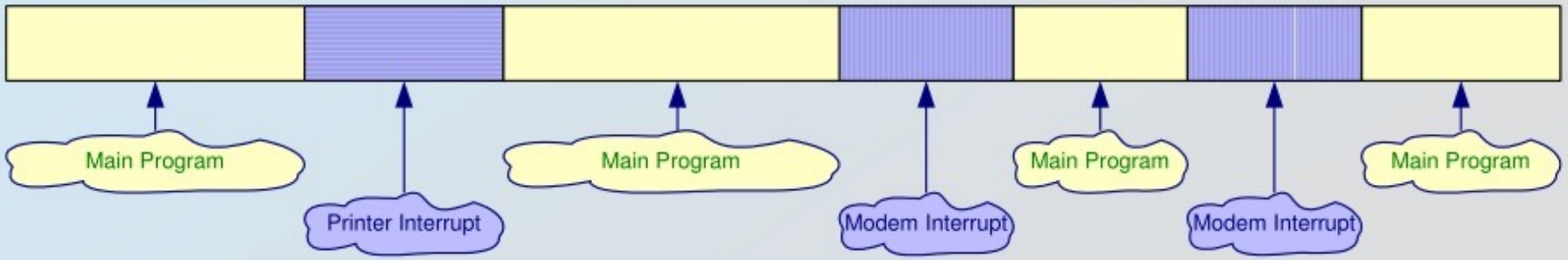
INTERRUPTS
&
INTERRUPT SERVICE ROUTINE(ISR)

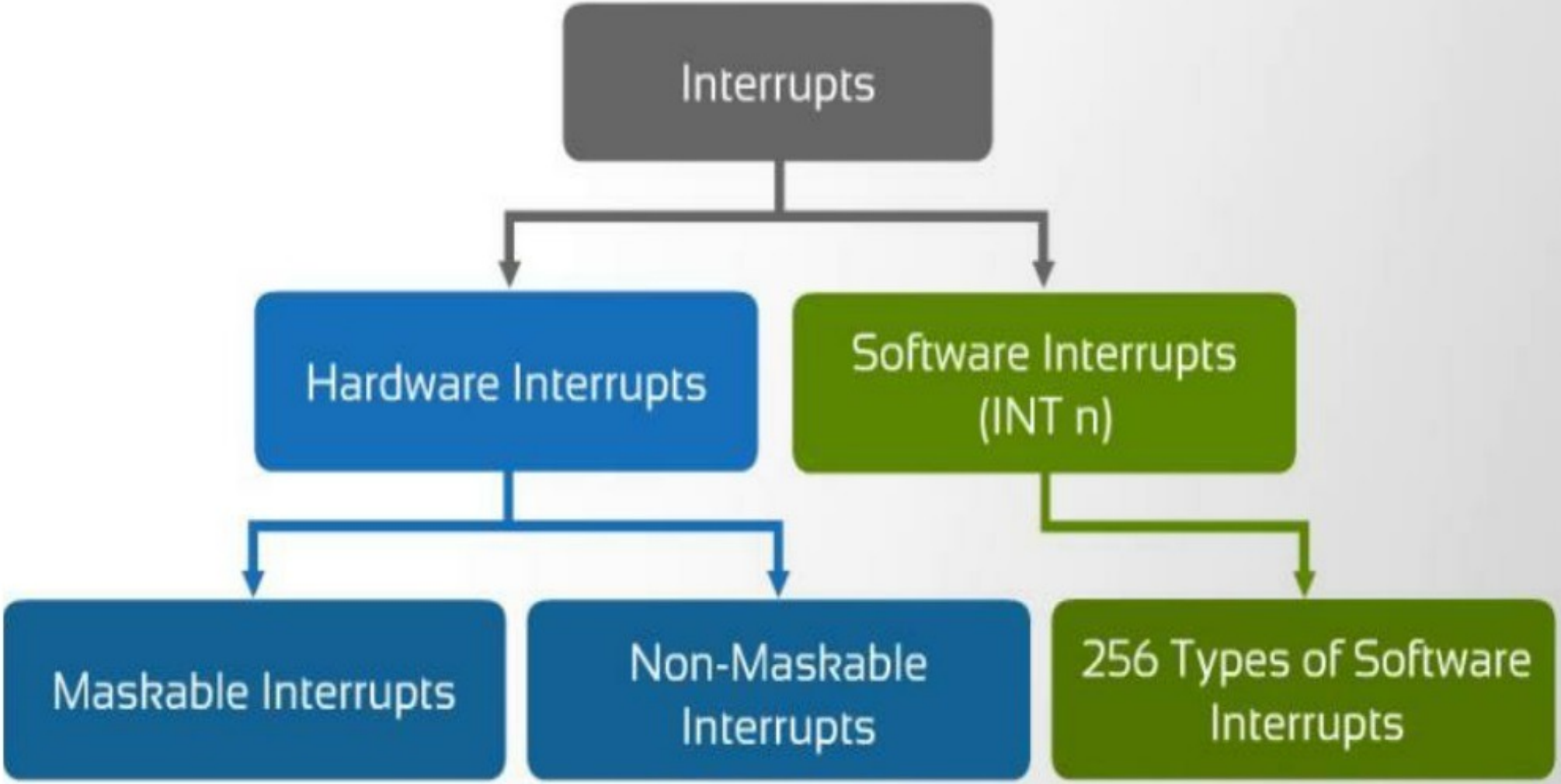
The Purpose of Interrupts

- **Interrupts are useful when interfacing I/O devices with low data-transfer rates, like a keyboard or a mouse, in which case polling the device wastes valuable processing time**
- **The peripheral interrupts the normal application execution, requesting to send or receive data.**
- **The processor jumps to a special program called *Interrupt Service Routine* to service the peripheral**
- **After the processor services the peripheral, the execution of the interrupted program continues.**



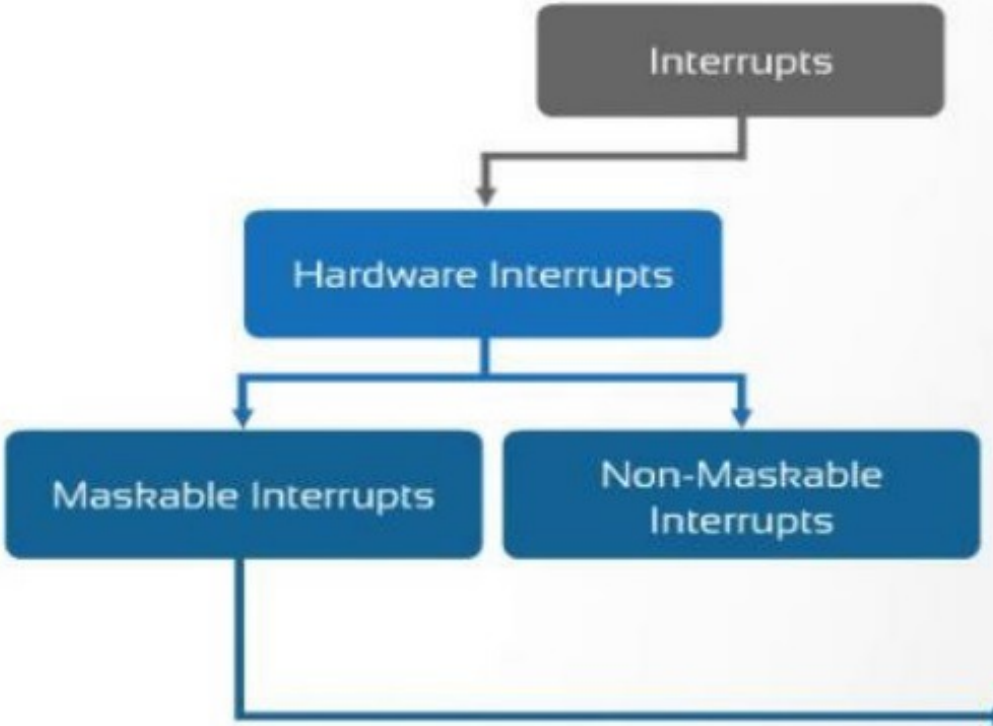
BASIC INTERRUPT TERMINOLOGY

- *Interrupt pins*: Set of pins used in hardware interrupts
- *Interrupt Service Routine (ISR) or Interrupt handler*: code used for handling a specific interrupt
- *Interrupt priority*: In systems with more than one interrupt inputs, some interrupts have a higher priority than other
  - They are serviced first if multiple interrupts are triggered simultaneously
- *Interrupt vector*: Code loaded on the bus by the interrupting device that contains the Address (segment and offset) of specific interrupt service routine
- *Interrupt Masking*: Ignoring (disabling) an interrupt
- *Non-Maskable Interrupt*: Interrupt that cannot be ignored (power-down)
- Software interrupts are machine instructions that amount to a call to the designated interrupt subroutine, usually identified by interrupt number. Ex: INT0 - INT255

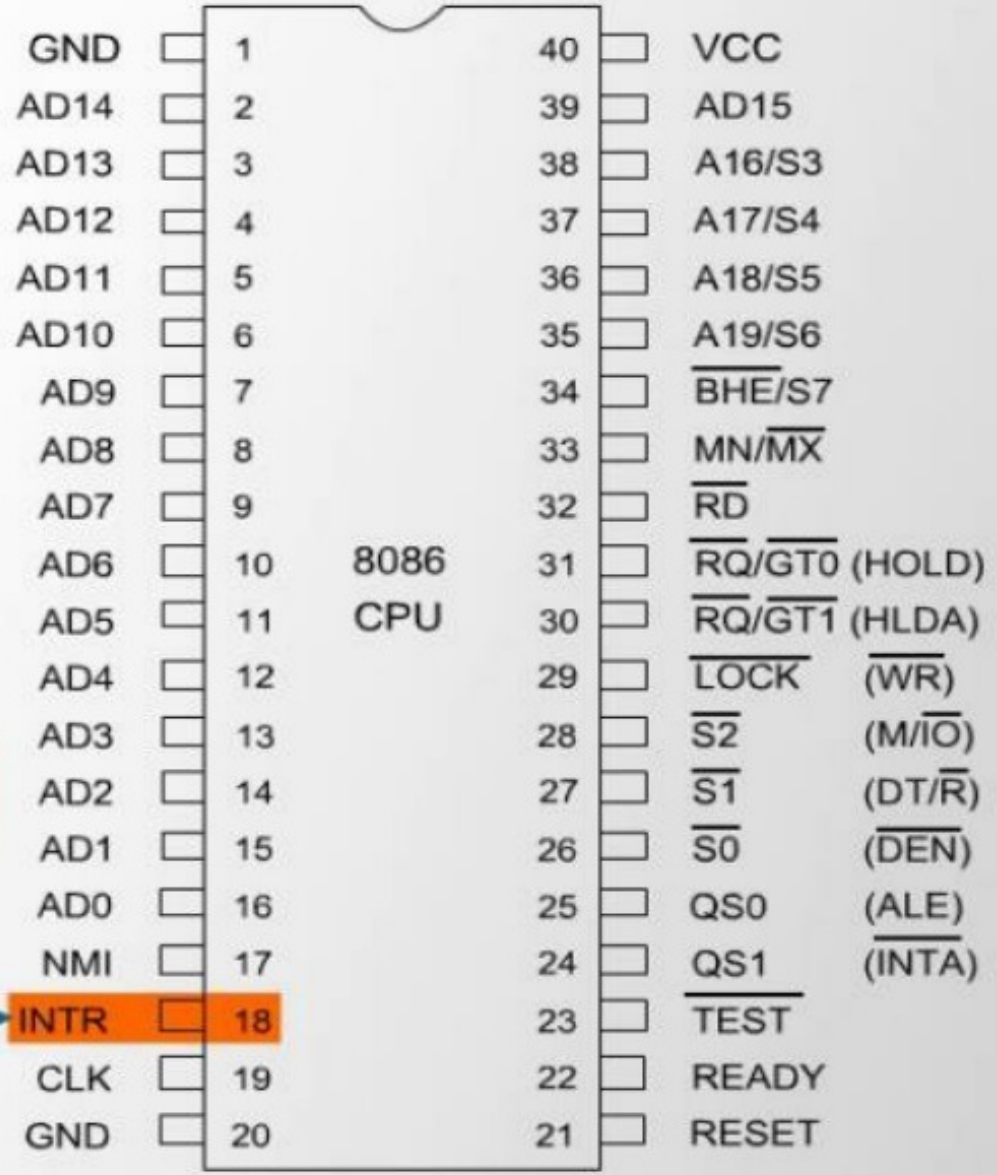

Interrupts
Hardware Interrupts
Software Interrupts (INT n)
Maskable Interrupts
Non-Maskable Interrupts
256 Types of Software Interrupts

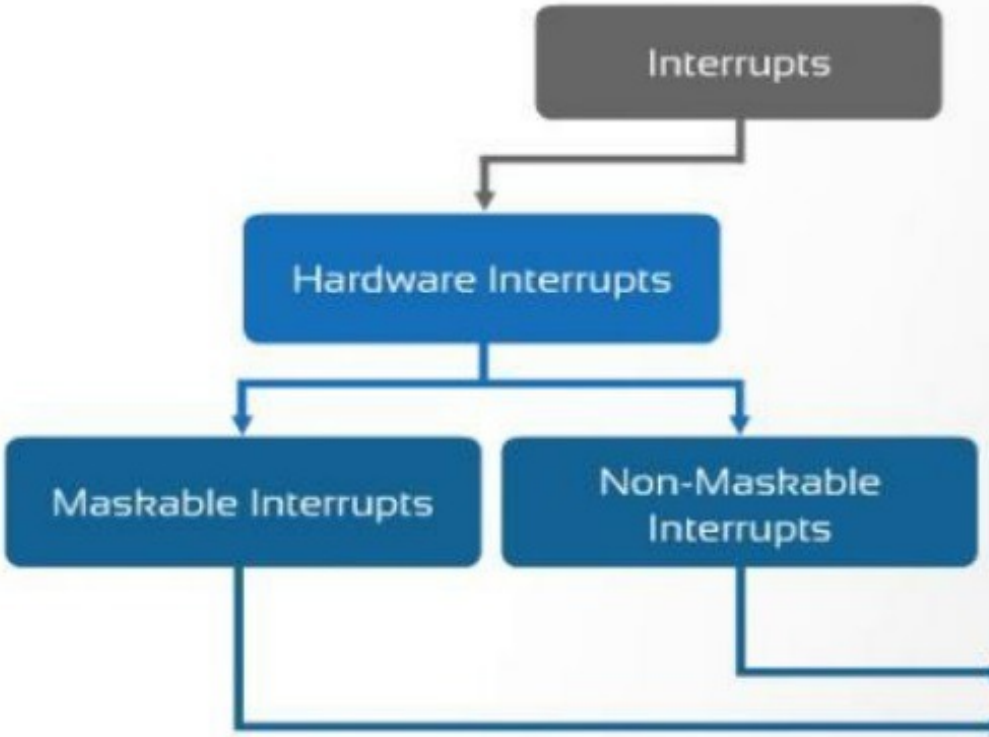
Hardware Interrupts

Used to handle external hardware peripherals, such as keyboards, mouse, hard disks, floppy disks, DVD drives, and printers

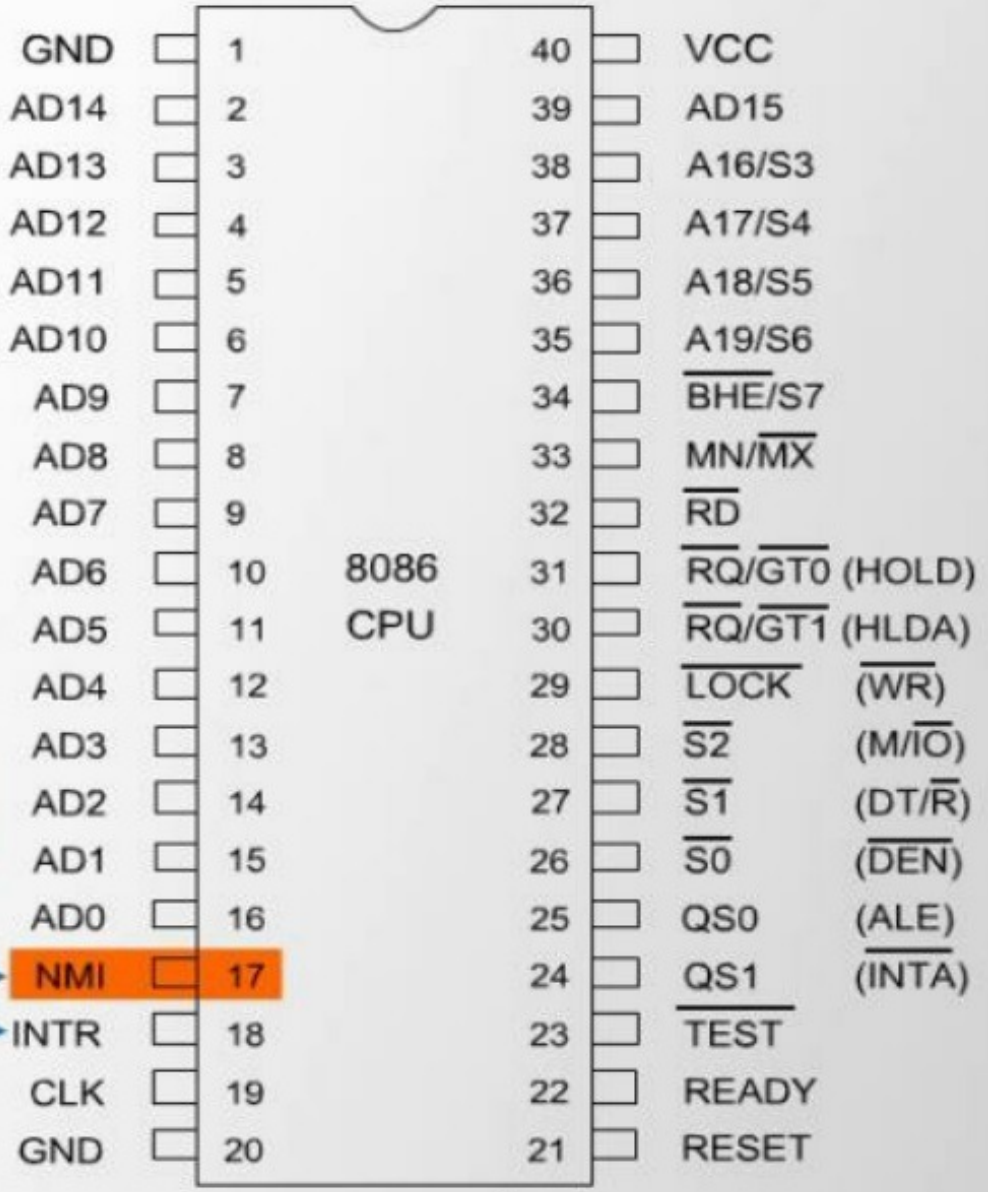

Interrupts
Hardware Interrupts
Maskable Interrupts
Non-Maskable Interrupts

8086 CPU

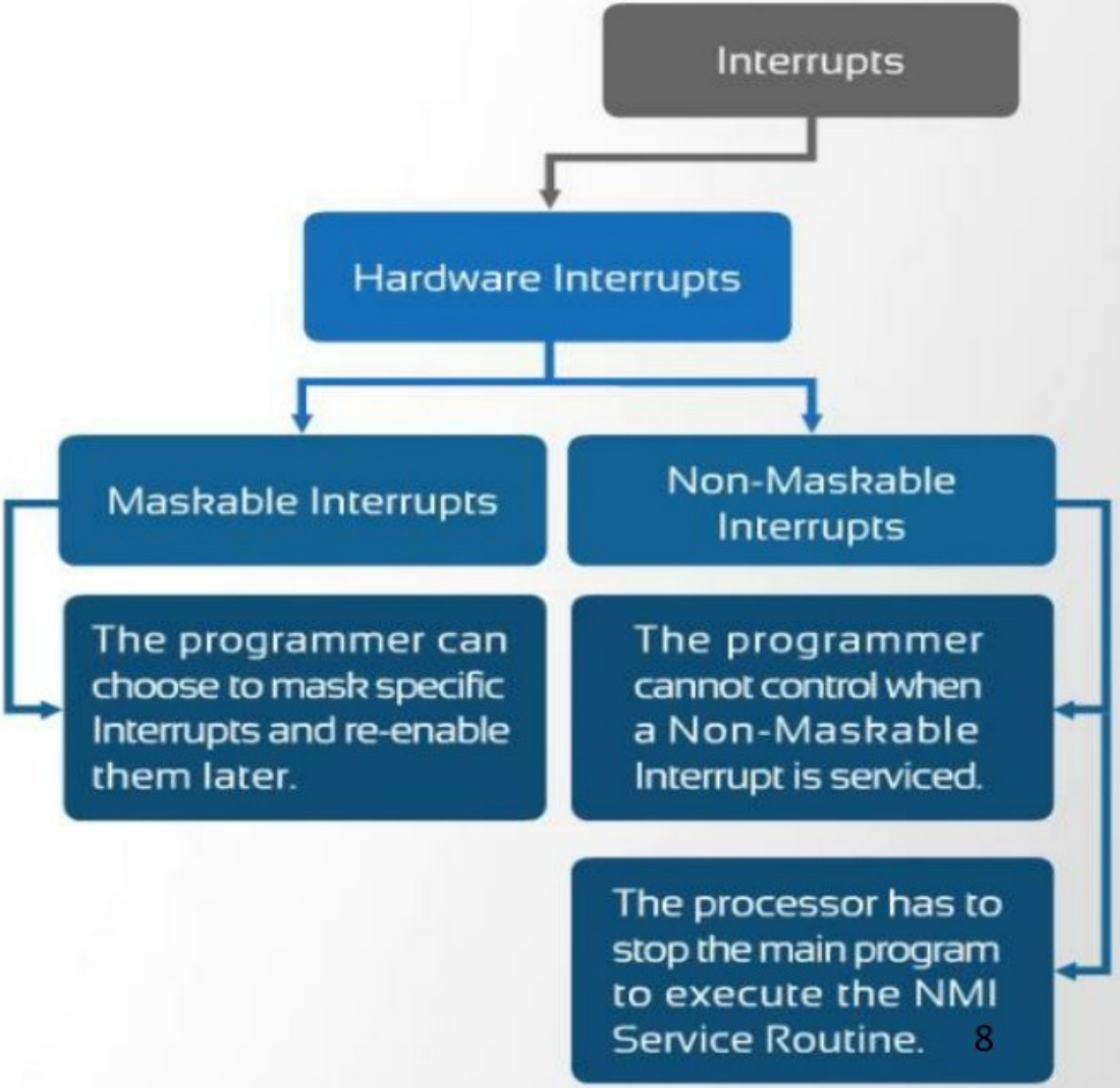

GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086 CPU
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 RQ/GT0 (HOLD)
30 RQ/GT1 (HLDA)
29 LOCK (WR)
28 S2 (M/IO)
27 S1 (DT/R)
26 S0 (DEN)
25 QS0 (ALE)
24 QS1 (INTA)
23 TEST
22 READY
21 RESET

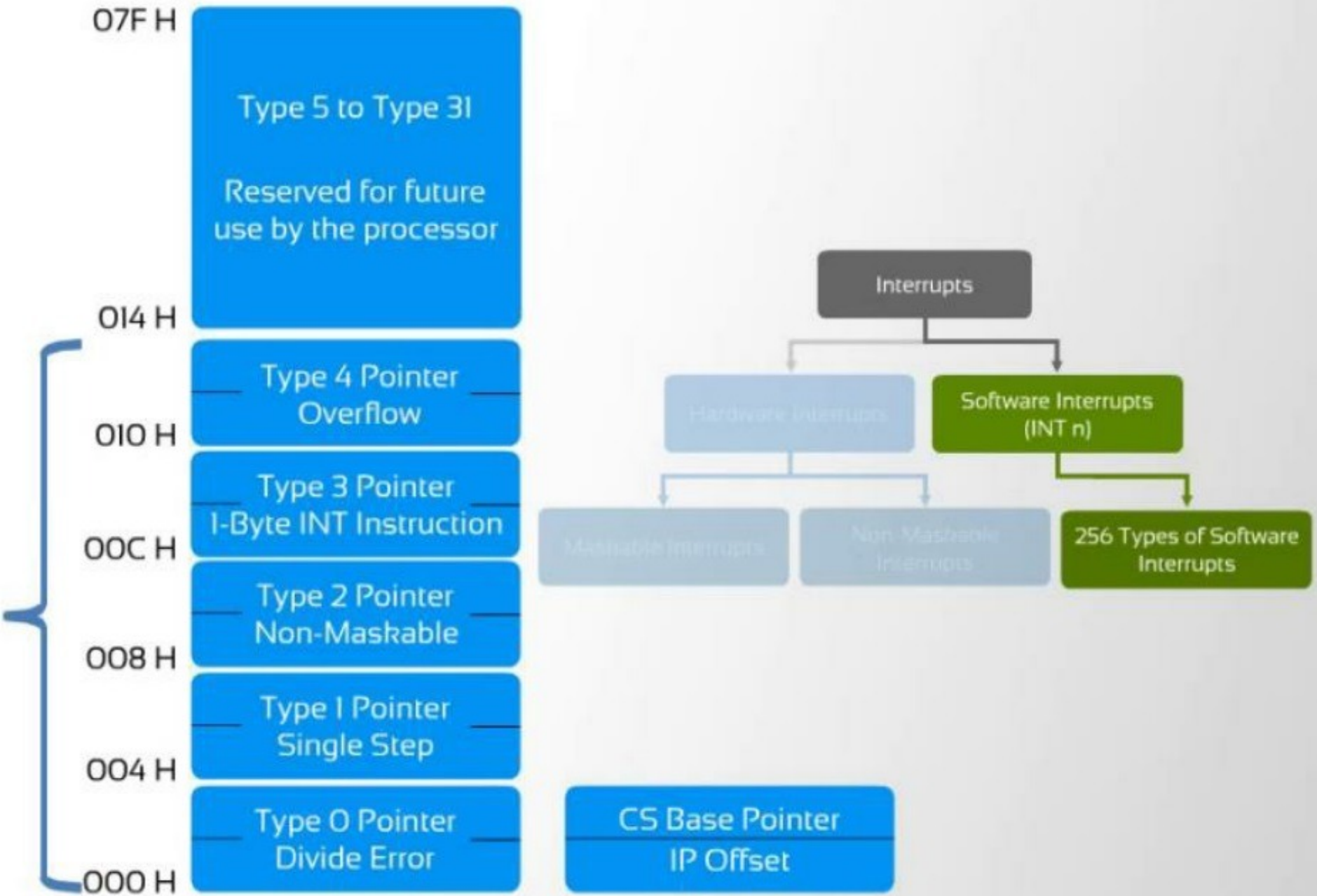

Interrupts
Hardware Interrupts
Maskable Interrupts
Non-Maskable Interrupts

8086 CPU


GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086 CPU
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 RQ/GT0 (HOLD)
30 RQ/GT1 (HLDA)
29 LOCK (WR)
28 S2 (M/IO)
27 S1 (DT/R)
26 S0 (DEN)
25 QS0 (ALE)
24 QS1 (INTA)
23 TEST
22 READY
21 RESET

Maskable Versus Non-Maskable Interrupts

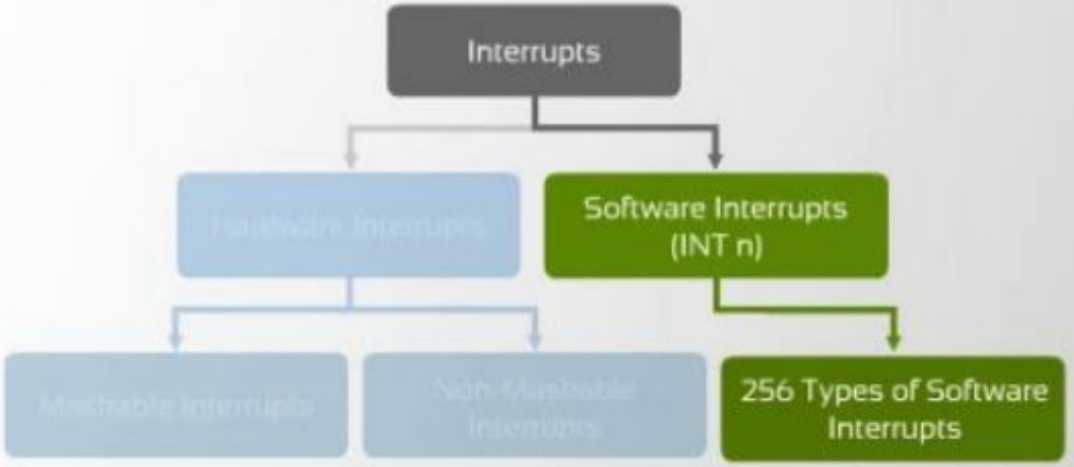



07F H
Type 5 to Type 31
Reserved for future use by the processor
014 H
Type 4 Pointer Overflow
010 H
Type 3 Pointer I-Byte INT Instruction
00C H
Type 2 Pointer Non-Maskable
008 H
Type 1 Pointer Single Step
004 H
Type 0 Pointer Divide Error
000 H
CS Base Pointer
IP Offset
Interrupts
Software Interrupts (INT n)
256 Types of Software Interrupts

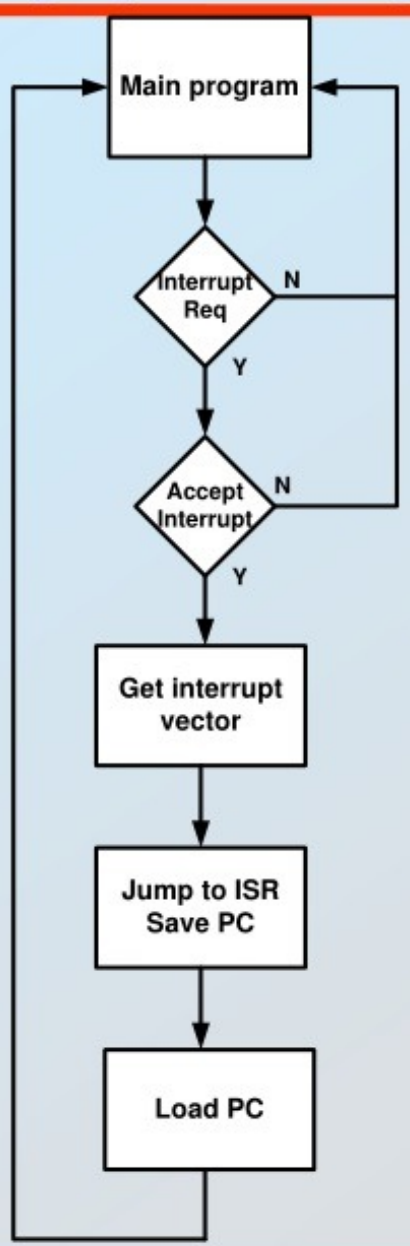
3FF H

Type 32 to Type 255
Free for User

080 H

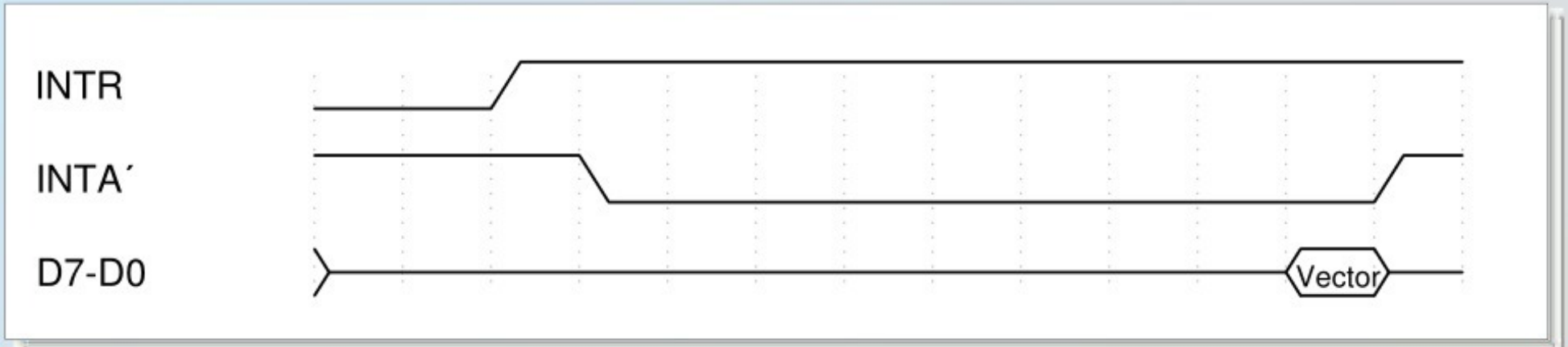


Interrupt processing flow



Hardware Interrupts – Interrupt pins and timing

- x86 Interrupt Pins
  - **INTR:** Interrupt Request. Activated by a peripheral device to interrupt the processor.
    - Level triggered. Activated with a logic 1.
  - **INTA:** Interrupt Acknowledge. Activated by the processor to inform the interrupting device the the interrupt request (INTR) is accepted.
    - Level triggered. Activated with a logic 0.
  - **NMI:** Non-Maskable Interrupt. Used for major system faults such as parity errors and power failures.
    - Edge triggered. Activated with a positive edge (0 to 1) transition.
    - Must remain at logic 1, until it is accepted by the processor.
    - Before the 0 to 1 transition, NMI must be at logic 0 for at least 2 clock cycles.
    - No need for interrupt acknowledgement.



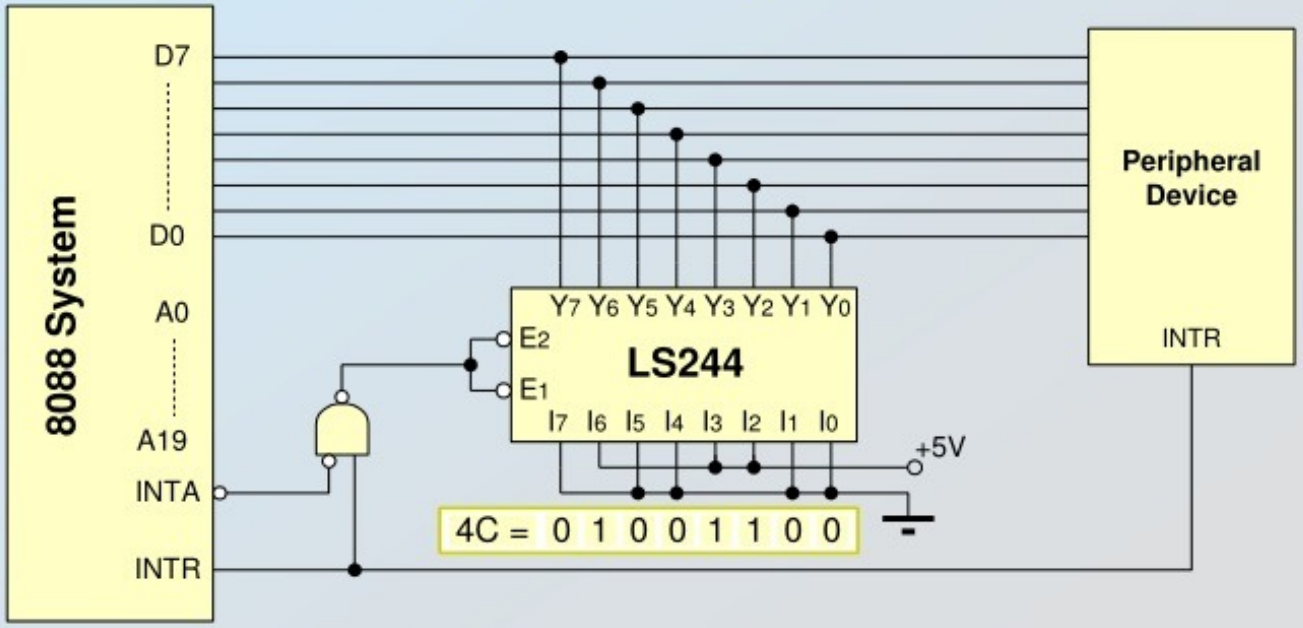
Interrupt Vectors

- **The *Interrupt Vector* contains the address of the interrupt service routine**
- **The *Interrupt Vector Table* is located in the first 1024 bytes of memory at address 000000H-0003FFH.**
- **It contains 256 different 4-byte interrupt vectors, grouped in 18 types**
  - **000H: Type 0 (Divide error)**
  - **004H: Type 1 (Single-step)**
  - **008H: Type 2 (NMI)**
  - **00CH: Type 3 (1-byte breakpoint)**
  - **010H: Type 4 (Overflow)**
  - **014H: Type 5 (BOUND)**
  - **018H: Type 6 (Undefined opcode)**
  - **01CH: Type 7 (Coprocessor not available)**
  - **020H: Type 8 (Double fault)**
  - **024H: Type 9 (Coprocessor segment overrun)**
  - **028H: Type 10 (Invlid task state segment)**
  - **02CH: Type 11 (Segment not present)**

- **030H: Type 12 (Stack segment overrun)**
- **034H: Type 13 (General protection)**
- **038H: Type 14 (Page fault)**
- **03CH: Type 15 (Unassigned)**
- **040H: Type 16 (Coprocessor error)**
- **044H-07CH: Type 14-31 (Reserved)**
- **080H: Type 32-255 (User)**

Interrupt Vector - Example

- **Draw a circuit diagram to show how a device with interrupt vector 4CH can be connected on an 8088 microprocessor system.**
- **Answer:**
  - **The peripheral device activates the INTR line**
  - **The processor responds by activating the INTA signal**
  - **The NAND gate enables the 74LS244 octal buffer**
    - **the number 4CH appears on the data bus**
  - **The processor reads the data bus to get the interrupt vector**



Interrupt Vector Table – Real Mode (16-bit) Example

- Write a sequence of instructions that initialize vector 40H to point to the ISR "isr40".
- Answer: Address in table = 4 X 40H = 100H
- Set ds to 0 since the Interrupt Vector Table begins at 00000H
- Get the offset address of the ISR using the Offset directive
- and store it in the addresses 100H and 101H
- Get the segment address of the ISR using the Segment directive
- and store it in the addresses 102H and 103H

```
push    ax
push    ds          } Save registers in the stack
mov     ax,0
mov     ds,ax       } Set ds to 0 to point to the interrupt vector table
mov     ax,offset isr40
mov     [0100h],ax  } Get the offset address of the ISR and store it in the address 0100h (4X40h = 100h)
mov     ax,segment isr40
mov     [0102h],ax  } Get the segment address of the ISR and store it in the address 0102h
pop     ds
pop     ax          } Restore registers from the stack
```

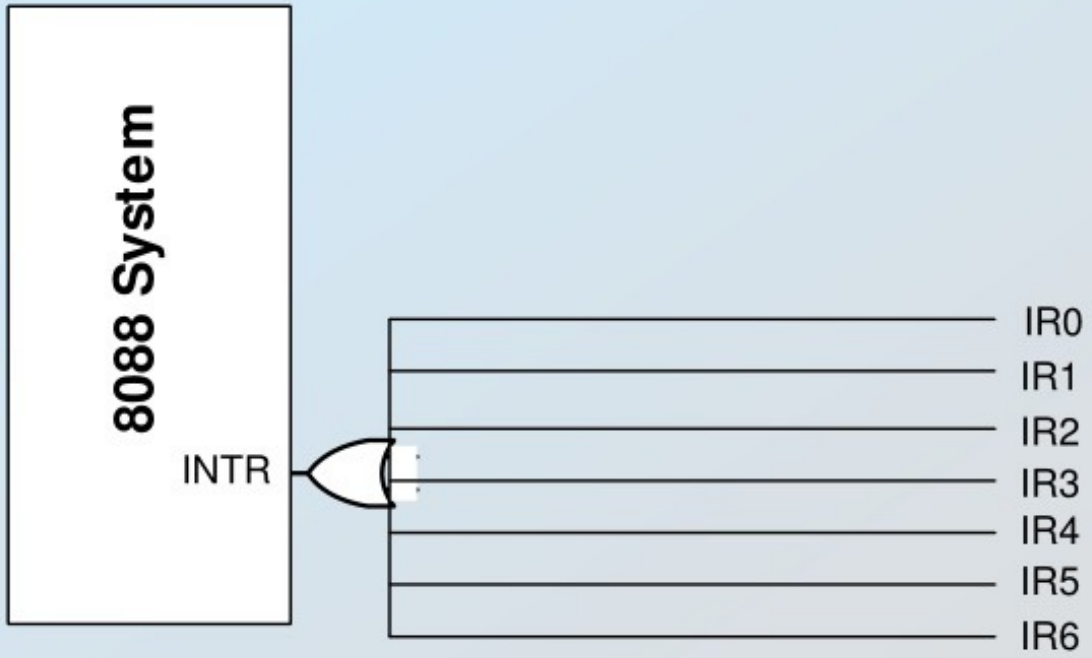
Interrupt Vector Table – Real Mode (16-bit) Example

- Write a sequence of instructions that initialize vector 40H to point to the ISR "isr40".
- Answer: Address in table = 4 X 40H = 100H
- Set ds to 0 since the Interrupt Vector Table begins at 00000H
- Get the offset address of the ISR using the Offset directive
- and store it in the addresses 100H and 101H
- Get the segment address of the ISR using the Segment directive
- and store it in the addresses 102H and 103H

```
push    ax
push    ds          } Save registers in the stack
mov     ax,0
mov     ds,ax       } Set ds to 0 to point to the interrupt vector table
mov     ax,offset isr40
mov     [0100h],ax  } Get the offset address of the ISR and store it in the address 0100h (4X40h = 100h)
mov     ax,segment isr40
mov     [0102h],ax  } Get the segment address of the ISR and store it in the address 0102h
pop     ds
pop     ax          } Restore registers from the stack
```

Daisy-Chained Interrupt

Each device is connected to the same interrupt request line, but there is only a single interrupt vector. The device that sent the request will respond.



Interrupt Masking

- The processor can inhibit certain types of interrupts by use of a special interrupt mask bit.
- This mask bit is part of the flags/condition code register, or a special interrupt register.
- If this bit is clear, and an interrupt request occurs on the Interrupt Request input, it is ignored.
- NMI cannot be masked

Interrupt Processing on the 8086 Microprocessor

- 1. External interface sends an interrupt signal, to the Interrupt Request (INTR) pin, (or an internal interrupt occurs.)
- 2. The CPU finishes the present instruction (for a hardware interrupt) and checks the INTR pin.
- If IF=0 the processor ignores the interrupt, else sends Interrupt Acknowledge (INTA) to hardware interface.
- 3. The interrupt type N is sent to the Central Processor Unit (CPU) via the Data bus from the hardware interface.
- 4. The contents of the flag registers are pushed onto the stack.
- 5. Both the interrupt (IF – FR bit 9) and (TF – FR bit 8) flags are cleared. This disables the INTR pin and the trap or single-step feature.
- 6. The contents of the code segment register (CS) are pushed onto the Stack.
- 7. The contents of the instruction pointer (IP) are pushed onto the Stack.
- 8. The interrupt vector contents are fetched, from (4 x N) and then placed into the IP and from (4 x N +2) into the CS so that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.
- 9. While returning from the interrupt-service routine by the Interrupt Return (IRET) instruction, the IP, CS and Flag registers are popped from the Stack and return to their state prior to the interrupt.

The Intel x86 Interrupt Software Instructions

- **All x86 processors provide the following instructions related to interrupts:**
  - **INT N: Interrupt. Run the ISR pointed by vector N.**
    - **INT 0 is reserved for the Divide Error**
    - **INT 1 is reserved for Single Step operation**
    - **INT 2 is reserved for the NMI pin**
    - **INT 3 is reserved for setting a Breakpoint**
    - **INT 4 is reserved for Overflow (Same as the INTO (Interrupt on overflow) instruction.**
  - **CLI: Clear Interrupt Flag. IF is set to 0, thus interrupts are disabled.**
  - **STI: Set Interrupt Flag. IF is set to 1, thus interrupts are enabled.**
  - **IRET: Return from interrupt. This is the last instruction in the ISR (Real Mode only). It pops from the stack the Flag register, the IP and the CS.**
    - **After returning from an ISR the interrupts are enabled, since the initial value of the flag register is poped from the stack.**
  - **IRETD: Return from interrupt. This is the last instruction in the ISR (Protected Mode only). It pops from the stack the Flag register, the IP and the CS.**

Interrupt Vectors

- **The *Interrupt Vector* contains the address of the interrupt service routine**
- **The *Interrupt Vector Table* is located in the first 1024 bytes of memory at address 000000H-0003FFH.**
- **It contains 256 different 4-byte interrupt vectors, grouped in 18 types**
  - **000H: Type 0 (Divide error)**
  - **004H: Type 1 (Single-step)**
  - **008H: Type 2 (NMI)**
  - **00CH: Type 3 (1-byte breakpoint)**
  - **010H: Type 4 (Overflow)**
  - **014H: Type 5 (BOUND)**
  - **018H: Type 6 (Undefined opcode)**
  - **01CH: Type 7 (Coprocessor not available)**
  - **020H: Type 8 (Double fault)**
  - **024H: Type 9 (Coprocessor segment overrun)**
  - **028H: Type 10 (Invlid task state segment)**
  - **02CH: Type 11 (Segment not present)**

- **030H: Type 12 (Stack segment overrun)**
- **034H: Type 13 (General protection)**
- **038H: Type 14 (Page fault)**
- **03CH: Type 15 (Unassigned)**
- **040H: Type 16 (Coprocessor error)**
- **044H-07CH: Type 14-31 (Reserved)**
- **080H: Type 32-255 (User)**

Interrupt Types

- **Type 0: Divide error – Division overflow or division by zero**
- **Type 1: Single step or Trap – After the execution of each instruction when trap flag set**
- **Type 2: NMI Hardware Interrupt – '1' in the NMI pin**
- **Type 3: One-byte Interrupt – INT3 instruction (used for breakpoints)**
- **Type 4: Overflow – INTO instruction with an overflow flag**
- **Type 5: BOUND – Register contents out-of-bounds**
- **Type 6: Invalid Opcode – Undefined opcode occurred in program**
- **Type 7: Coprocessor not available – MSW indicates a coprocessor**
- **Type 8: Double Fault – Two separate interrupts occur during the same instruction**
- **Type 9: Coprocessor Segment Overrun – Coprocessor call operand exceeds FFFFH**
- **Type 10: Invalid Task State Segment – TSS invalid (probably not initialized)**
- **Type 11: Segment not present – Descriptor P bit indicates segment not present or invalid**
- **Type 12: Stack Segment Overrun – Stack segment not present or exceeded**
- **Type 13: General Protection – Protection violation in 286 (general protection fault)**
- **Type 14: Page Fault – 80386 and above**
- **Type 16: Coprocessor Error – ERROR´ = '0' (80386 and above)**
- **Type 17: Alignment Check – Word/Doubleword data addressed at odd location (486 and above)**
- **Type 18: Machine Check – Memory Management interrupt (Pentium and above)**