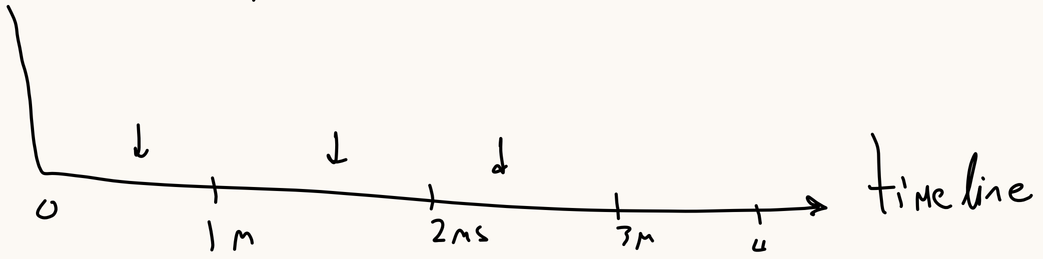
# Interrupt

↳ Some Concept

↳ Determinizm →

↳ in sw :: knowing what is happening at every Point in time line



↳ ResPonsivness

↳ How fast you can Respond to external Action (event)

---

system type

① super loop → while(1) → ✓
② For & Back ground system → Interrupt →
③ operating system → Rtos → x

*super loop → Your Project write inside infinite loop and Instruction execute in ordered.

ex

```
while(1)
{ . MDIo (GroupA, Pino, High); ✓
  . hLCD_Send char ('A'); ✓
  . h SSD_ DisPlay Number (three); ✓
state = h BtN_ get is Pressed (GroupA, Pin5); x
  if (state == Pressed) →
  { Action;
  }
}
```

Advantage

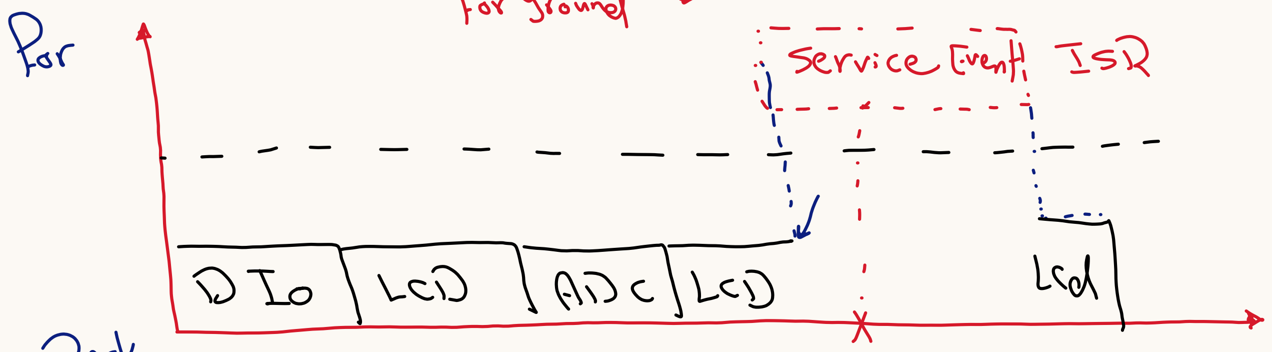
① High determinism
② simple SW
③ Minim H.W

disadvantage

① Low Responsivness
② High Power Consumation

② For & Back Ground system

↳ divide Application Component → Driver
to → (High Priority / Normal)
For ground ← High Priority
Normal → Back ground



Back
↓
inside superloop

→ Interrupt →

→ ISR → Interrupt service Routine
↳ Function → execute → just → event Happen.

Advantage

① High Responsivness.

disAdvantage

① Low determinism.
② Extra H.W
③ Complex SW
↳ complex Design SW
↳ Design Pattern

③ Rtos

# Interrupt

→ to Interrupt Processor → Must be have 3 factor

① Specific Interrupt Enable (SIE) → ⓪ ①

ex ADC → ADIE → ADC Interrupt Enable
↳ ①

② Global Interrupt Enable (GIE) → ⓪ ①
X ↳ 0

③ Peripheral Interrupt Flag (PIF)
↳ ①

---

① SIE → give allow to specific peripheral to make Interrupt

② GIE → give allow to concept interrupt is work

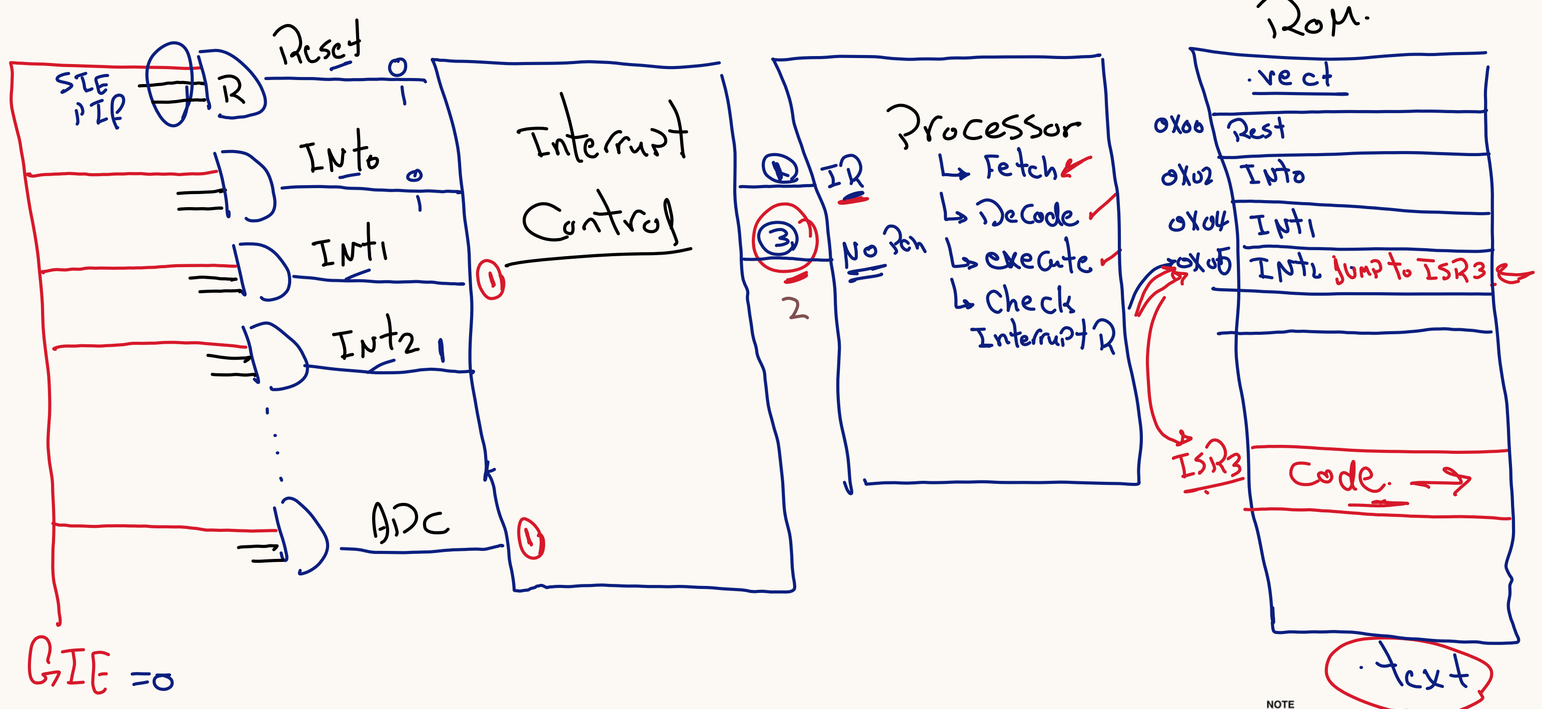
③ PIF → if event happen will set this flag.

---

## Interrupt Handle

[1] By vectorable Proirity ←
[2] By Flexable Proirity

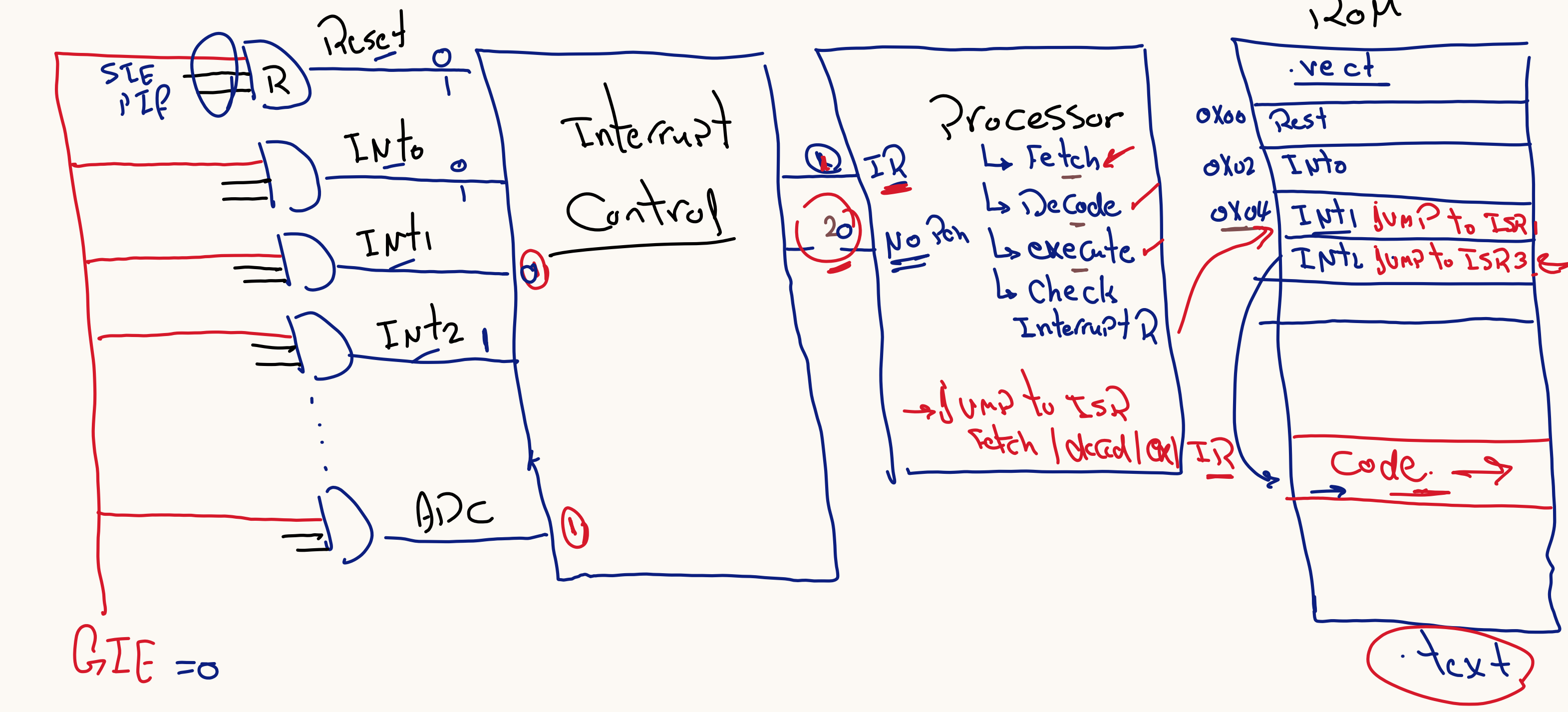
### → By vectorable Proirity.



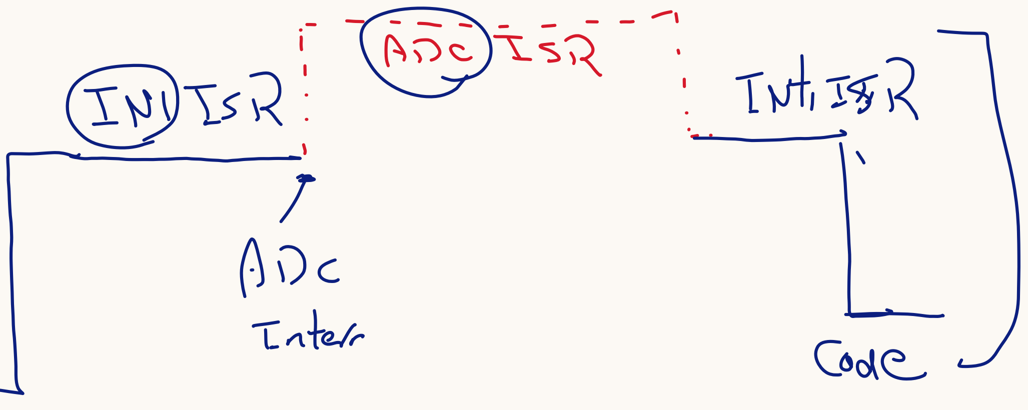
* vector table → ① which Periphel have Interrupt Circuit
↳ ② Fixed Priority

Advantage
↳ system is very Fast
↳ every thing happen by H.W

dis Advantage
① High Cost
② No change Priority
Fixed by H.W



Nested ISR →



Atmega32 Not support Nested ISR
↳ Disable For GIE

* Processor steps when Interrupt happen (AVR_Processor).

① Complete execution of Current Assembly Instruction -

② Disable Global Interrupt -

③ Save Program Counter in RAM(stack) -

④ Save CPU Register (SP, Accumalter, status Reg - General Purpose Register)

⑤ Jumb to vector table in .vect section (.vector) -

⑥ jumb to ISR in .text section → start execute ISR

⑦ get the CPU Register (SP, Accumalter, SR, GPR)

⑧ get the PC

⑨ Enable GIE

⑩ jump to Continue the Code

---

Revision Context switching

↳ it operation to save the last Point of Code before jump to another Code

↳ save the Processor Registier instack

↳ Context switching type

H.W C.S

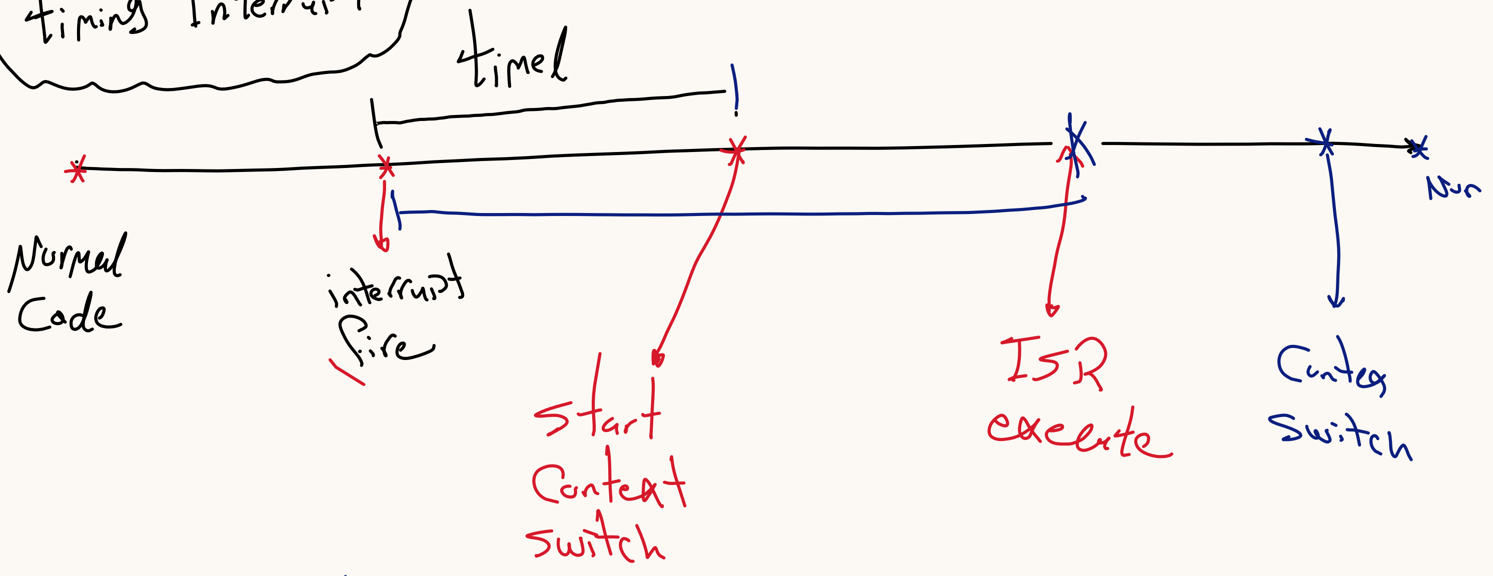
↳ Interrupt

SW C.S

↳ Function

↳ code Assembly code.

## timing Interrupt



First time :. it's time between Event happen and start Context switch

→ [Interrupt latency]

Second time !. it's time between event happen & Process execute ISR

→ [Interrupt Response]

---

Flag Handle

↳ this bit represent the event happen or Not

↳ After Handle event the Flag must by Clear

GIE →

SIE →

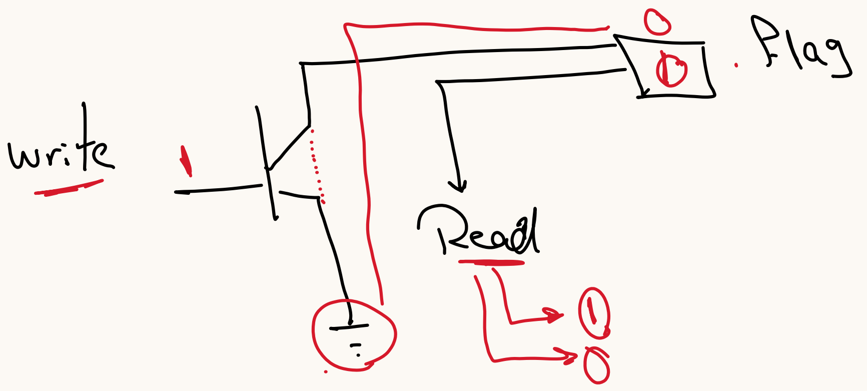
PIP →

(How clear Flag)

↳ ① Auto clear → b.1 H.w when You Enable SIE & start execute the ISR.

stm PIC ← ↳ ② write Zero → b1 sw → Flag cleared

↳ ③ write one → Atmega → set Bit ⇒
↳ Cleared.



---

Event Handle

Polling → waiting
↳ Flag → set

* wait untill Flag on
  ↳ take Action
* Must be clear Flag by sw
* type of Busy wait

Interrupt →
↳ set | GIE | SIE
↳ Cont work
↳ when Flag is one
↳ jump to ISR

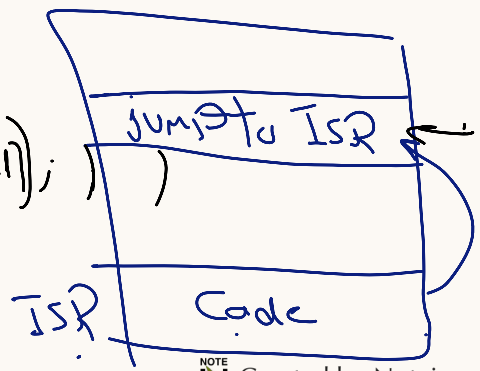
---

ISR
↳ it's Function has the Action will Happen when event Fire
↳ this Function Call by H.W

How write ISR dependon target

Void __vector_No (void) __attribute__((signal));

void __vector_No (        )
{
   Action;
}



$$\text{void \_\_vector\_No(void)} \underbrace{\text{\_\_attribute\_\_((signal));}} \rightarrow \text{Prototype}$$

```
void __vector_No( )
{
    =
}
```