8086 INTERRUPTS

- An interrupt is a special condition that arises during the working of a µP.
- The µP services it by executing a subroutine called Interrupt Service Routine (ISR).
- There are 3 sources of interrupts for 8086:

External Signal (Hardware Interrupts):

These interrupts occur as signals on the external pins of the µP.
8086 has two pins to accept hardware interrupts, NMI and INTR.

Special instructions (Software Interrupts):

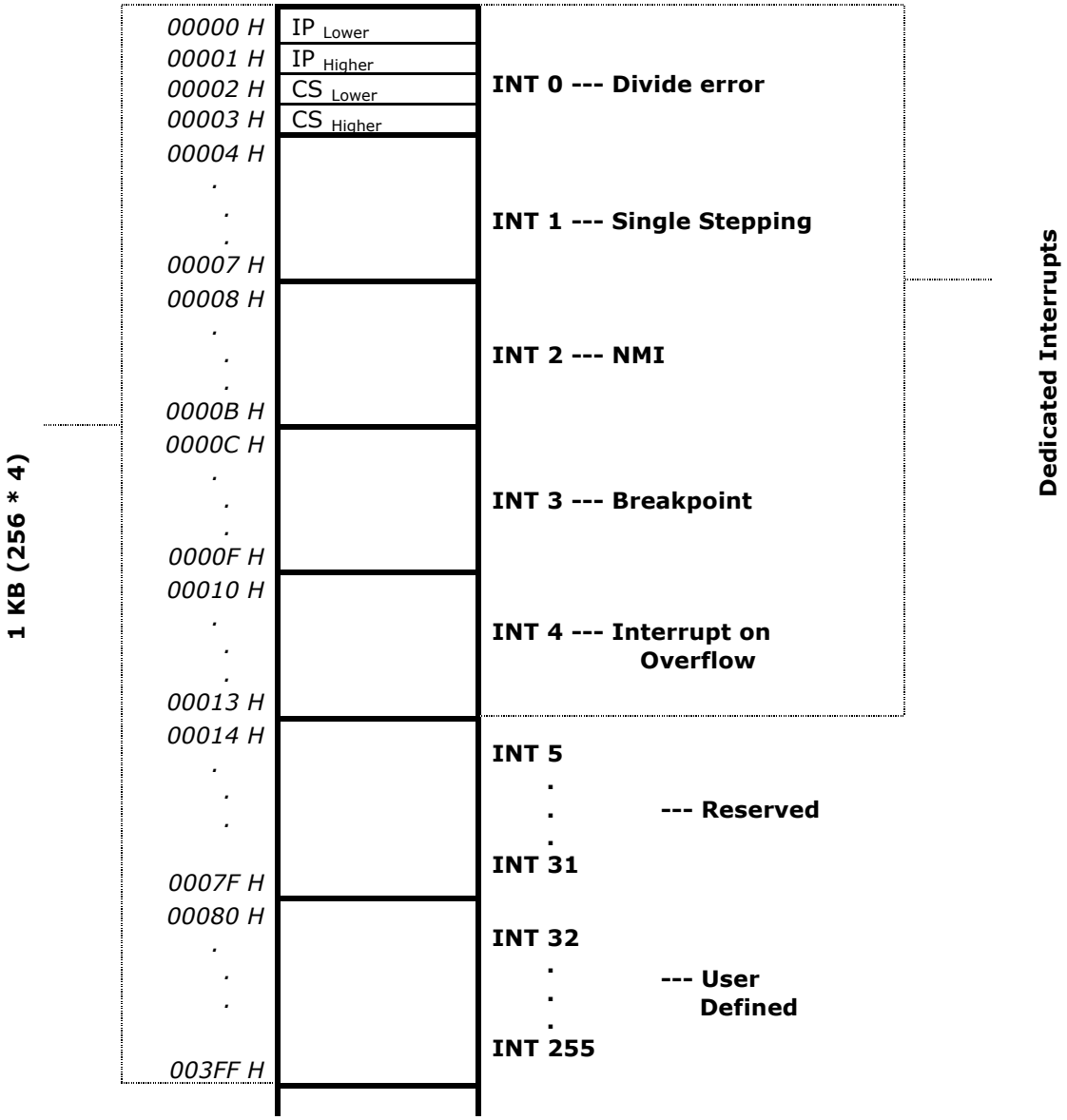
These interrupts are caused by writing the software interrupt instruction INTn where "n" can be any value from 0 to 255 (00H to FFH).
Hence all 256 interrupts can be invoked by software.

Condition Produced by the Program (Internally Generated Interrupts):

8086 is interrupted when some special conditions occur while executing certain instructions in the program.
Eg: **An error in division** automatically causes the INT 0 interrupt.

INTERRUPT VECTOR TABLE (IVT) {10M --- IMPORTANT }

The **IVT contains ISR address** for the 256 interrupts.
Each ISR **address** is stored as **CS and IP**.
As each ISR address is of 4 bytes (2–CS and 2-IP), each ISR address requires 4 locations to be stored.
There are **256 interrupts**: INT 0 … INT 255 ∴ the **total size of** the **IVT** is 256 x 4 **= 1KB**.
The first 1KB of memory, address 00000 H … 003FF H, are reserved for the IVT.
Whenever an interrupt **INT N occurs**, **µP does N x 4 to get** values of **IP and CS from the IVT and hence perform the ISR**.


1 KB (256 * 4)
00000 H
00001 H
00002 H
00003 H
IP Lower
IP Higher
CS Lower
CS Higher
INT 0 --- Divide error
00004 H
00007 H
INT 1 --- Single Stepping
00008 H
0000B H
INT 2 --- NMI
0000C H
0000F H
INT 3 --- Breakpoint
00010 H
00013 H
INT 4 --- Interrupt on Overflow
00014 H
0007F H
INT 5
INT 31
--- Reserved
00080 H
003FF H
INT 32
INT 255
--- User Defined
Dedicated Interrupts

DEDICATED INTERRUPTS (INT 0 ... INT 4)

1) INT O (Divide Error)
This interrupt occurs whenever there is **division error**
i.e. when the result of a division is too large to be stored.
This condition normally occurs when the divisor is very small as compared to the dividend or the divisor is zero. #Refer example from Bharat Sir's lecture notes...
Its ISR address is stored at location 0 x 4 = 00000H in the IVT.

2) INT 1 (Single Step)
The µP executes this interrupt **after every instruction if the TF is set**.
It puts µP in **Single Stepping** Mode i.e. the µP pauses after executing every instruction.
This is very useful during **debugging**. #Refer example from Bharat Sir's lecture notes...
Its ISR generally displays contents of all registers.
Its ISR address is stored at location 1 x 4 = 00004H in the IVT.

3) INT 2 (Non Maskable Interrupt)
The µP executes this ISR in **response to** an interrupt on the **NMI** line.
Its ISR address is stored at location 2 x 4 = 00008H in the IVT.

4) INT 3 (Breakpoint Interrupt)
This interrupt is used to cause **Breakpoints** in the program.
It is caused by writing the instruction INT 03H or simply INT.
It is useful in **debugging large programs** where Single Stepping is inefficient.
Its ISR is used to **display** the **contents of all registers** on the screen.
Its ISR address is stored at location 3 x 4 = 0000CH in the IVT.

5) INT 4 (Overflow Interrupt)
This interrupt occurs if the **Overflow Flag is set AND** the µP executes the **INTO** instruction (Interrupt on overflow). #Show example from Bharat Sir's lecture notes...
It is used to detect overflow error in **signed arithmetic** operations.
Its ISR address is stored at location 4 x 4 = 00010H in the IVT.

Please Note: INT 0 … INT 4 are called as dedicated interrupts as these interrupts are dedicated for the above-mentioned special conditions.

RESERVED INTERRUPTS

INT 5 … INT 31

These levels are **reserved** by INTEL to be used in higher processors like 80386, Pentium etc. They are **not available** to the user.

User defined Interrupts

INT 32 … INT 255

These are **user defined**, **software** interrupts.
ISRs for these interrupts are written by the users to service various user defined conditions.
These interrupts are invoked by writing the instruction INT n.
Its ISR address is obtained by the µP from location n x 4 in the IVT.

HARDWARE INTERRUPTS

1) NMI (Non Maskable Interrupt)

This is a **non-maskable**, **edge** triggered, **high priority** interrupt.
On receiving an interrupt on NMI line, the µP executes **INT 2.**
µP obtains the ISR address from location 2 x 4 = 00008H from the IVT.
It reads 4 locations starting from this address to get the values for IP and CS, to execute the ISR.

2) INTR

This is a **maskable**, **level** triggered, **low priority** interrupt.

On receiving an interrupt on INTR line, the µP executes **2** $\overline{\textbf{INTA}}$ pulses.

1st $\overline{\textbf{INTA}}$ pulse --- the interrupting device **calculates** (prepares to send) the **vector number**.

2nd $\overline{\textbf{INTA}}$ pulse --- the interrupting device **sends** the **vector number "N"** to the µP.

Now µP multiplies N x 4 and goes to the corresponding location in the IVT to obtain the ISR address.
INTR is a maskable interrupt.
It is masked by making IF = 0 by software through **CLI** instruction.
It is unmasked by making IF = 1 by software through **STI** instruction.

Response to any interrupt --- INT N

i) The µP will **PUSH Flag** register into the Stack.
SS:[SP–1], SS:[SP-2] ← Flag
SP ← SP – 2

ii) **Clear IF and TF** in the Flag register and thus disables INTR interrupt.
IF ← 0, TF ← 0

iii) **PUSH CS** into the Stack.
SS:[SP–1], SS:[SP-2] ← CS
SP ← SP – 2

iv) **PUSH IP** into the Stack.
SS:[SP–1], SS:[SP-2] ← IP
SP ← SP – 2

v) **Load new IP** from the IVT
IP ← [N x 4], [N x 4 + 1]

vi) **Load new CS** from the IVT
IP ← [N x 4 + 2], [N x 4 + 3]

Since CS and IP get new values, control shifts to the address of the ISR and the ISR thus begins. At the end of the ISR the µP encounters the IRET instruction and returns to the main program in the following steps.

Response to IRET instruction

i) The µP will **restore IP from the stack**
IP ← SS:[SP], SS:[SP+1]
SP ← SP + 2

ii) The µP will **restore CS from the stack**
CS ← SS:[SP], SS:[SP+1]
SP ← SP + 2

iii) The µP will **restore FLAG register from the stack**
Flag ← SS:[SP], SS:[SP+1]
SP ← SP + 2

Interrupt Priorities

Interrupt	Priority	
	(Simultaneous occurrence)	**(To interrupt another ISR)**
Divide Error, INT n, INTO	1 (**Highest**)	Can interrupt any ISR
NMI	2	
INTR	3	Cannot interrupt an ISR (IF, TF ← 0)
Single Stepping	4(**Lowest**)	

Priority in 8086 interrupts is of two types:

1. Simultaneous Occurrence:

When more than one interrupts occur simultaneously then, **all s/w interrupts except single stepping**, get the **highest priority**.
This is followed by **NMI**. Next is **INTR**. Finally, the **lowest priority** is of the **single stepping** interrupt.

Eg: Assume the µP is executing a **DIV** instruction that causes a **division error** and **simultaneously INTR** occurs.
Here **INT 0** (Division error) will be **serviced first** i.e. its ISR will be executed, as it has higher priority, and **then INTR** will be **serviced.** #Please refer Bharat Sir's Lecture Notes for this ...

2. Ability to interrupt another ISR:

Since software interrupts (**INT N**) are **non-maskable**, they **can interrupt** any ISR.
NMI is also **non-maskable** hence it **can** also **interrupt** any ISR.
But **INTR** and **Single stepping cannot interrupt** another ISR **as** both are **disabled** before µP enters an ISR by **IF ← 0 and TF ← 0**.

Eg: Assume the µP executes DIV instruction that causes a **division error**. So µP gets the **INT 0** interrupt and now **µP enters the ISR for INT 0**. **During** the execution of **this ISR, NMI and INTR occur**.
Here **µP will branch out** from the ISR of INT 0 and **service NMI** (as **NMI is non-maskable**).
After completing the ISR of NMI **µP** will **return to** the **ISR for INT 0**.
INTR is **still pending but** the **µP will not service INTR** during the ISR of INT 0 (as **IF ← 0**).
µP will first **finish** the **INT 0 ISR** and **only then service INTR**.
Thus INTR and Single stepping cannot interrupt an existing ISR.