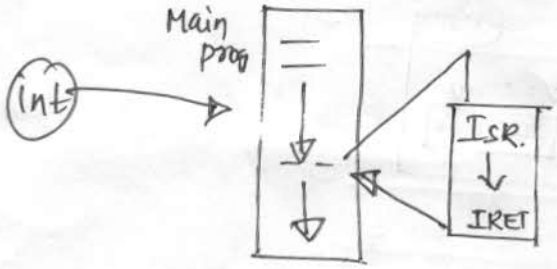
Interrupts..②

Interrupt is a request to a microprocessor to suspend its main program under execution and execute ISR (Interrupt service routine) to service the request.

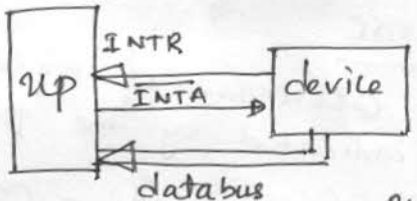


Based on the source of interrupt, they are classified as follows,

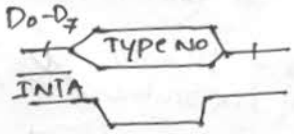
① Hardware Interrupts (external): These are raised by the external devices like KB, Mouse .., to gain attention of up. So every up provides few Interrupt pins, to support this feature. 8086 provides, NMI & INTR pins as external Hardware interrupts.

NMI – * Non maskable Interrupt (can't be disabled by instn)
* Vectored interrupt, (TYPE 2 interrupt)
* edge triggered type, ↑ on raising edge int is accepted,

INTR – * Maskable interrupt (enabled using ~~EI~~ STI & disabled using ~~DI~~ CLI instructions) (STI & CLI)
* Non vectored, so that external device supplies interrupt type no to up using data bus and $\overline{INTA}$ signal. (perticular type NO. is not predefined)



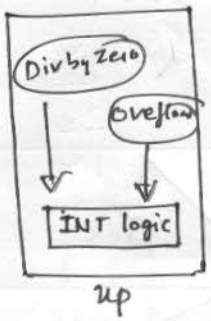
'INTR' request, raised by the device, is acknowledged by the up, by generating $\overline{INTA}$ pulses, up ~~sends~~ receives the dev adds int type no, on data bus during second $\overline{INTA}$ pulse (interrupt acknowledgement machine cycle)



* Level triggered (⎍), maintaining '1' (high level) generates an interrupt)

② Hardware interrupts (internal) OR referred as "Exceptions".

These are generated by internal hardware, on occurance of some error, i.e exceptional conditions. 8086 has two types of exceptions,



(i) Divide by zero (TYPE NO-0)

ex: MOV AX, 1000
MOV BL, 0
DIV BL

(even when the quotient exceeds, its destination reg size, then also this int occurs)

(ii) Overflow error (TYPE NO-4)

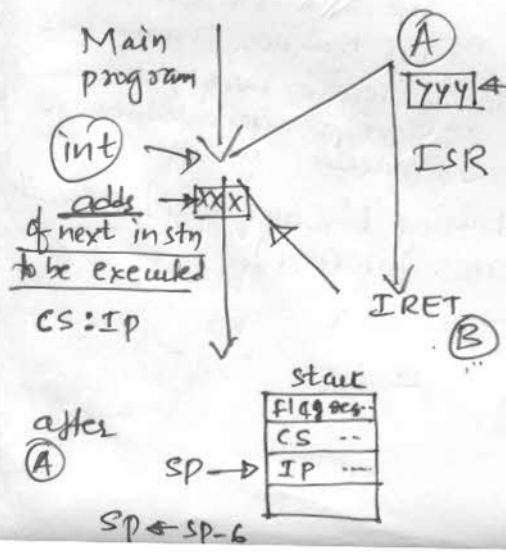
ex: MOV AL, -128
MOV BL, -80
ADD AL, BL

When signed arithmetic is performed, and the destination (answer) size exceeds its capacity, overflow occurs (i.e, in the above example, permitted answer size is, -128 to +127).

③ Software Interrupts: These are software instructions, like INT n, on execution of this instruction, ISR related to "type N" is executed. This feature is used to implement DOS & BIOS services, ex: INT 21H, INT 16H etc

Interrupt response

On occurance of interrupt, following sequence of events takes place as indicated by the following diagram,



(A) - on occurance of int,

* Flag reg (16 bit) is pushed to stack - ①
* Segment reg (CS) is pushed to stack - ④
* IP reg (offset adds) is pushed to stack ⑤
* IE flag is cleared (INTR is ~~enabled~~ disabled) - ②
* Trap flag is cleared (single stepping is disabled) ③

(SP is dected by 6)

NOW, adds of ISR (i.e New value of CS & IP) is fetched from IVTable (explained in next section) and loaded to CS & IP registers and execution of ISR starts

(B), On execution of IRET, (last instruction of any ISR), following events occurs,

* IP contents are popped (i.e loaded to IP reg)
* CS contents are popped (i.e loaded to CS reg)
* Flag registers contents are popped (Since flag register is popped, old values of TF & IE flags are restored)

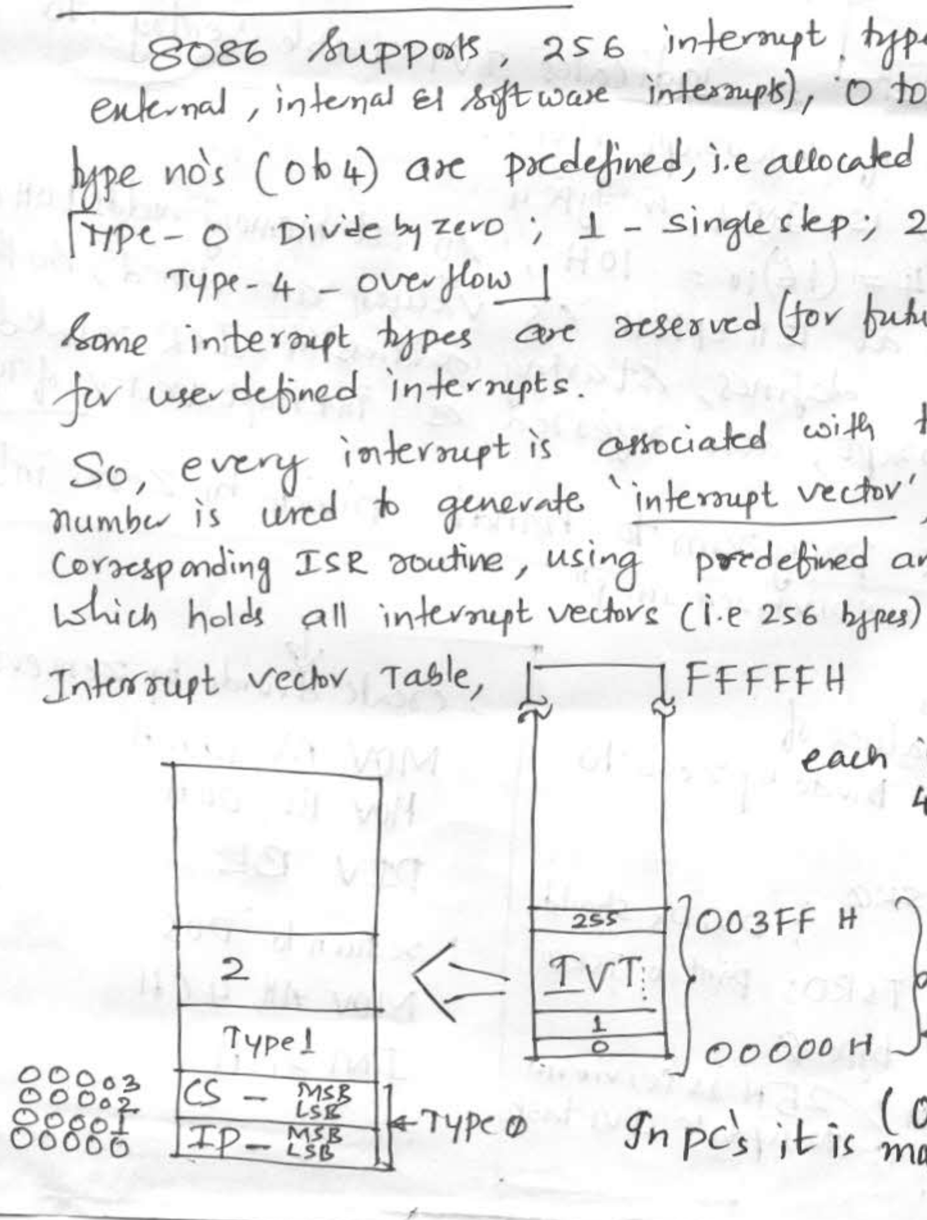
effectively SP is incremented by 6 (SP ← SP+6), and the control is transferred to the main program, by loading CS & IP registers with the return address.

(Note: Difference between procedure & ISR: In procedure either RET (near procedures) or RETF (far procedures) is used to return to the main program (RET – SP is inc'ed by 2, (IP) is reloaded) RETF – SP is inc'ed by 4 ((CS & IP) are reloaded)
To invoke procedure Call is used, unlike in ISR's)

Interrupt Vector Table

8086 supports, 256 interrupt types (which are allocated to external, internal & software interrupts), 0 to 255. Among these, some type no's (0 to 4) are predefined, i.e allocated to perticular type of interrupts, [Type – 0 Divide by zero ; 1 – single step ; 2 – NMI ; 3 – Breakpoint ; Type – 4 – overflow]
Some interrupt types are reserved (for future processors), some are meant for user defined interrupts.

So, every interrupt is associated with type number, and this type number is used to generate 'interrupt vector', i.e starting address of corresponding ISR routine, using predefined and preallocated memory table, which holds all interrupt vectors (i.e 256 types). This table is referred as Interrupt vector Table,



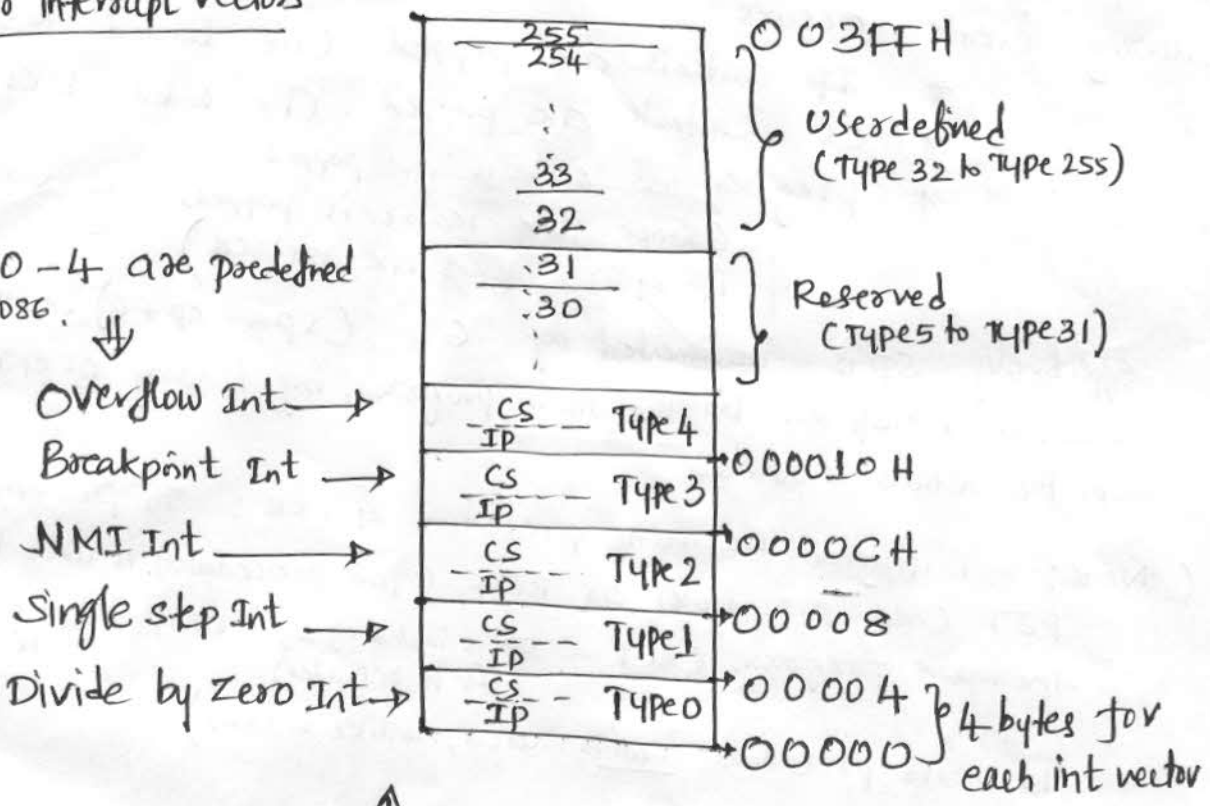
each interrupt vector, requires 4 bytes (2 for CS, 2 for IP)
so total size of IVT =

$256 \times 4 = 1024$ bytes

← Memory map of IVTable (00000 to 003FFH)

In pc's it is mapped to RAM area.

IVT table allocation for interrupt vectors



Int n

$n \times 4 =$ [XXXXX], indicates IVT table entry, to fetch CS & IP values of interrupt vector.

Ex: overflow int, is INT 4 or type 4

$4 \times 4 = (16)_{10} = 10H$, so at memory adds 10H & 11H, IP is stored, at 12H & 13H CS values are stored, so these four bytes CS:IP defines, starting address of ISR related to overflow interrupt, also referred as interrupt vector of type 4

① Write an ISR program to handle Divide by zero interpt

```
.DATA
MSG  DB  10,13,"DIV by zero INT $"
.CODE
; load CS & IP values
; i.e seg & offset values of
; new ISR to handle Divide by zero to
;     IVT table
MOV  AX, seg ISR0
MOV  DS, AX          ; DS:DX should
MOV  DX, offset ISR0; point to ISR0
MOV  AL, 0 ; type 0
MOV  AH, 25H ; 25H is service no
INT  21H     ; to update IVT table

; create divide by zero error
MOV AX, 1000H
MOV BL, 00H
DIV BL
; return to DOS
MOV AH, 4CH
INT 21H
```

```
ISR0  PROC
  ; set DS to data segment to access data
   MOV AX, @DATA
   MOV DS, AX
   LEA DX, offset MSG
   MOV AH, 09H ; display msg, "Divide by zero"
   INT 21H
  ; return to main prog
   IRET
ISR0  ENDP

END
```

② Write ISR program to handle overflow error?

Note: Overflow is of "type 4" interrupt, on occurance of overflow, interrupt is automatically not generated, unlike divide by zero. User has to implement this, by explicitly using instruction, INTO, it generates an interrupt only on occurance of overflow error (i.e overflow flag == 1)

```
.CODE
; set up / update IVT table for entry (Type No 4)
   MOV AX, seg ISR4
   MOV DS, AX
   MOV DX, offset ISR4      ] seg & IP adds of new ISR, ISR4.
   MOV AL, 4 ; type no. 4
   MOV AH, 25H ; function to updat IVT
   INT 21H
; Create overflow error
   MOV AL, -128
   MOV BL, -21
   ADD AL, BL
   INTO ; since, in this case, overflow flag is set ISR4 is
        :   executed
; return to DOS
   MOV AH, 4CH
   INT 21H                  (ISR4 & .DATA contents are
    :                        same as previous program).
END
```