Embedded system Concept

① Embedded system Definition

↳ it's computing system with limited Resource to do specific task

② E.S Component

- ↳ Processor → single core
- ↳ Memory → ROM 32 kByte, RAM 2 kByte
- ↳ IO Peripheral → No of Peripheral

→ limited Resources

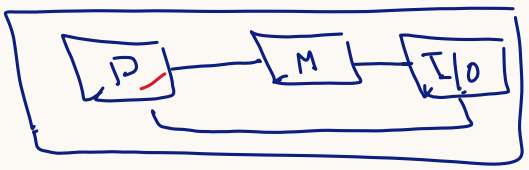
③ Embedding system

↳ it's multi Embedded system together

④ How Design E.S

system on Board

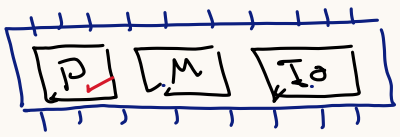
SOB [Pcb / white]



↳ Connection (wire / track)

system on chip

SOC [Factory]



↳ Connection (tunnel)

→ Performance =
→ Cost = Based on Phase for H.W (dev / pro)
→ size =
→ Power Consumption =

→ Performance
→ Cost
→ size
→ Power Consumption

development H.W
SOC → Cost ↑
SOB → Cost ↓

Production H.W
SOC → Cost ↓
SOB → Cost ↑

Development H.W → SOB | Production H.W → SOC

Main Component

↳ ① Processor ② Memory ③ I/O Peripheral.

→ Processor

↳ Mathimatical & logical operation

data → [P] → Information

↳ History & type

X ← ↳ Based on tube

↳ Based on transistor

(Processor)

(Micro Processor)

(CPU)

Centeral Processor Unit

in E.s → Single Core

↳ μProcessor = CPU

↳ Processor ⇒ μProcessor ⇒ CPU

* Diffierent between μProcessor → E.s (Micro Controller)

μ Processor (μPU)

* it's Part From μCU

μ Controller (μCU)

* the E.s Chip

AVR → Atmega3

PIC → 16F877A

ARM → STM32 F103

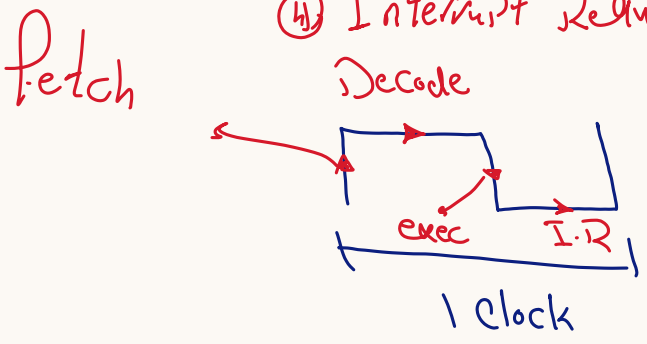
Tiva_C

* Inside Processor

↳ Processor Communicate with memory only
  ↳ Rom → to get code
  ↳ Ram → to write / Read data
  ↳ I/o memory → to control on out Put device

↳ Processor need (Clock / Freq) to make operation

↳ Processor → Run Machine Cycle
  ↳ ① fetch ② decode ③ execute
  ④ Interrupt Request



Machine Cycle

↳ ① Fetch → Processor go to Rom to get Instruction will be execute.

↳ ② Decode — analysis for Instruction to get
  ↳ operand 1 ↳ operand 2
  ↳ op code → operation code

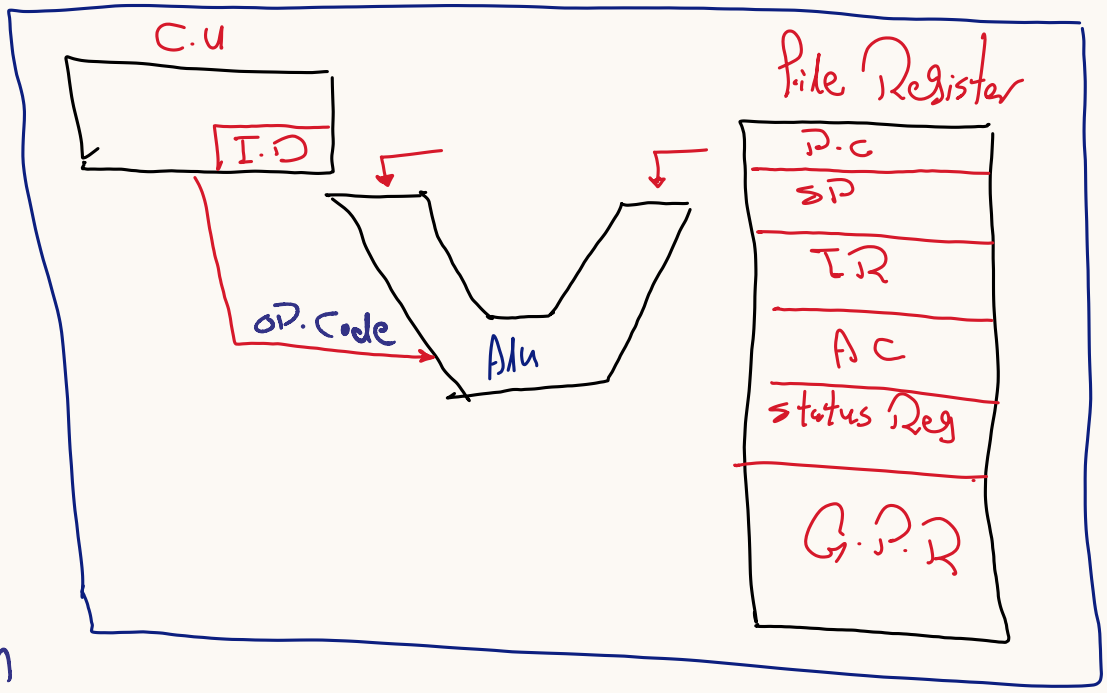
↳ ③ execute → Run operation

→ Main Component

↳ Control unit (CU)

↳ Arthimatical logical unit (Alu)

↳ File Register (Register Bank)



Fetch

→ P.C → Program Counter

↳ store Address → next Instruction

↳ Ram

→ get Instruction & store it in IR → Instruction Register

→ Add step for Pc (PC++)

DeCode

↳ Control unit take Instruction from IR

↳ Pass this Instruction into I.D → Instruction Decoder

I.D → it's part from C.U

I.D → Can be logical circuit
→ small chip

I.D → Detecte the operand
→ Detect the op-Code.

I.D → op Code type

Not Mathimatical / logical

ex: store
Call
load

→ this op-Code
the C.U will execute
it

→ Pass for op Code into
C.U

Mathimatical / logical

Add
sub
Inc

→ this op Code
the ALU will execute
it

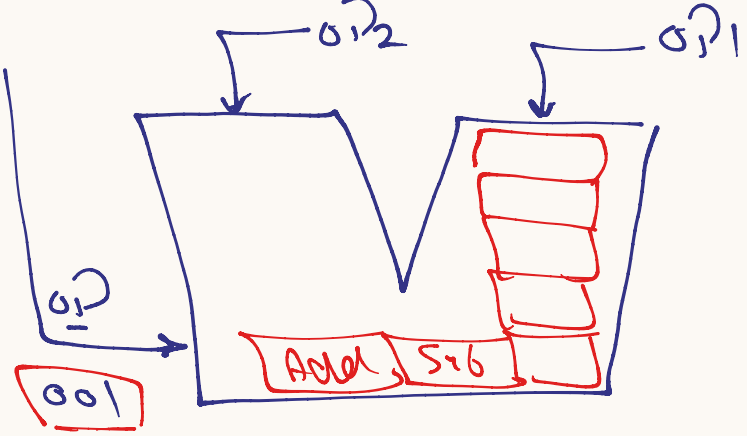
→ Pass op Code into
ALU

execute

↳ if op-Code type is Non Mathmatical or logical
the C.U will execute

↳ if op-Code type is Mathmatical or logical
the ALU will execute

ALU



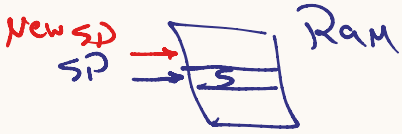
AC → ACCUMULATOR → store the Result execute ALU

---

① Control unit ✓

② ALU ✓

③ Register Bank → P.C : Program Counter
- I.R : Instruction Register
- AC : ACCUMULATOR
- status Register :
- SP : Stack Pointer
  - Pointer → in RAM.



- general Purpose Register

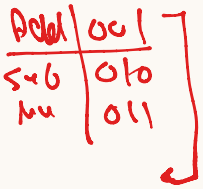
Function Purpose Register ←

ISA : Instruction Arch

↳ Risc → Almost No of Instruction
↳ 100 → 131
↳ execute Almost Instruction in one clk. → 1

↳ CISC → Almost No of Instruction
↳ 1000 ≅ 1000
↳ execute Almost Instruction in 4 clk

Compare between Risc & CISC

	Risc	CISC
Per Formance	→ 4 + 4 → 1 clk → 5 X 4 → 4 clk 5 + 5 → R1 → R1 + 5 → R2 R2 + 5 → R3 → → 5 X 100 ~	→ 4 + 4 → 4 clk → 5 X 4 → 4 clk → 5 X 100 ~
Cost	H.W ⇒ Low Cost SW ⇒ High Cost ~	H.W ⇒ High Cost SW ⇒ low Cost ~
Size	Alu → size ↓ I.D → logical circuit  ~	Alu → size ↑ I.D → chip ↓  ~

Power Consumption	ALU → Run one Instruction ~	Circuit at one ~

Apple

↳ MacBook → Based on Intel

↳ CISC

↳ New versions → Based on ARM

↓

RISC