## Interrupts

IVT contains all the ISR addresses.

IVT is in memory.

ISRs are also in memory.

→ MP gets an interrupt, want to got to ISRs. i.e. memory.

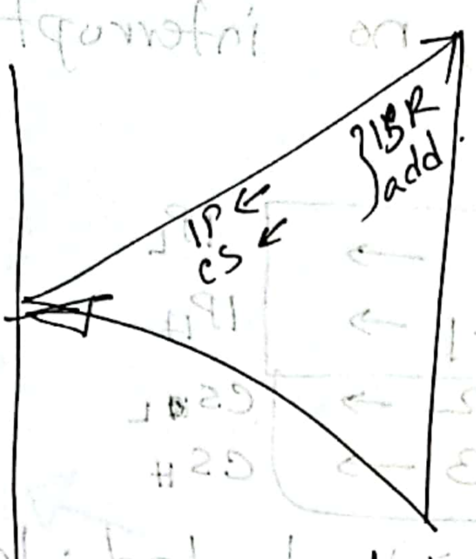
→ But 256 interrupts are scattered in the memory (all over 1 MB memory) (10 lakh memory locations)

→ So, whenever μp gets an interrupt, it goes to memory twice.

→ 1st. it goes to memory just to obtain ISR address. from IVT

→ Once, it gets an ISR address from IVT, it will load that address into IP & CS

→ So, IVT basically vectors the μp to go to the actual ISR address corresponding to the Interrupts.



IVT does not contain subroutine; it contains the address (ISR add) for that corresponding subroutine.

→ IVT addresses are fixed, thankfully, otherwise that addresses should have been stored in some other location.

→ $\boxed{\text{00000H is the permanent add. of IVT to FFFFFH}}$

↳ physical add.

→ I~~nterrupts are 32 bits~~.

→ size of IVT is 1 KB of 1 MB memory
→ one thousand of one million location.

→ 256 interrupts × 4 Bytes = 1024 kB
≃ 1 KB.

→ without IVT, no interrupt can be serviced.

→

| | | |
|---|---|---|
| Int low | n × 4 → | I.P$_L$ |
| | n × 4 + 1 → | IP$_H$ |
| High | n × 4 + 2 → | CS$_L$ |
| | n × 4 + 3 → | CS$_H$ |

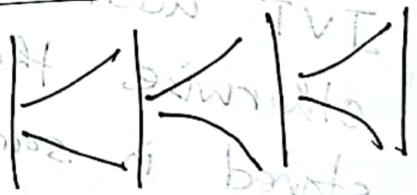
→ Lower byte ~~is~~ is loaded in lower address

→ higher " ,, " loaded in higher " "

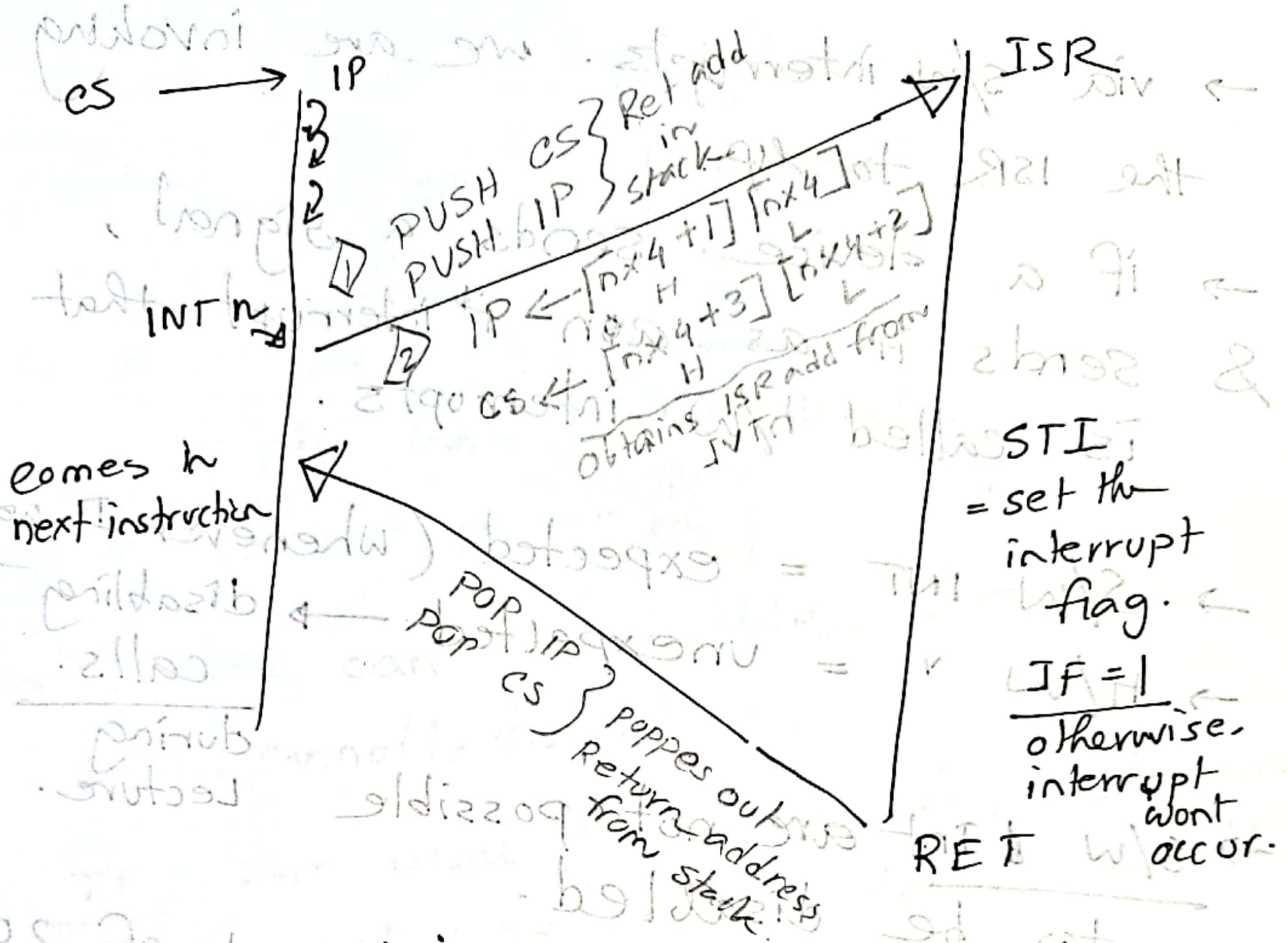
→

| CS H\|L | IP H\|L |
|---|---|
| Higher | lower |

☑ nested interrupts ⟶



☑ by making IF = 0, does not mean we are not disabling all the interrupts,



remember: it is intersegment, since interrupt does not associated with your normal operation. i.e. playing games.

→ Software interrupts are given by the programmers. by writting the ~~int~~ instruction INTn.

→ ~~hoa~~ H/W interrupts given by the circuits or device.

'n' can be any number from 0 to 255.

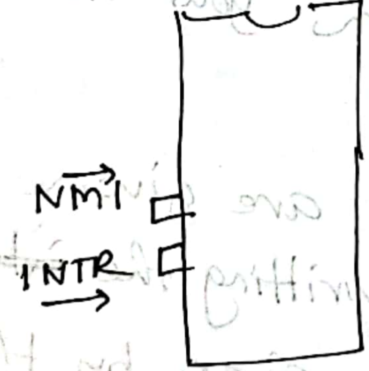
→ via s/w interrupts. we are invoking the ISR to up.

→ if a device sends a signal, & sends it as ~~a~~n ~~i~~ interrupt that is called h/w interrupts.

→ S/W INT = expected (whenever I want)
→ H/W " = unexpelted → disabling calls. during Lecture.

→ S/W ~~I~~ Int are not possible to be disabled.

→ there are 2 h/w interrupts. out of 256.



NMI example: when motherboard gets overheated.

→ airbag is an example of NMI.

→ only NMI cannot be disabled among the hardware i.

→ ~~e~~ in every ~~[illegible]~~ hardware device, some interrupts cannot be disabled, for car it is airbag.

→ only INTR can be disabled ; so ISR cannot be disturbed.

⇒ TF - is used to perform single stepping
→ you go line by line to debug a program., like pressing F7 in ~~sing~~ C programming.

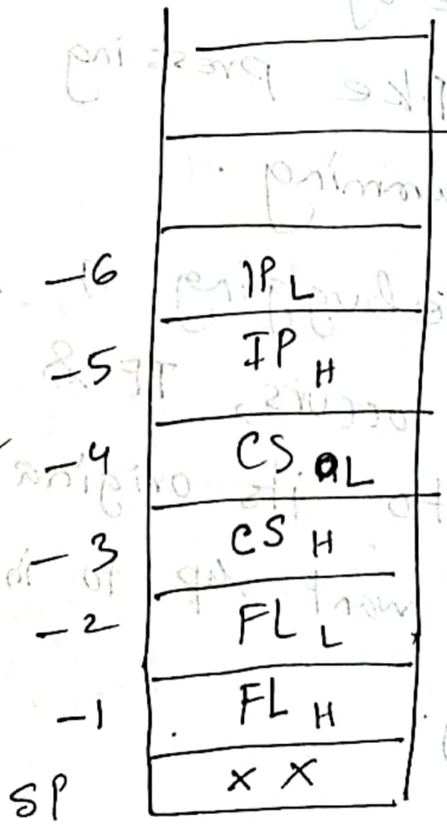
⇒ So, during debugging a prog. if an interrupt occurs, TF & IF should be restored to its original value, since, I don't want µp to interfer with my debbugging.

Scanned with CamScanner

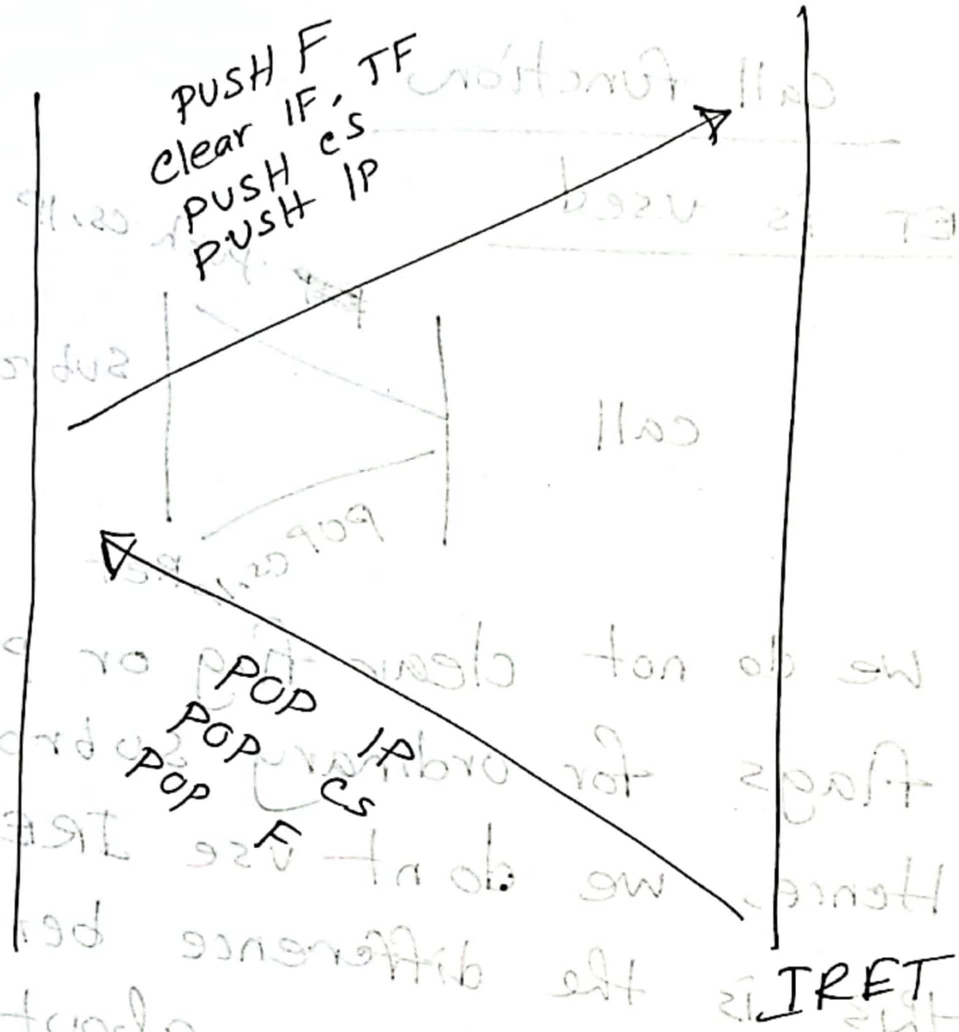
→ hence, before clearing IF & TF, we need to PUSH F i.e. push the whole flag into stack.

→ So, after servicing ~~inte~~, any interrupts, all the contents of flags will be restored to its original value; ~~be~~ like before : POPF

⇒ So, the ~~total~~ scenario of ~~flag~~ wi stack will be :



eve



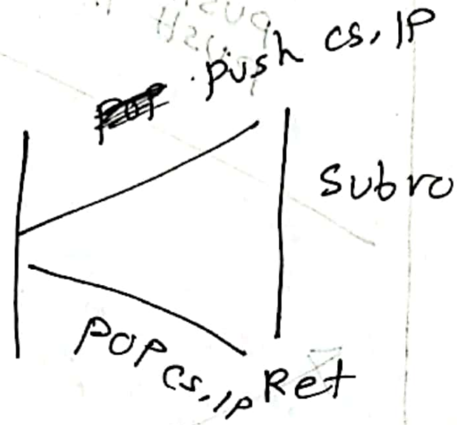
→ the command at the end is IRET → interrupt return.

→ Ret is written for supporting ordinary function.

→ IRET is written at the end of every ISR.

## call function

RET is used



we do not clear flag or push/pop flags for ordinary subroutine. Hence, we dont use IRET. this is the difference bet RET & IRET. RET does not care about flags.

→ By clearing IF & TF,

IF disables INTR ⇸ line

TF " single stepping

we do disable INTR, so that we can entertain one ISR at a time.