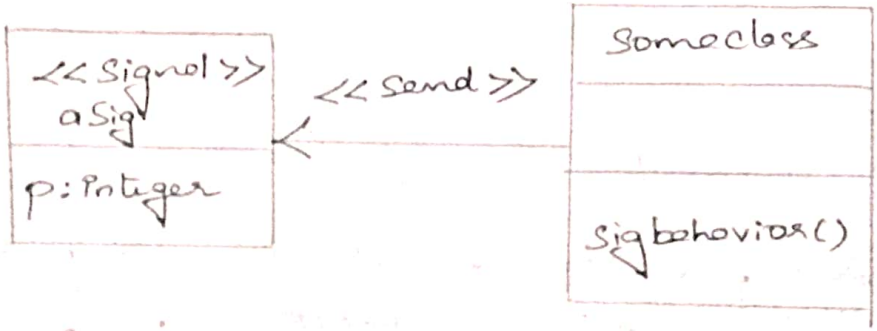


Fig - Use of UML Signal.

VIII) Distributed Embedded Systems:

→ In Distributed embedded s/m, several PE (Processing Elements) are connected by a n/w that allows to communicate.

→ PE may includes DSP, CPU or µc.

(i) Network Abstractions: OSI Model:

Application	End-use interface
Presentation	Data format
Session	Appl^{c} dialog Control
Transport	Connections
Network	End-to-end service
Data link	Reliable data transport
Physical	Mechanical, electrical.

Fig - OSI Model layer.

(i) Physical layer:-

→ This layer defines basic properties of interface b/w s/m which includes physical conn^{n}, electrical properties, basic fu^{c} of electrical & physical components.

(ii) Data link layer:-

→ The primary purpose of this layer is error detection & control.

(iii) Network layer.

→ This layer defines end-to-end data tr^{n}

(iv) Transport layer:

→ This layer defines Connection-Oriented services which ensures that data are delivered in proper order & without errors across multiple links.

(v) Session layer:

→ It provides mechanism for controlling the interaction of end-user services across a n/w such as data grouping & check pointing.

(vi) Presentation layer:

→ This layer defines data exchange formats & provides transfr utilities to an appln pgm.

(vii) Applr layer:

→ It provides the applr interface b/w n/w & end-user programs.

CAN Bus (Controller Area N/w):

→ It is a vehicle bus std designed to allow µc & devices to communicate with each other applr without host Computer.

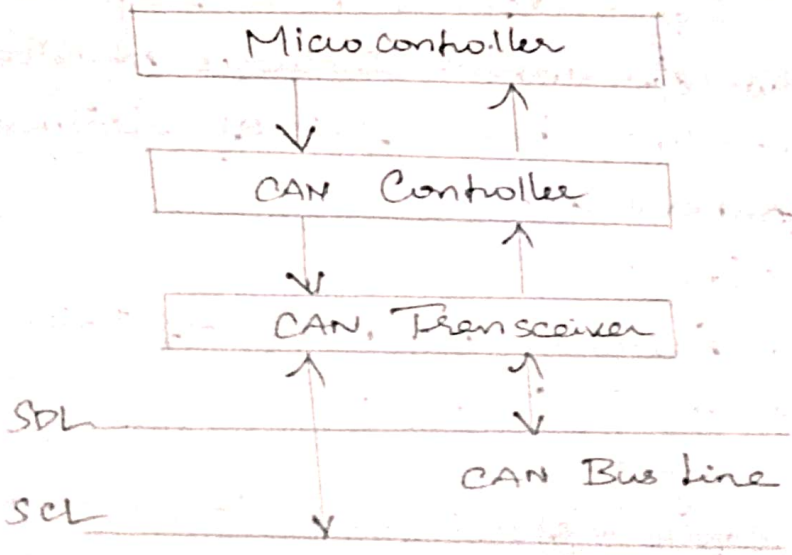


fig - CAN Interface in Embd sfm.

→ CAN uses bit-Serial Commⁿ & runs at (12)
rates of 1Mb/s over a twisted pair Conⁿ of 40m.

① Physical layer:

→ In CAN terminology, a logical 1 on the bus is called recessive & logical 0 is dominant.

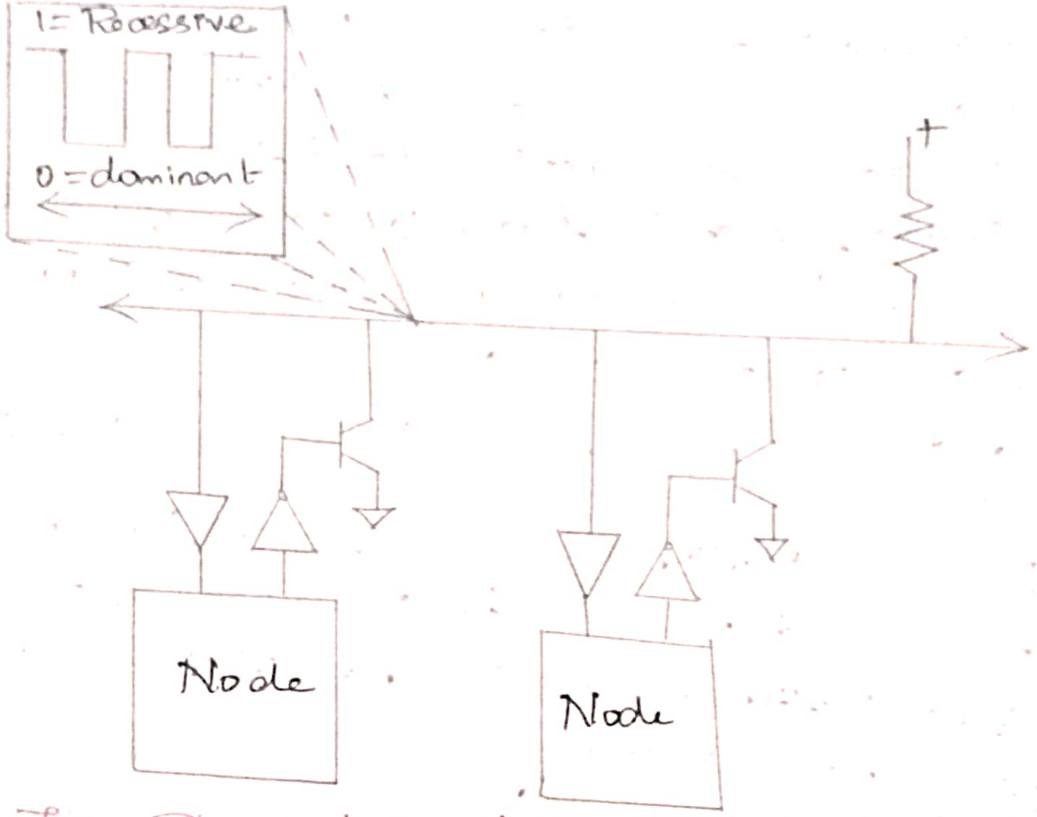


Fig. Physical & electrical Orgⁿ of CAN Bus.

→ When all nodes are TX'g 1s, bus is said to be in recessive state, when a node transmits a 0, the bus is in dominant state.

→ Data are sent on the n/w in packets known as data frames.

→ CAN is a synchronous bus bcoz all TX's must send at the same time for bus arbitration to work.

② Data frame:

→ Starts with 1 & ends with a string of seven zeros.

→ The 1st field in the packet contains the packet dest° address which is known as arbitration field.

→ The dest° identifier is 11 bits long.

→ The trailing Remote Transmission Request (RTR) bit is set to 0 if the data frame is used to request data from device specified by identifier.

→ When RTR = 1, the packet is used to write data to dest° identifier.

→ Data field is from 0 to 64 bytes.

→ Cyclic Redundancy check (CRC) is used for error detection.

IX) MPSOCs & Shared Memory Multiprocessors

MPSoc → Multiprocessor System on chip means System-on-chip with two or more CPU Cores.

→ The Multiprocessor is a parallel processor with single shared memory.

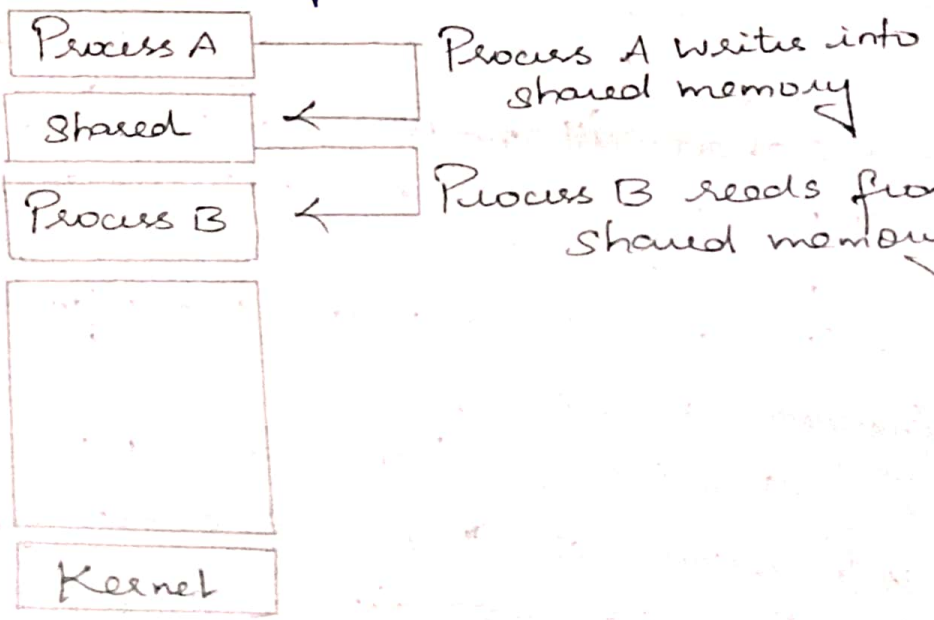


fig- Shared Memory

Shared Memory:

→ It is a memory that shares b/w 2 or more processes.

→ Each process has its own address space.

→ If any process wants to communicate with some info from its own address space to other process then it is only possible with IPC.

Accelerators

→ Accelerator is a PE for embedded multi processor which can provide large performance ↑ for appl^n with Computational Kernels that spend a great deal of time in small section of code.

→ Accelerators also provide Critical speed ups for low latency I/o fu^n.

→ CPU is often called host & it communicates to the accelerator through data & control registers in accelerator.

→ CPU & accelerator may also communicate via shared memory & use synchronizⁿ mechanism to ensure that they do not destroy each other data.

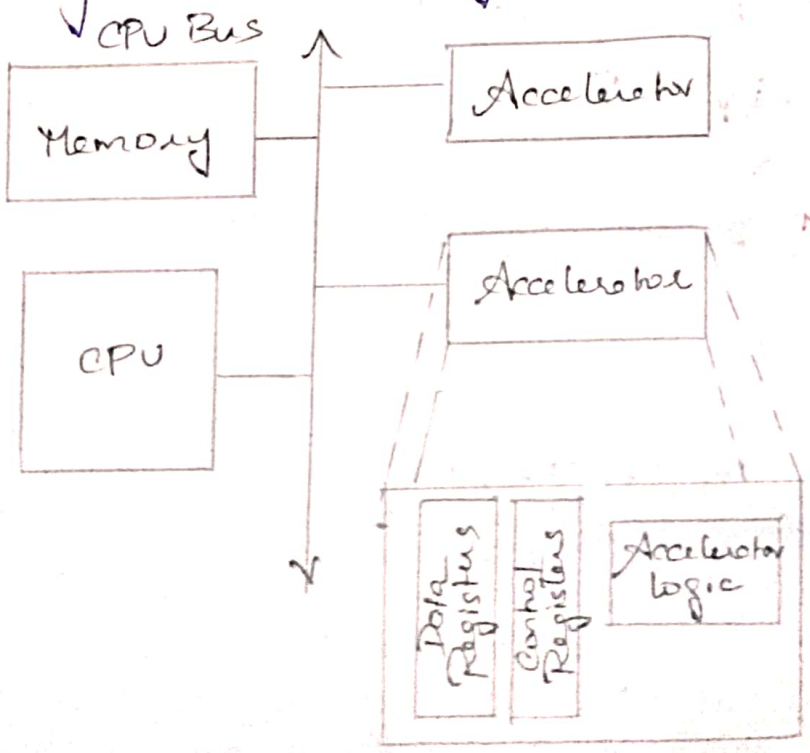


Fig- CPU accelerators in a Sfm.

→ Accelerator interacts with CPU through programming model interface which does not execute itself.

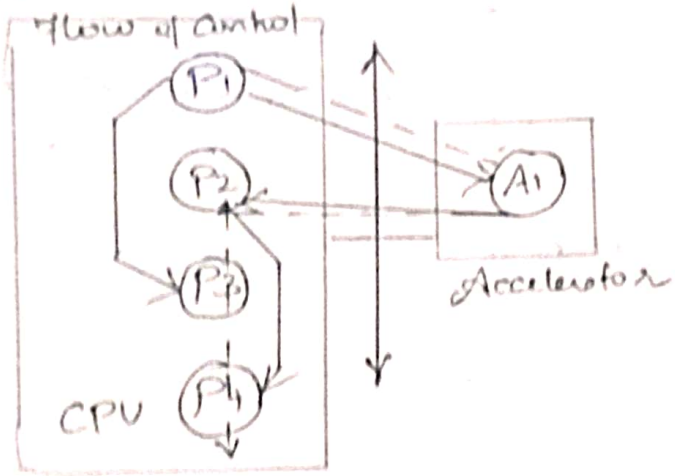
→ Field Programmable gate Arrays (FPGA) provide useful platform for Custom Accelerators.

→ FPGA has a fabric with both programmable logic gate & programmable interconnect that can be configured to implement a specific fun°.

Accelerator Performance Analysis

→ Speedup factor depends on whether the app is single threaded or multithreaded.

(i.e.) Whether CPU is mostly idle while the accelerator runs in single threaded case or in multi-threaded case, the CPU can do useful work in parallel with accelerator.



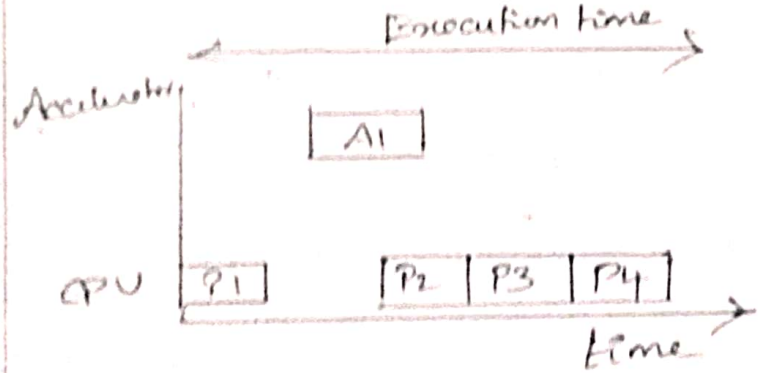


Fig. Single-Threaded.

→ Fig shows that data dependencies allow P2 & P3 to run independently on CPU, but P2 relies on the results of the AI process that can be implemented by accelerator.

→ In Single Threaded case, CPU blocks to wait for the accelerator to return the results of its Computation.

→ As a result, it does not matter whether P2 or P3 runs next on CPU.

→ Total execution time is expressed,

$$t_{accel} = t_{in} + t_x + t_{out}$$

t_x → Execut^n time of the accelerator

t_{in} & t_{out} → times required for reading & writing the required variables.

→ total speedup (S) is expressed as,

$$S = n(t_{cpu} - t_{accel})$$
$$= n[t_{cpu} - (t_{in} + t_x + t_{out})]$$

Scheduling & Allocation:

→ Schedule oper^n in time which includes Com^n on n/w & Computations on Processing Elements (PE)

→ The Scheduling of oper^n on PE & Com^n b/w PEs are linked.

→ If one PE finishes its Computations too late, it may interfere with another Comm^n on the n/w as it fails to send its result to the PE that needs it.

→ We must allocate Comput^n to the PEs & determines what Com^n are required.

Design Example - Audio Player:

→ Audio players are often called MP3 players.

→ MP3 player performs 3 basic fun°

- → Audio storage
- → Audio decompression
- → User Interface.

→ In MP3 players, the following 3 layers std define audio compression.

① Layer 1 (MP1)
- It uses lossless Compress° of subbands

② Layer 2 (MP2)
- It uses more advanced masking model.

③ Layer 3 (MP3)
- It performs additional processing to provide lower bit rates.

→ The various layers that support several diff. i/p sampling rates, o/p bit rates & modes such as mono, stereo etc.

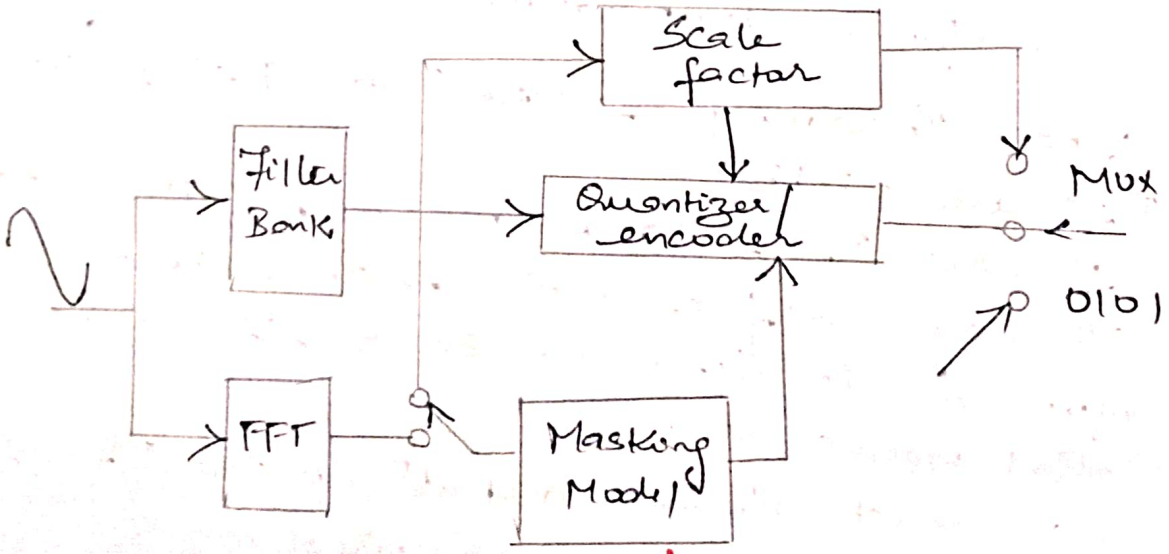


Fig - MPEG layer 1 encoder.

→ Fig above gives a block Diag of layer 1 encoder.
→ The main processing path includes the filter bank & quantizer / encoder.
→ Filter bank splits the signal into set of 32 subbands that are equally spaced in freq domain & together covering entire freq range of audio.
→ The masking model selects the scale factors which is obtained by separate (FFT).
→ The Mux at o/p of encoder passes along all required data.

Header	CRC	Bit allocation	Scale factors	Subband Samples	Aux data

Fig. MPEG layer-1 data frame format.

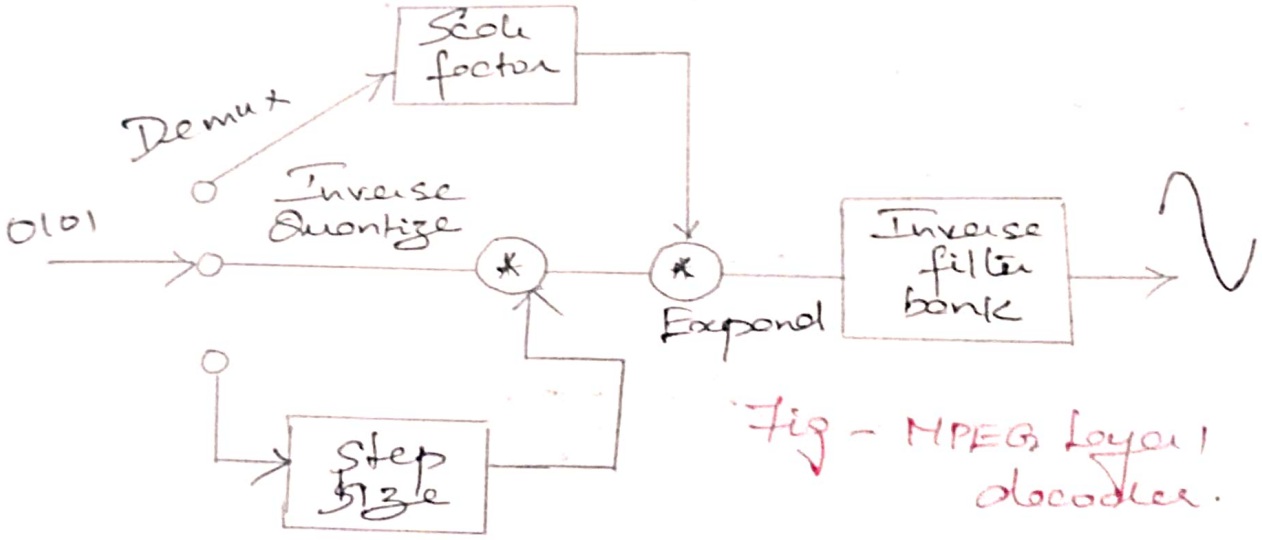


Fig - MPEG Layer 1 decoder.

→ MPEG data streams are divided into frame which carries the basic MPEG data, error correction codes & additional inf?

→ After disassembling the data frame, the data are unscaled & inverse quantized to produce sample streams for subband.

Requirements:-

Name	Audio Player.
Purpose	Play audio from files
Inputs	Flash memory socket, on/off
Outputs	speaker
Functions	Display list of files in flash memory, Select file to play.
Performance	Sufficient to play audio file's at required rate.

Specification:-

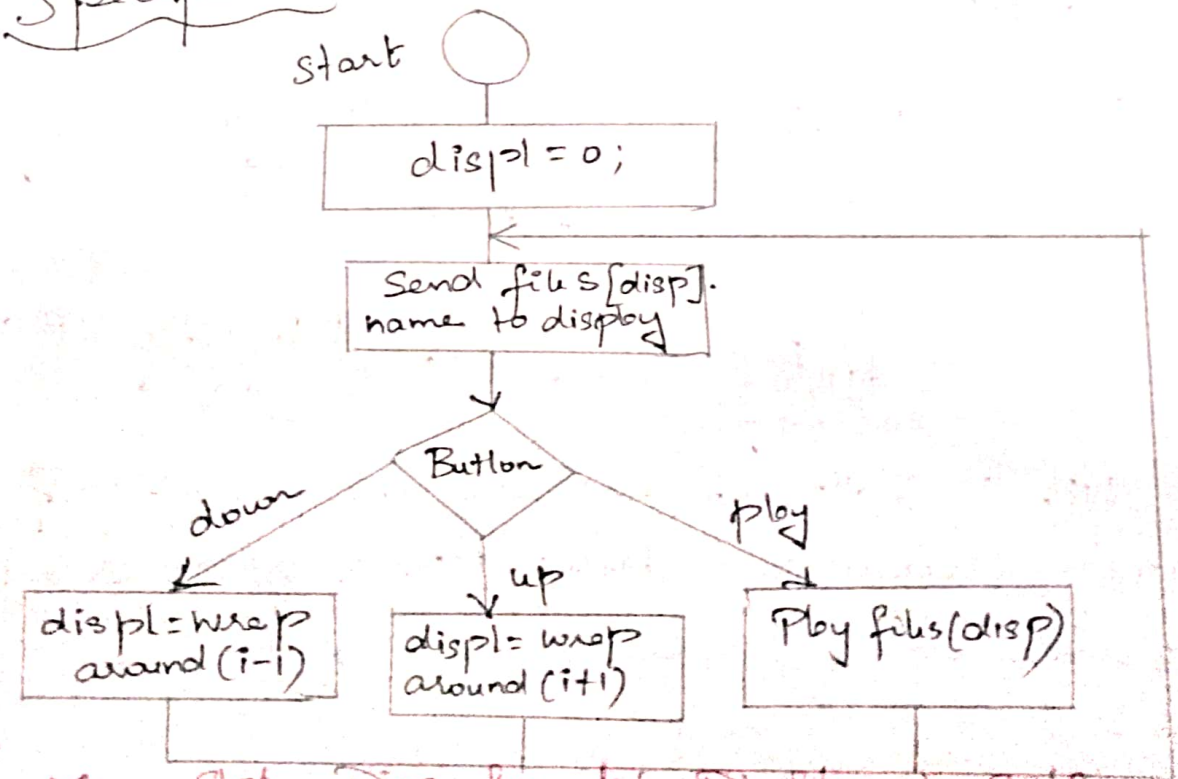


Fig. State Diag for file Display & Select

→ Specifⁿ assumes that all files are in the root directory & all files are playable audio.
→ This status diag refers to sending the samples to the audio s/m because playback & reading next data frame must be overlapped to ensure Continuous operⁿ.

System Architecture:

→ The audio Controller includes 2 processors.
(i) A 32-bit RISC processor is used to perform s/m Control & audio decoding.
(ii) A 16 bit DSP is used to perform audio effects such as equaliz?.

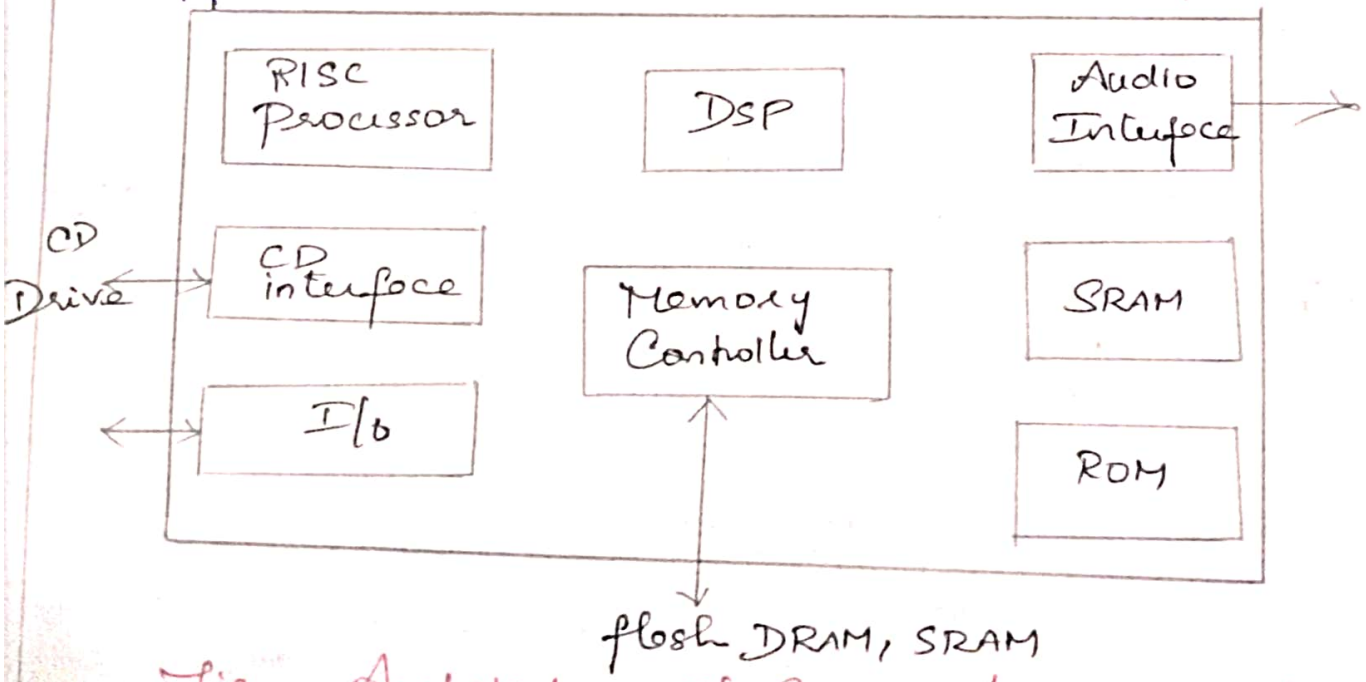


Fig - Architecture of Cirrus Audio proc. for CD/MP3 player.

→ Fig shows Cirrus chip uses 2 processors: a Risc & DSP.
→ Memory Controller Can be interfaced to several diff types of memory.
→ flash memory → Used for data or Code Storage

DRAM — Used as a buffer to handle temporary disrupt° of CD data stream

→ O/p of audio interface unit is in audio formats that can be used by A/D Converters

→ General purpose I/O pins can be used to decode buttons, run displays etc.

Engine Control Unit

→ It Control the opera° of fuel injected engine based on several measurements taken from running engine.

Theory of opera° & Requirements :-

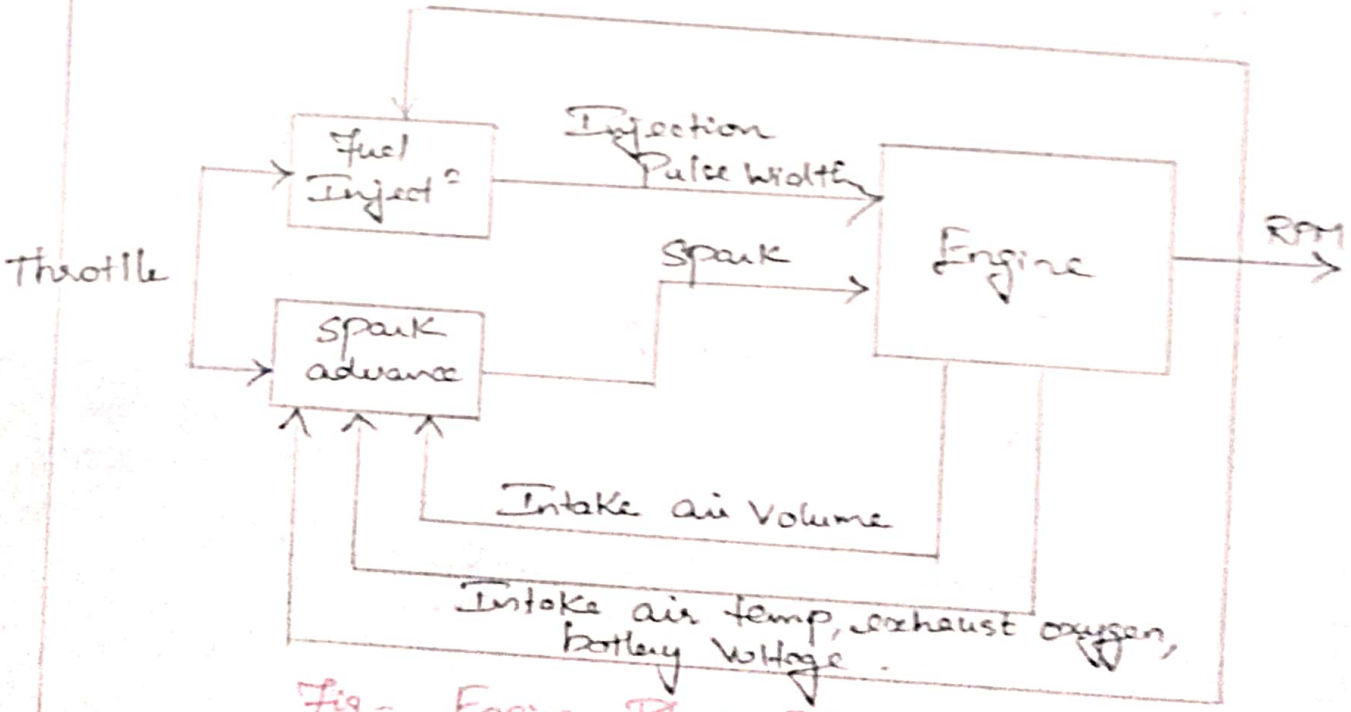


Fig - Engine Block Diagram

→ The Throttle is the Command i/p & engine measure Throttle, RPM, intake air Volume & other variables.

→ Engine Controller Computes injector pulse width & spark.

→ This design does not Compute all o/p required by real engine.

Requirements:

Name	ECU
Purpose	Engine Controller for fuel-injected engine
Inputs	Throttle, RPM, intake air Volume
Outputs	Injector pulse width, spark advance angle
Functions	RPM, intake air volume, intake manifold pressure, Compute Injector pulse width
Performance	Injector pulse updated at 2ms period, spark advance angle updated at 1ms period.
Power	Powered by engine generator

Specification:

→ Engine Controller must deals many processes

→ ∇NE & ∇T represents the change in RPM & throttle position.

Signal	Variable name	In/out	Update period (ms)
Throttle	T	Input	2
RPM	NE	Input	2
Intake air Volume	VS	Input	25
Injector pulse width	PW	Output	2
Intake air temp	THA	Input	500

→ Controller Computes 2 o/p signals
* Injectors Pulse Width
* Spark advance angle (S)

$$PW = \frac{2.5}{2NE} \times VS \times \frac{1}{10 - K_1 \Delta T}$$

$$S = K_2 \times \Delta NE - K_3 VS.$$

→ Controller applies Corrections to these initial values.
(i) As the intake air temperature ↑ during engine warm-up, then Controller reduces the inj° dur°.
(ii) As throttle opens, the Controller temporarily ↑ the inj° frequency.

System Architecture:

→ Two major process are pulse-width & advance angle.
→ The Control parameters are Computed for spark plugs & injectors.
→ The Control parameters rely on changes in some of i/p signals.
→ Each change must be updated at variable sampling rate.
→ Fig below shows the state diag for throttle sensing, which saves both the Current value & change in value of throttle.

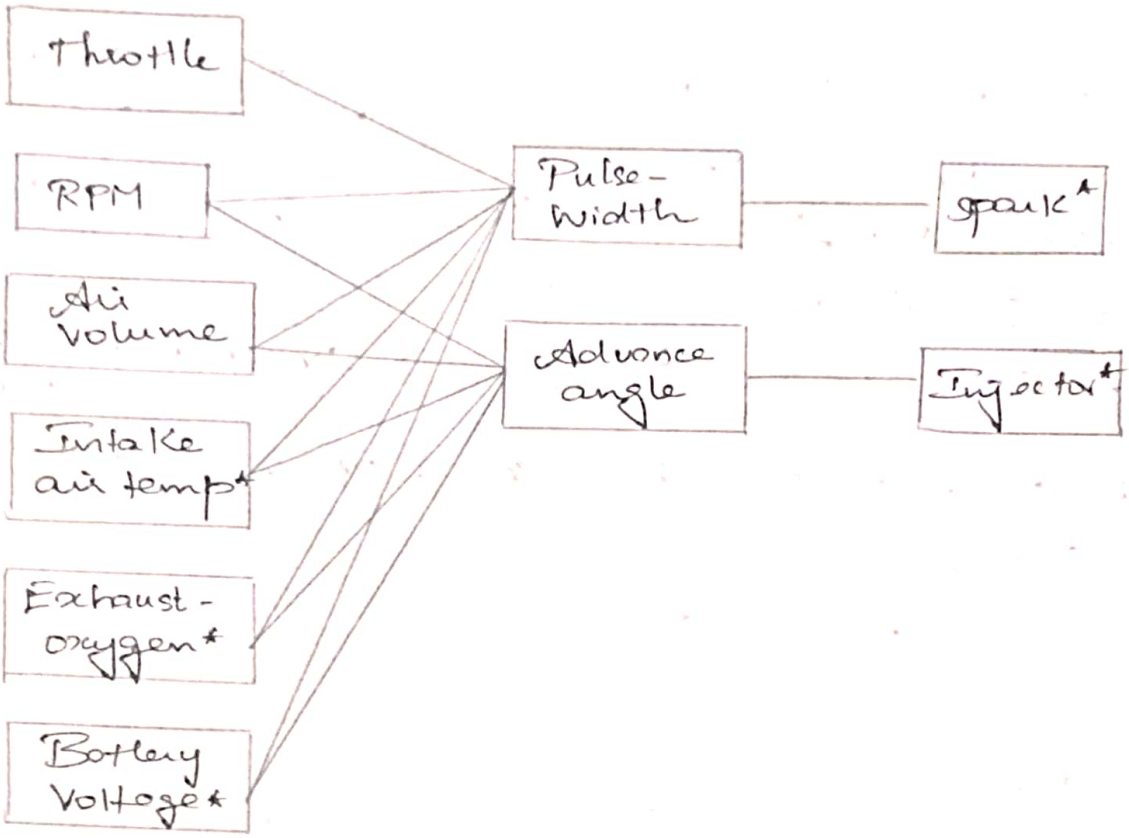


Fig - Class Diag for Engine Controller

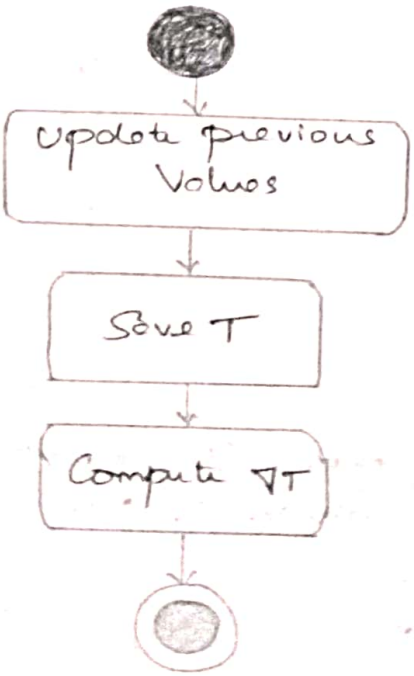


Fig - State Diag for Throttle position Sensing

Video Accelerator:

→ A video Accelerator significantly speeds up the updating of images on a screen which makes CPU free to take care of other tasks.

→ It is mainly used

(i) To increase the overall capabilities of video graphics.

(ii) To provides critical speed ups for low-latency I/O fun.

Video Compression:

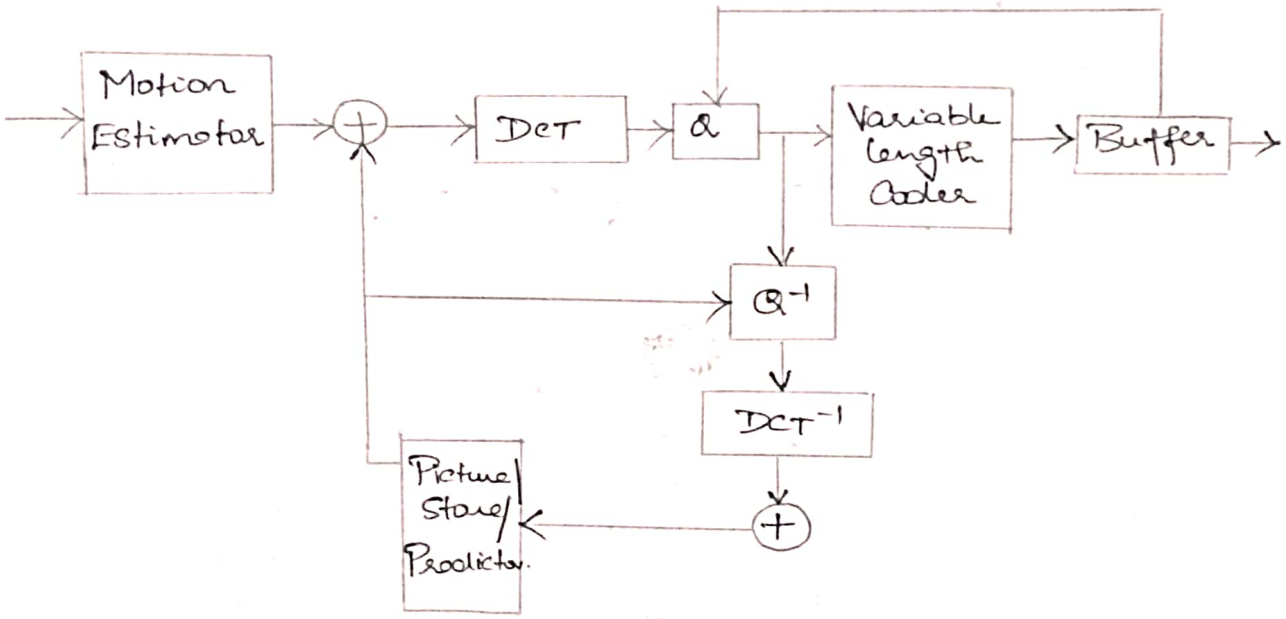


Fig - Block Diag of MPEG-2 Compression Alg.

→ Fig shows block Diagram for MPEG-2 video Compression Alg which is the basis for US HDTV broadcasting.

→ DCT (Discrete Cosine Transform) used in JPEG also plays a key role in MPEG-2.

→ MPEG-2 encoder also uses a feedback loop to further improve image Quality.

(1) Motion-Based Coding:

→ Some frames are sent as modified forms of other frames using a technique known as block motion estimation.

Block Motion Estimation:

→ A block matching Algorithm is a way of locating the matching macro blocks in a sequence of digital video frames for purpose of motion estimation.

→ During encoding, the frame is divided into macro blocks which is identified from one frame in other frames using Correlation.

→ The frame can then be encoded using motion vector that describes the motion of macroblock from one frame to another without the all of pixels

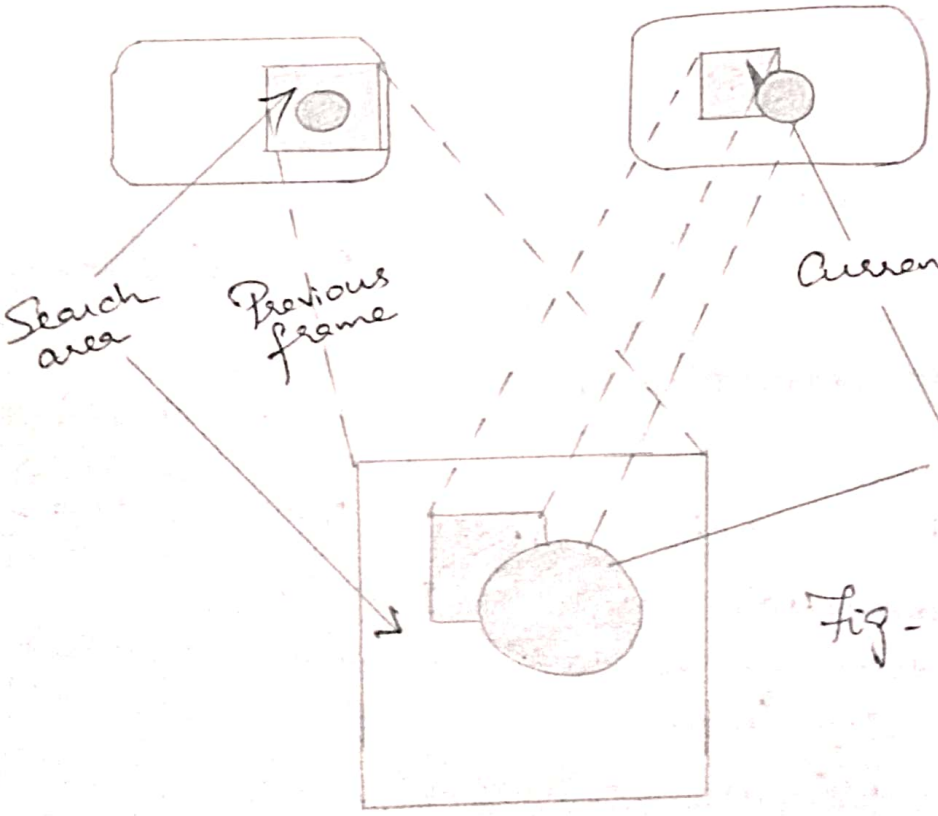


Fig- Block Motion Estimation.

Requirements.

Name	Block motion Estimator.
Purpose	Perform block motion est° within PC s/m
Inputs	Macroblocks & Search areas
Outputs	Motion Vectors
Functions	Compute motion vectors using full search alg.
Power	Powered by PC power supply
Size & weight	Packaged as PCI Card for PC.

Specification:

→ Defines some classes that describe basic data types in a s/m

- → Motion Vector
- → Macroblock
- → Search area.

Motion-Vector
x, y

Macroblock
pixels[]

Search-area
pixels[].

fig. classes describing basic data types

Architecture:

→ Accelerator will be implemented in FPGA on a Card Connected to PC's PCI slot.

→ Architecture for motion estimation accelerator is shown in fig below.

→ Machine has two memories. a) Macroblock & b) Search Memories.

→ FI has 16 Processing Elements (PEs) that is used to perform difference Calculs on pair of pixels.

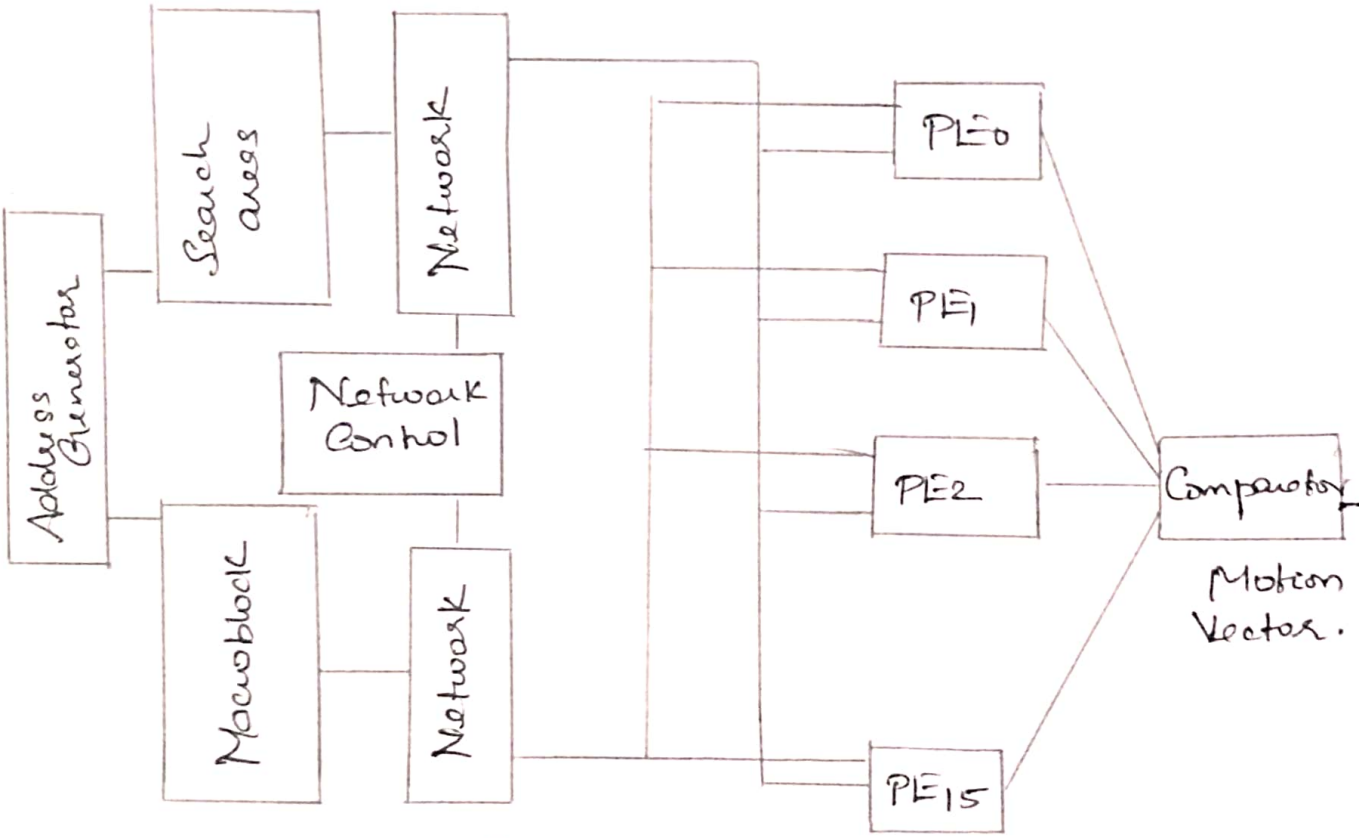


Fig - Architecture for Motion estim^n accelerator.

→ The Comparator sums them up & selects the best value to find motion Vector.

→ This architecture can be used to implement algorithms other than full search by changing the address gen^r & Control.

Unit-IV

IoT Architecture & Protocols.

Internet of Things - Physical Design - Logical Design - IoT Enabling Technologies - Domain Specific IoTs - IoT & M2M - IoT s/w management with NETCONF-YANG - IoT platform Design - Methodology - IoT Reference Model - Domain Model - Comm° Model - IoT Reference Architecture - IoT Protocols - MQTT, XMPP, Modbus, CANBUS & BACNet.

1) Internet of Things :-

→ Internet of Things (IoT) describes the n/w of physical objects which are considered as things that are embedded with sensors, software & other technologies for the purpose of connecting & exchanging data with other devices & s/m over Internet.

Applications :-

- → Homes
- → Cities
- → Environment
- → Energy s/m
- → Retail s/m
- → Agriculture
- → Industrial
- → Health & life style

Characteristics:

- → Dynamic & Self-Adapting
- → Self-Configuring
- → Unique Identity
- → Integrated into Info° n/w.

II) Physical Design of IoT:

→ IoT devices which have unique identities & Can perform remote sensing, actuating & monitoring Capabilities.

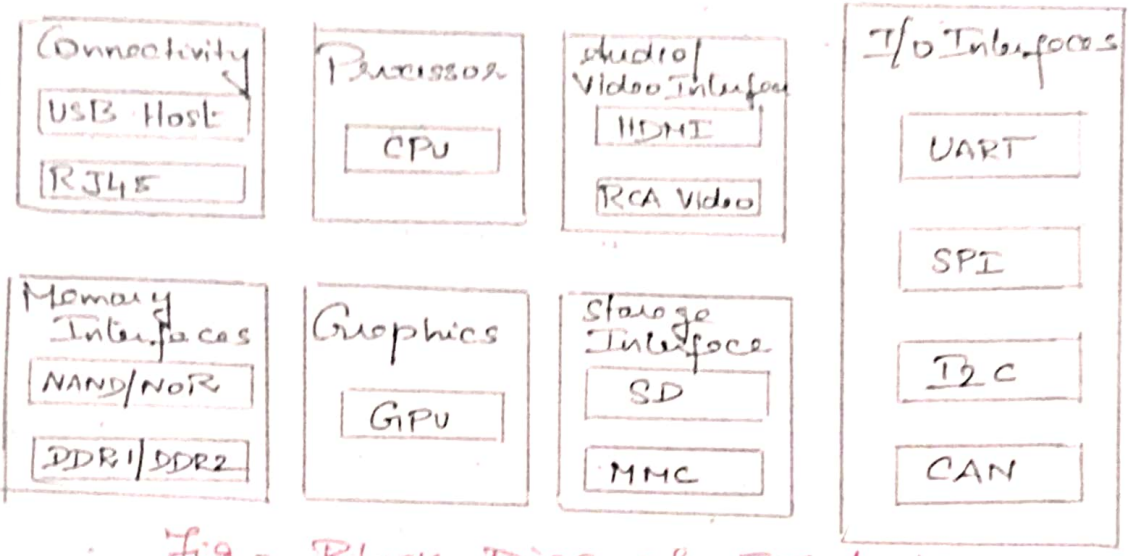


Fig - Block Diag of IoT device.

→ IoT device which may Consists of Several interfaces for Conn^s to other devices

(i) I/o interface for sensors
(ii) Interfaces for Internet Connectivity
(iii) Memory & Storage interfaces
(iv) Audio/video interfaces.

IoT Protocols:

(i) Link Layer:

→ Data link layer determine that how the data is physically sent over n/w physical layer.

→ Host on the same link exchange data packets over the link layer.

802.3 Ethernet:

→ IEEE 802.3 — 10BASE5 Ethernet (use Coaxial Cable)

(ii) IEEE 802.11: Wi-fi

→ 802.11 a – operates 5GHz band.
802.11 b & g – 2.4 GHz band
802.11 n – 2.4/5 GHz bands
802.11 ad – 60GHz band.

(iii) 802.16: WiMax.

→ Provides data rates of 100 Mbit/s for Mobile Station.

(ii) Network layer.

→ It is responsible for sending IP datagrams from source n/w to dest n/w.
→ This layer performs host addressing & packet routing.
→ IPv4 – use 32 bit address
→ IPv6 – use 128 bit address.

(iii) Transport layer:

→ The transport layer protocol provides end-to-end message transfer capability independent of underlying n/w.
→ The fn of transport layer are
(i) Error Control
(ii) Segmentation
(iii) Flow Control
(iv) Congestion Control.

(iv) Application layer:-

→ It defines how the appln interface with the lower layer protocols to send the data over the n/w.

→ HTTP (HyperText Transfer Protocol)
→ WebSocket
→ MQTT (Message Queue Telemetry Transport)
→ DDS (Data Distrc Service)
→ XMPP (Extensible Messaging & Presence Protocol)

(iii) Logical Design of IoT.

→ Logical Design of IoT s/m refers to an abstract represent^n of entities & process without going into low level specif^n of implement^n.

IoT Functional Blocks:-

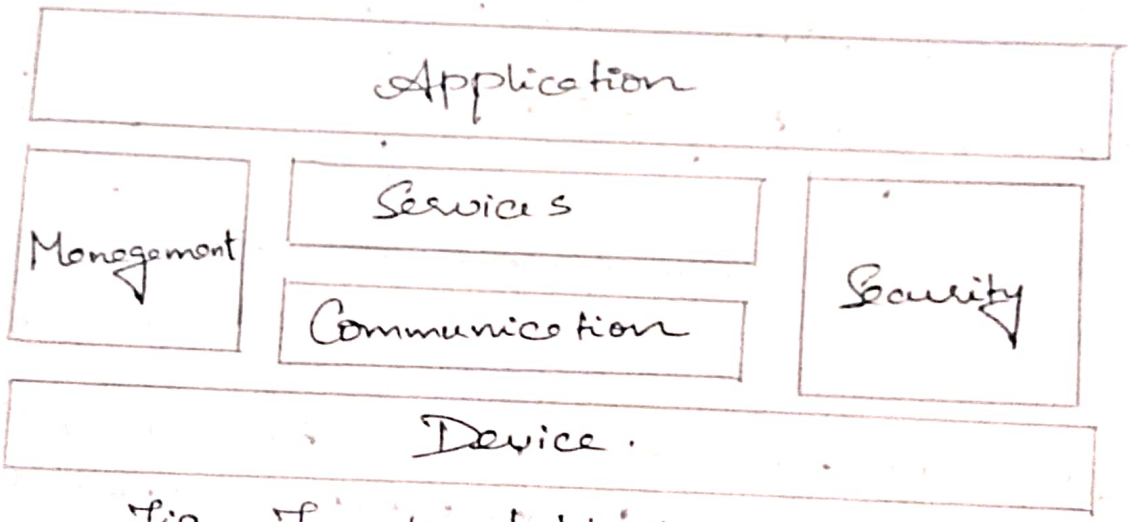


Fig- Functional blocks of IoT.

→ The functional blocks are described as follows

(i) Device:

→ An IoT s/m comprises of devices that provide sensing, actuation, monitoring & control fu^n.

(ii) Communication:-

→ Handles Comm^n for IoT s/m.

(iii) Services:

→ Services for device monitoring, device control services, data publishing services

(iv) Management:
→ These fun^{s} blocks provide various fun^{s} to govern IoT s/m.

(v) Security:
→ Provide fun^{s} such as authentication, authoriz^{n}, message & Content integrity & data Security.

IoT Communication Models:

(i) Request-Response:
→ Client sends request & Server responds.
→ When the Server receives a request, it decides how to respond, fetches the data, retrieves resource repres^{n}, prepares the response & then sends the response to the client.
→ It is a stateless Comm^{n} model & each request-response pair is independent of others.

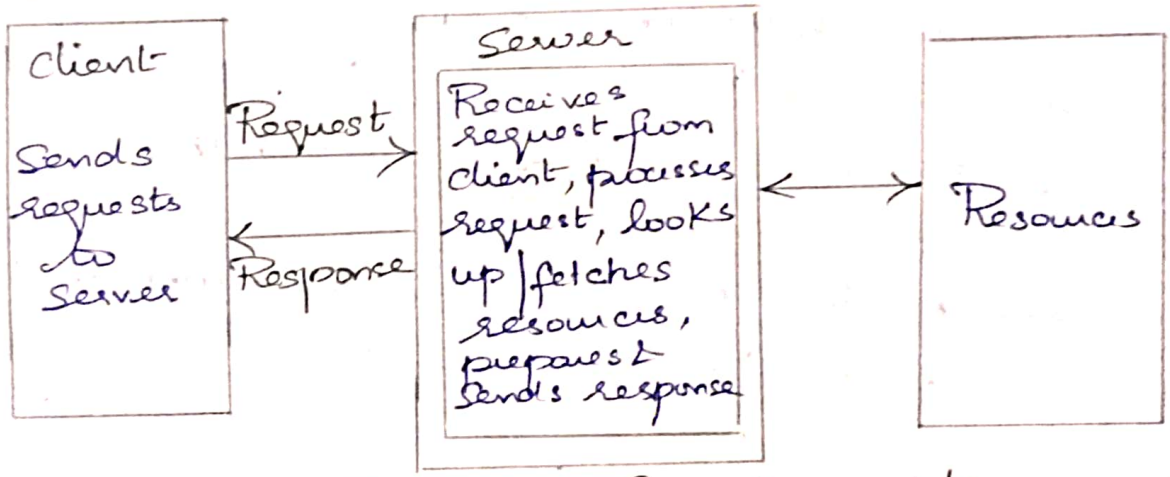


Fig - Request Response Comm^{n} models.

(ii) Publish-Subscribe:
→ It is Comm^{n} model that involves publishers, brokers & Consumers.
→ Publishers are the source of data & send the data to the topics which is managed by broker.

→ Once broker receives the data for a topic from publisher, it sends the data to all subscribed Consumers.

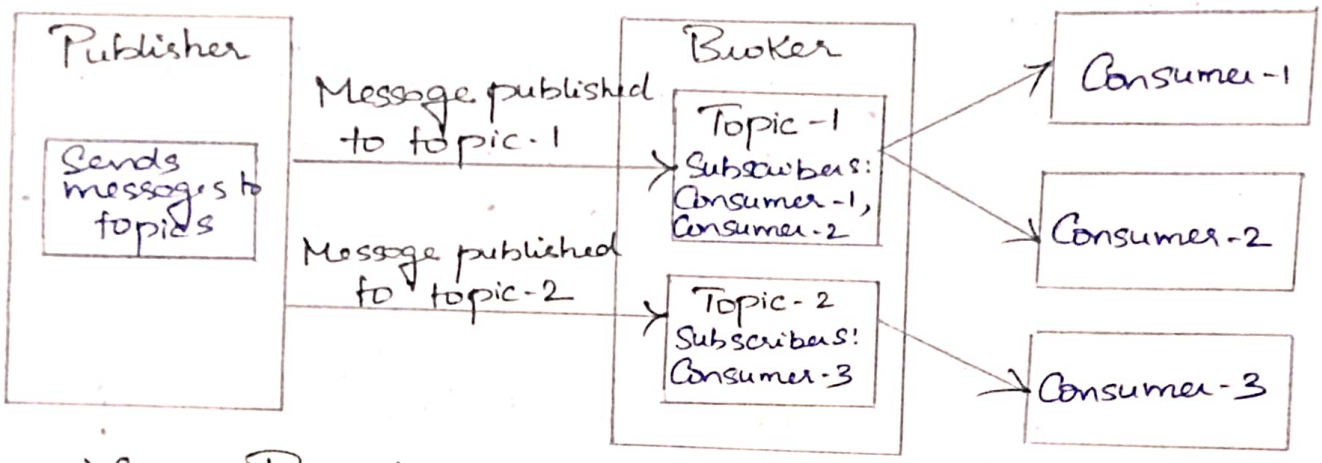


Fig - Publish-Subscribe Comm^{n} Model.

(iii) Push-Pull :-

→ Push-pull is a Comm^{n} model in which the data producers pushes the data in to the queues & Consumers can pull the data from Queues.

→ Queues help in decoupling the messaging b/w producers & Consumers.

→ It also acts as a buffer which helps in situations when there is mismatch b/w the rate at which the producers pushes the data & rate at which the Consumers pulls the data.

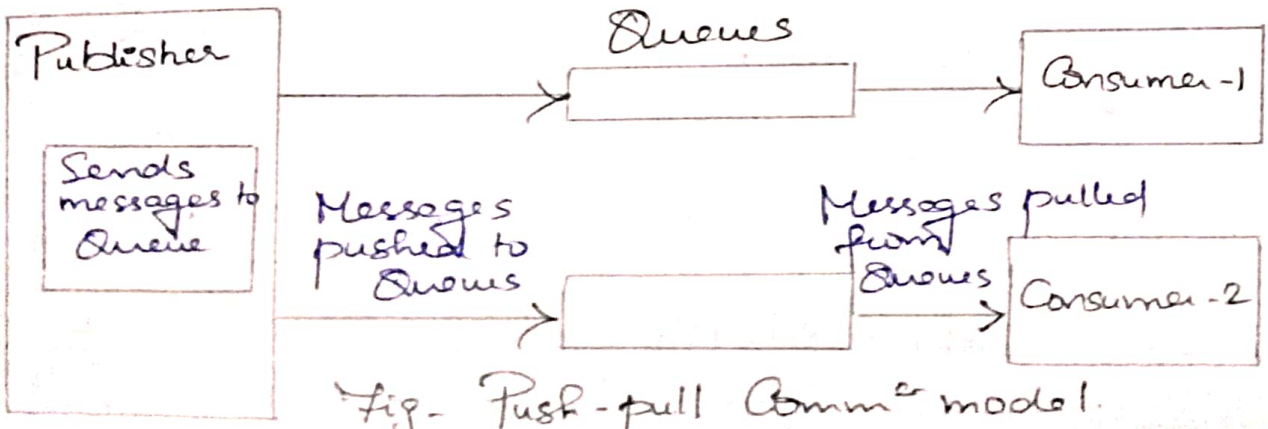


Fig- Push-pull Comm^{n} model.

IV IoT Enabling Technologies :-

(i) Wireless Sensor Networks (WSN) :-

→ WSN Comprises of distributed devices with the sensors which are used to monitor the environment & physical Cond's.

→ WSN Consist of no. of end-nodes & routers & a Co-ordinator.

→ End nodes have several sensors attached to them.

→ Each node act as a routers.

→ The Co-ordinator node Collects the data from all the nodes.

→ Co-ordinator also act as a gateway that connects the WSN to the Internet.

→ Some of WSN used in IoT are

(i) Weather Monitoring system
(ii) Air Quality Monitoring system
(iii) Soil Moisture Monitoring system
(iv) Surveillance system
(v) Smart Grids.

→ WSN are enabled by wireless Communication protocols such as IEEE 802.15.4.

→ Zigbee Can operate at 2.4 GHz frequency & offers data rate upto 250 Kb/s. & ranges from 10m to 100m.

(ii) Cloud Computing :-

→ It refers to the use of hosted services, such as data storage, servers, databases, networking & software over Internet.

→ The data is stored on physical servers which are maintained by cloud service provi

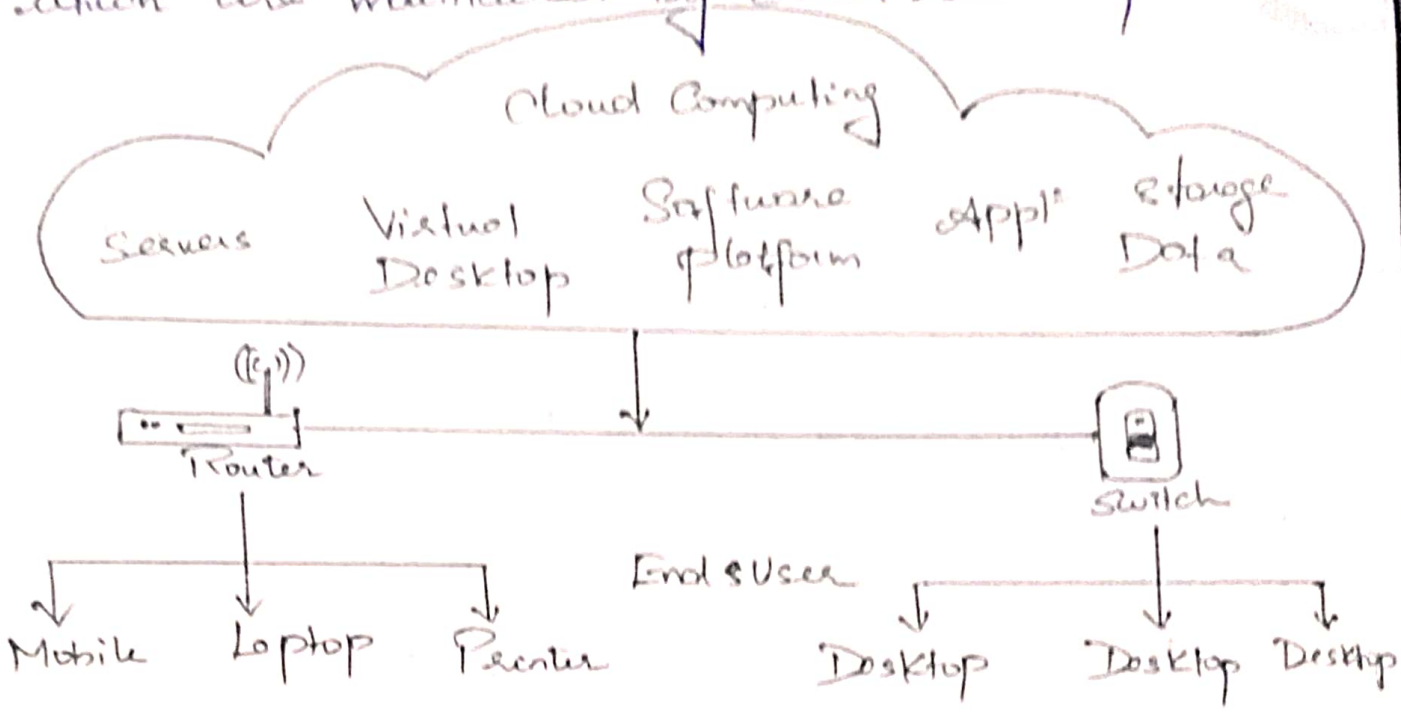


Fig. Cloud Computing Architecture.

(iii) Big Data Analytics :-

→ Big Data is defined as Collection of dataset whose volume is so large that it is difficult to store, manage, proces & analyze the data using traditional database & data processing tools.

→ The following steps are involved in big data analytics

(i) Data cleaning
(ii) Data Munging
(iii) Data processing & Visualization

→ Some eg of big data generated by IoT are

1) data generated by IoT sm such as weather monitoring stations
2) for location & tracking of vehicles.

(iv) Communication Protocols :-

→ Comm^n Protocols form the backbone of IoT s/m which enables n/w Connectivity & Coupling into appl^n.

→ It allows devices to exchange data over n/w.

→ These protocols define data exchange formats, data encoding, addressing schemes for devices.

→ This protocol also support sequence control that helps in ordering the packets to determine lost packets, flow control that helps in controlling the sender data rate with receiver data rate.

(v) Domain Specific IoTs :-

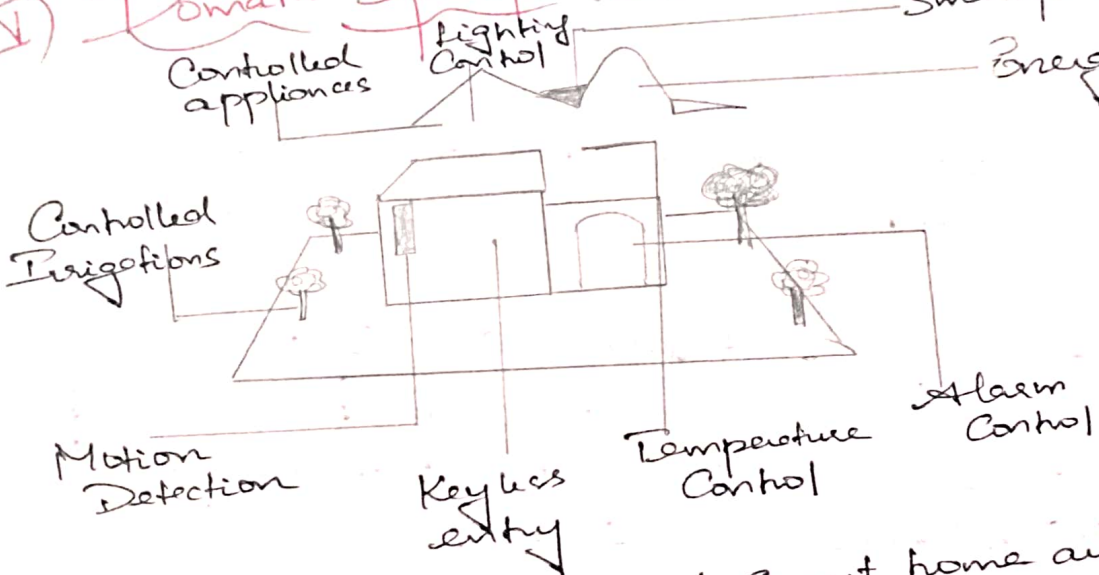


Fig - IoT based Smart home automation s/m.

(1) Home Automation

(i) Smart lighting :

→ Smart lighting for homes helps in saving energy by adapting the lighting to the ambient Cond^n & switching on/off lights when needed.

→ Smart lighting uses IoT enabled sensors, bulbs or adaptors to allow users to manage their home or office lighting.

(ii) Smart Appliances:

→ Managing & Controlling home appliances such as TVs, refrigerators, music s/m, washer/dryer etc is difficult because each appliance either having its own controls or remote controls.

Advantage:

→ Enable users to control, connect & monitor their appliances allowing to save time, energy & money.

(iii) Smoke/gas Detectors:

→ F.I use an optical detection, ionization or air sampling techniques to detect smoke.

→ Alerts raised by smoke detectors can be in form of signals to fire alarm system.

→ Gas detectors can detect the presence of harmful gases such as Carbon Monoxide (CO) & liquid Petroleum gas (LPG).

(2) Smart Cities:-

(i) Smart Parking:

→ Also known as Connected parking s/m which is a centralized management s/m that allows drivers to use a smartphone app to search for & reserve a parking spot.

Features:-

(i) Remote Parking Monitoring
(ii) Automated Guidance
(iii) Parking Reservation Mechanism.

(ii) Smart Lighting:

→ Smart lighting is for roads, parks & buildings can help in saving energy.
→ Smart lights equipped with sensors can communicate with other lights & exchange info on the sensed ambient cond= to adopt lighting.

(3) Health & Lifestyle:-

i) Health & Fitness Monitoring:

→ Wearable IoT devices allow non-invasive & continuous monitoring of physiological parameters can help in continuous health & fitness monitoring.
→ The wearable devices form a type of wireless sensor n/w called body area networks in which measurements form a no. of wearable devices.

(ii) Wearable Electronics:-

→ Wearable Electronics such as smart watches, smart glasses & wristbands that provide various fun? & features to assist us in our daily activities & making us to lead a healthy lifestyles.

(vi) IoT & M2M.

→ A Machine to Machine Connection is a Connection b/w 2 machines without any human Interaction. (ii) it allows two devices to Communicate autonomously.

→ M2M devices Can autonomously Collect & Transmit data, facilitating real-time decision making & Improving ?.

M2M Architecture :-

→ M2M Consists of
- (i) M2M Area Network
- (ii) Communication n/w &
- (iii) Application Domain.

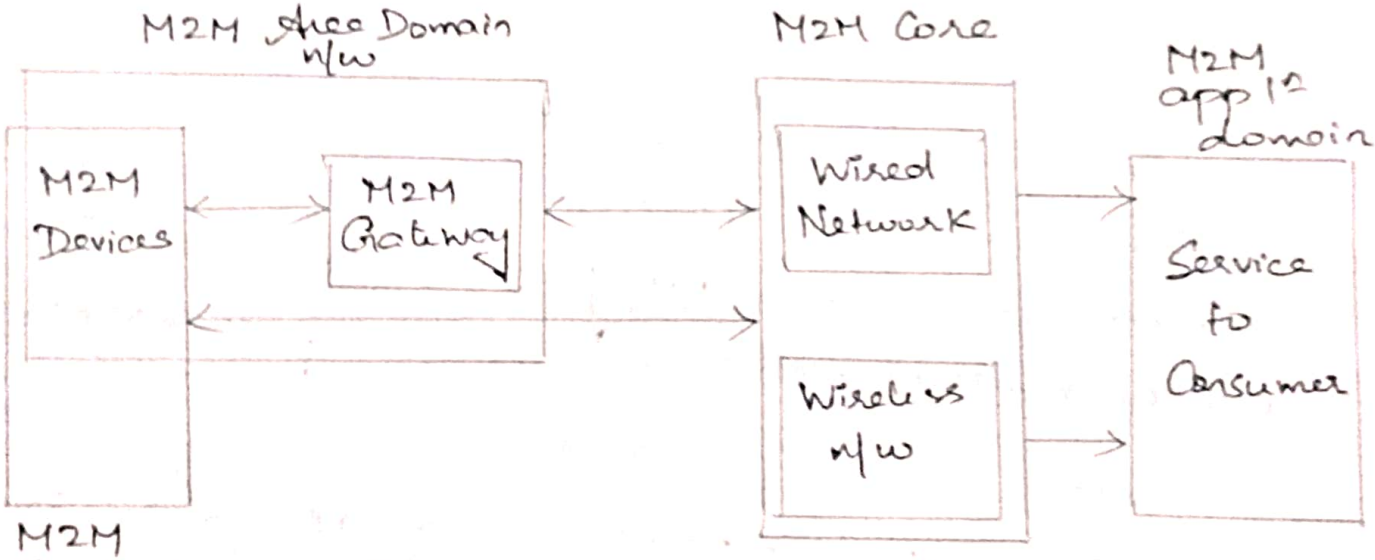


fig - M2M system Architecture.

(i) Area Network:

→ M2M area n/w Compuises of machines as nodes which have embedded h/w modules for sensing, actuation & Communication.

→ Various Comm^n protocols that can be used for M2M local Area n/w such as Zigbee, Bluetooth, Modbus, M-Bus, 6LOWPAN, IEEE 802.15.4 etc.

Gateways:-

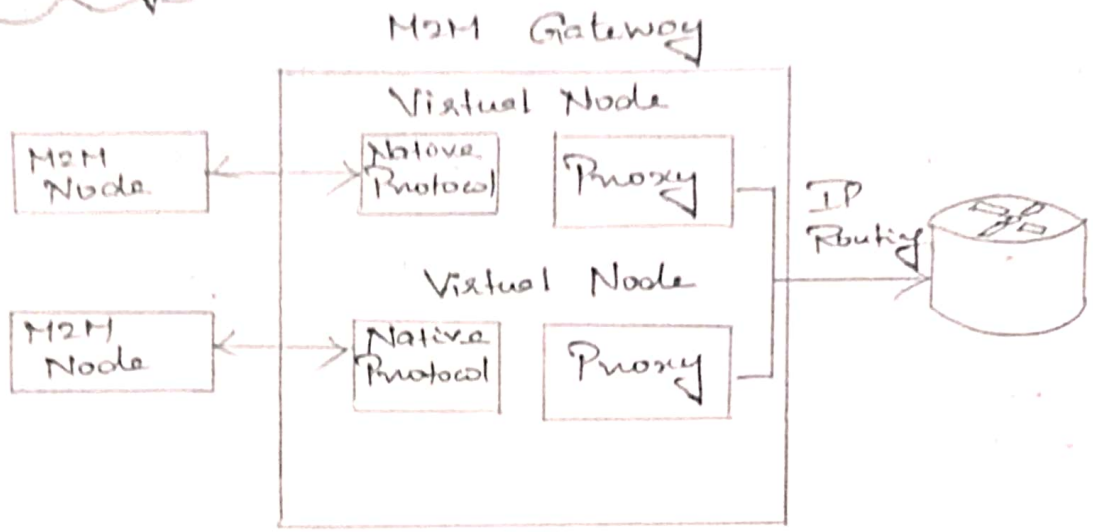


Fig. Block Diag of M2M gateway.

→ M2M gateways are used to bridge one or more locally n/w devices to a wired or wireless broadband Comm^n.

→ M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol (IP).

(ii) M2M Care:-

→ The Comm^n n/w can use either wired or wireless n/w while M2M area n/w use non-IP based Comm^n protocols (i.e) M2M nodes within one n/w cannot communicate with nodes in an external n/w.

(iii) M2M Applications:

→ Smart metering
→ Home automation
→ Industrial automation -

IoT	M2M
→ Internet-based Connectivity	→ Direct Machine-to-Machine Communication
→ Uses the Internet & Cellular n/w	→ Uses either an Internet or non-Internet Conn°.
→ Cloud based storage analytics.	→ Local data processing
→ Most Scalable	→ less Scalable.
→ Efficient resource utilization	→ Optimized resource utilization.

VII) IoT System Management with NETCONF-YANG:-

(i) Need for IoT System Management:

(i) Automating Configuration
(ii) Improved reliability
(iii) System Wide Configurations
(iv) Multiple system Configurations.

Network Configuration Protocol (NETCONF):-

→ NETCONF is a n/w management protocol allowing a n/w management s/m (NMS) to deliver, modify & delete Config° of network devices.

→ Standard Appl² programming Interfaces (API) are available on n/w devices for NMS to manage the devices using NETCONF.

→ NETCONF uses Extensible Markup Language (XML) - based data Encoding for Config & data & protocol messages & uses a Simple Remote Procedure Call (RPC).

Basic Network Architecture of NETCONF :-

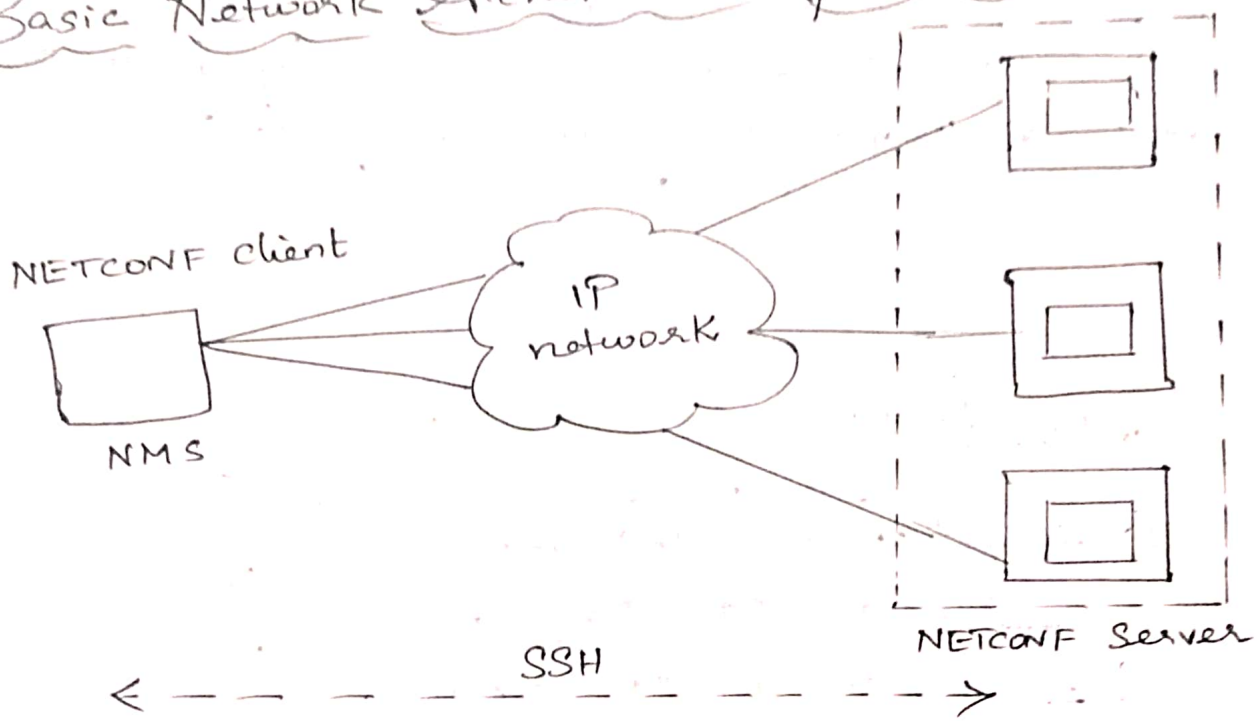


Fig - Network Architecture of NETCONF

→ NETCONF architecture Consists of two roles: client & Server.

(i) client provides the following func.
→ Manages n/w devices using NETCONF.
→ Sends RPC requests to NETCONF Server to query or modify one or more parameter values.
→ Learns the status of moneged devices.

(ii) Server maintains infor about moneged devices & responds to the client-initiated requests.
↳ When receiving a request from a NETCONF client, the NETCONF Server parses the request &

Sends a reply to the client.

② Establishing a NETCONF Session:

→ The NETCONF client & server uses the RPC mechanism to communicate with each other.

→ The client sends an RPC request to the server, & the server returns a reply to the client after processing that request.

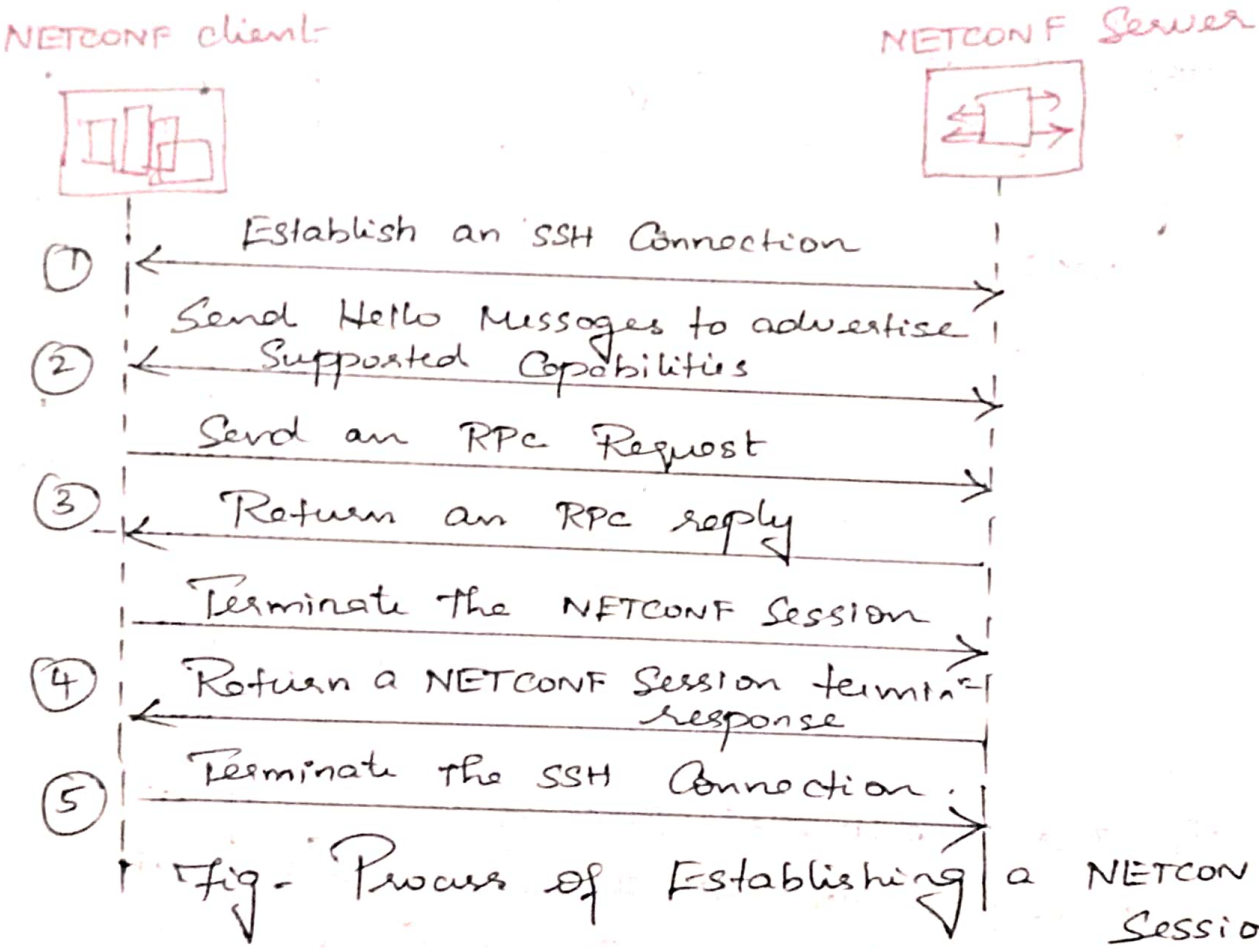


Fig- Process of Establishing a NETCONF Session.

→ A client establishes a Secure Shell (SSH) Connⁿ with a server & then establishes a NETCONF session with the server once the authentication & authorization are complete.

→ Client & server send Hello Messages to Negotiate Capabilities.

YANG - Yet Another Next Generation :

→ YANG is a data modeling language used to model Config & state data manipulated by the NETCONF protocol.

→ YANG modules contain the definitions of the Config data, state data, RPC calls that can be issued to maintain the format of notif.

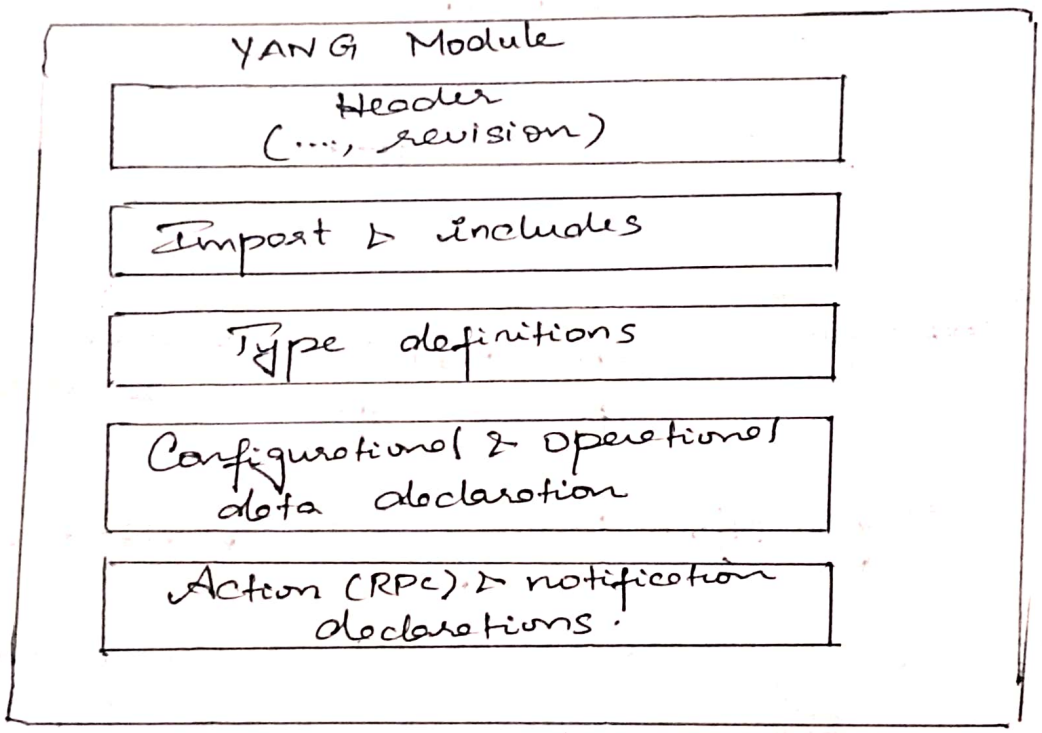


Fig - YANG module.

→ YANG modules define the data exchanged b/w the NETCONF client & server.

→ YANG is a modular language representing data structures in XML tree format.

→ YANG model is integrated on the devices, which function as a servers.

→ NETCONF or RESTCONF to centrally manage, configure & monitor various YANG-capable n/w devices.

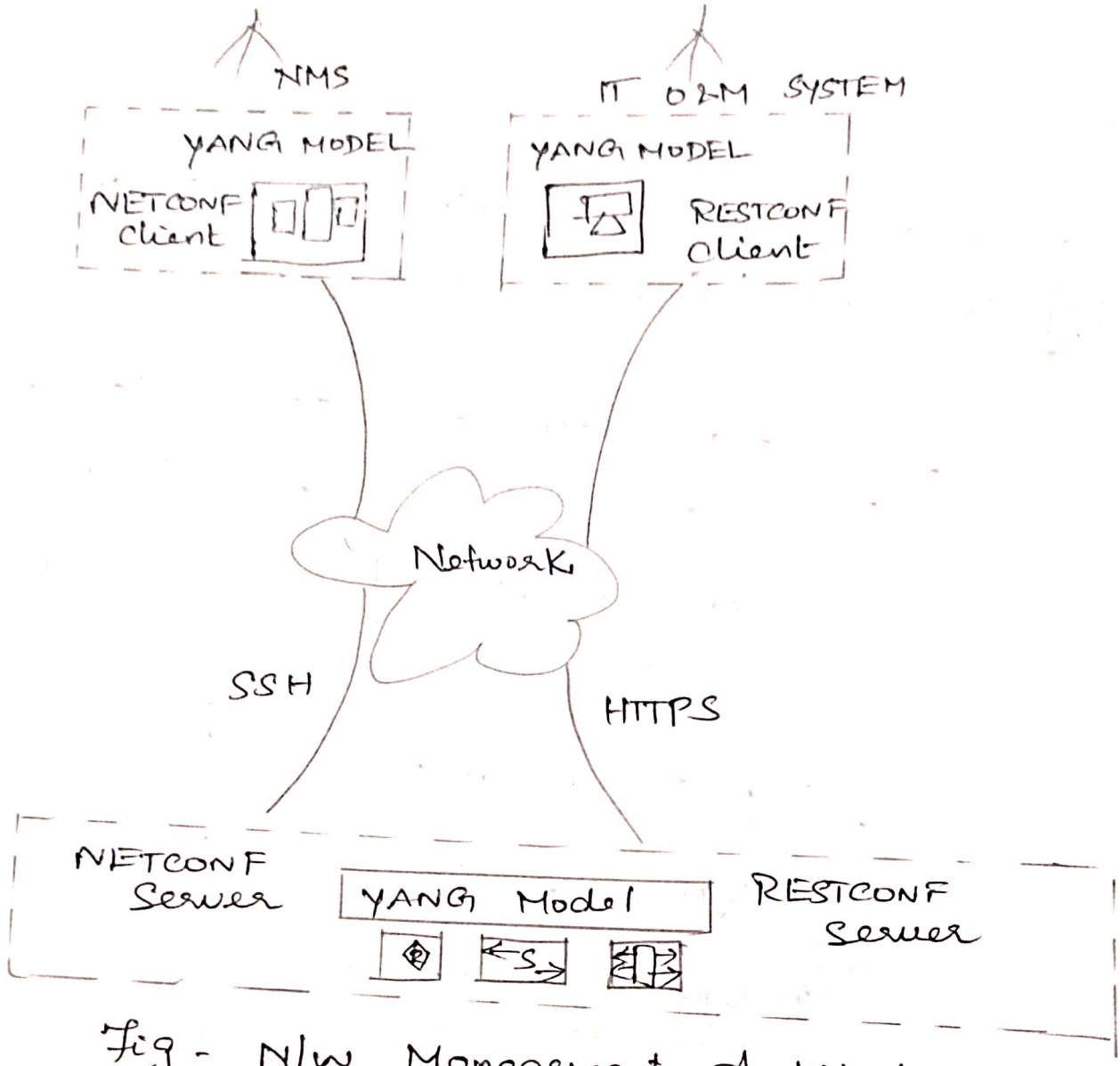


Fig - N/W Monogement Architecture.

→ This Module Comprises of number of leaf nodes that are orgonized into hierarchical tree structure.

→ Leaf nodes are orgonized using Container or list Constructs.

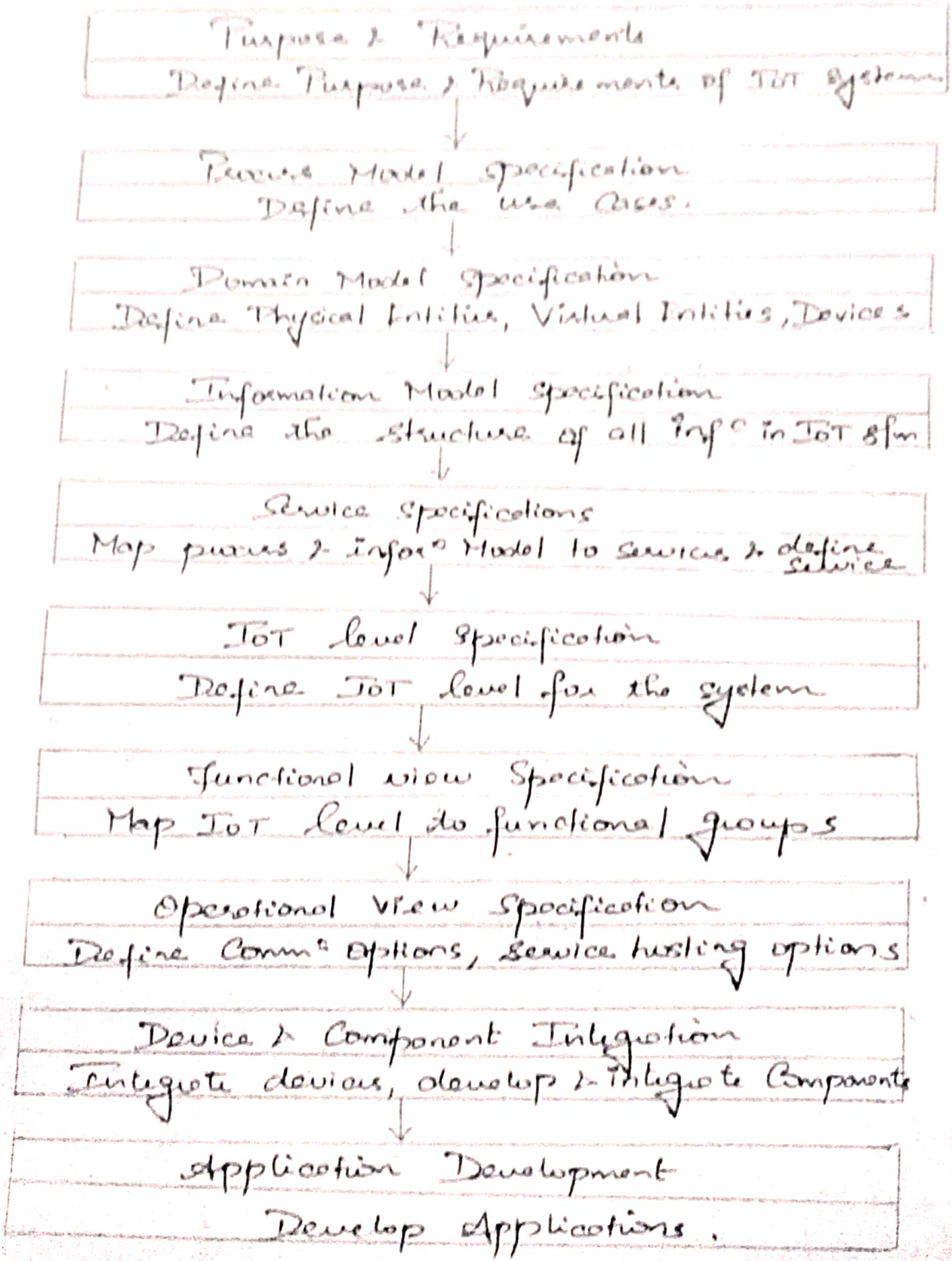
→ A YANG module Can import definitions from other modules.

→ Constraints Can be defined on data nodes eg. allowed values.

IoT Platforms Design Methodology

Step 1: Purpose & Requirements of IoT sys:

In this step, system purpose, behaviour & requirements such as data collection requirements, data analytics requirements, system management requirements, data privacy & security requirements are defined.



Step 2:
→ The use cases of IoT system are formally based on & derived from the purpose & requirement specif

Step 3:
→ The Domain model describes the main concepts, entities & objects in the domain of IoT s/m.
→ Domain model defines the attributes of objects & relationships b/w the objects.

Step 4:
→ The information model defines the structure of all the infor^n in an IoT s/m.
→ It does not describe how the inf^c is represented or stored.

Step 5:
→ Service specifications define the services in IoT system, service types, service i/p / o/p, service endpoints.

Step 6:
→ Defines IoT level for the s/m.

Step 7:
→ This step describes the functional view which defines the fun^s of IoT s/m grouped into various functional groups.

Step 8:
→ The various options pertaining to IoT s/m deployment & operation are defined such as service hosting options, storage options, device opts.

Step 9:
→ Defines the overall connections of all IoT devices & components in IoT platforms.

IoT Reference Architecture & Reference Model. (1)

→ An Architecture & Reference Model (ARM) Consists of 2 main parts

(i) Reference Architecture &
(ii) Reference Model.

→ A Reference Model is a model that describes the main Conceptual entities & how they are related to each other, while reference architecture aims at describing the main functional components of a SFm as well as how the SFm works.

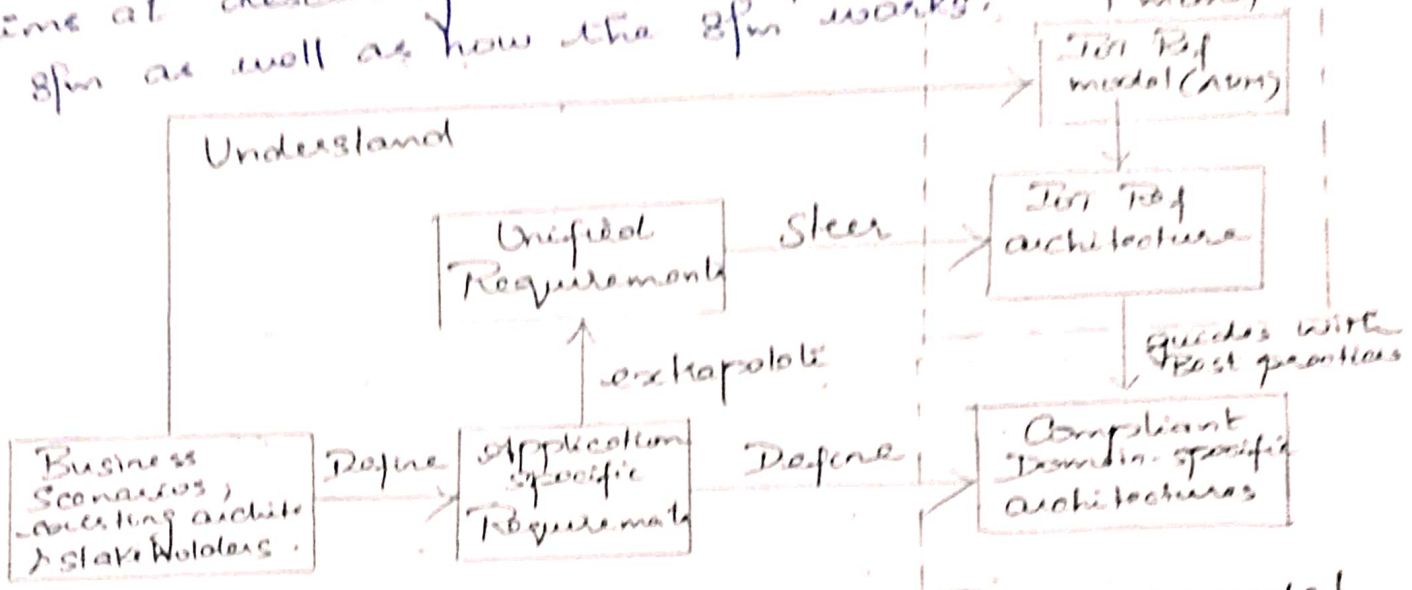


Fig. IoT Reference Architecture & Reference model.

IoT Reference Architecture:

→ It outlines the essential components & interactions within IoT System.
→ It provides a solid starting point for designing & implementing IoT sol°.
→ IoT functional view for IoT Ref. architecture is described below.

(i) Device & application functional group (FG):
→ The physical devices or sensors that collect

data from environment
→ These devices can include temperature sensors, motion detectors, Cameras.

(ii) Communications functional Group:-
→ It takes care of node-to-node (hop-to-hop) Comm^n until they reach a gateway node which forwards the message to Internet.
→ Network Functional Components (FC) is responsible for message routing & forwarding.

(iii) IoT Service functional group.
→ IoT Service FC is a Collection of service implementations which interface the related & associated resources.

(iv) Virtual Entity (VE) functional group:-
→ It Contains functions that support the interactions b/w users & physical things through Virtual Entity Services.

(v) Process Management functional group:-
→ The process modeling FC provides the right tools for modeling a business process that utilizes IoT related services.
→ Process Execution FC Contains the execution environment of process models hosted by FC.

(vi) Service Organization functional group:
→ Service Composition FC manages the descriptions & execution environment of complex services Consisting of simpler dependent services.

IoT Reference Model:

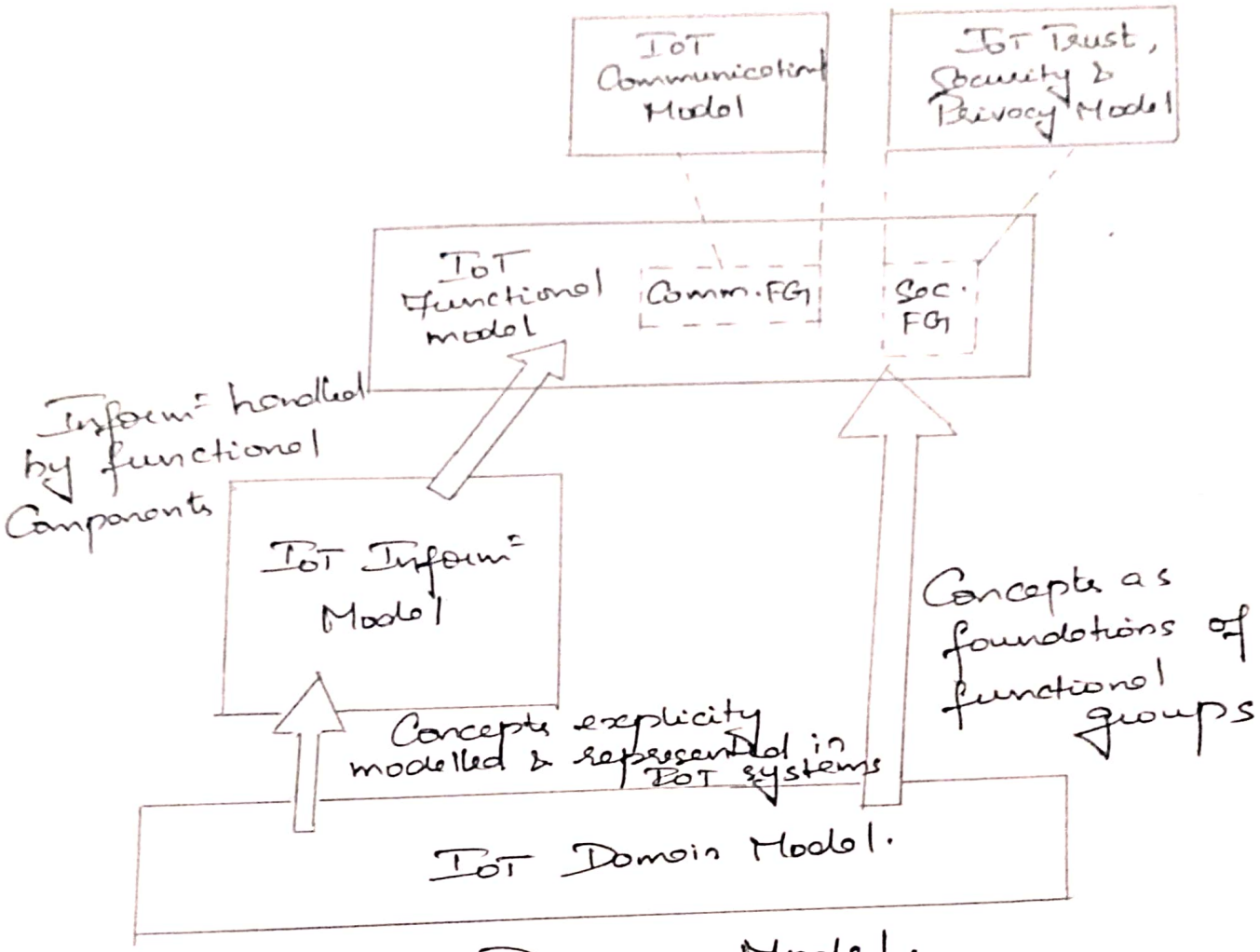


Fig - IoT Reference Model.