Polling & Interrupts

- 2 techniques are available to service external devices:
  - **Polling**
  - **Interrupts**
- **Polling**: **CPU periodically polls** ALL external devices & takes action if required. if the frequency of polling is high – a significant overhead is incurred on a CPU
- If, however, the frequency of polling is low - events may be undetected - data lost due to latency etc.
-
- **Interrupts: external device** indicates request for attention by **sending a signal** via a control line it stops the current CPU activity and responds to the device that requested attention

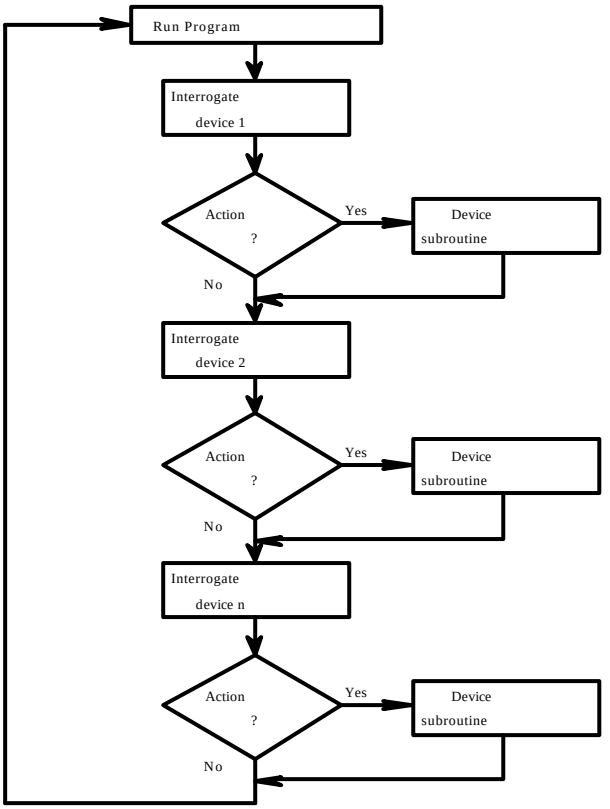
Interrupt service routine (ISR)

For every interrupt there must be a program associated with it. When an interrupt is invoked it si asked to run a program to perform a certain service. This program is commonly referred to as an interrupt service routine (ISR).

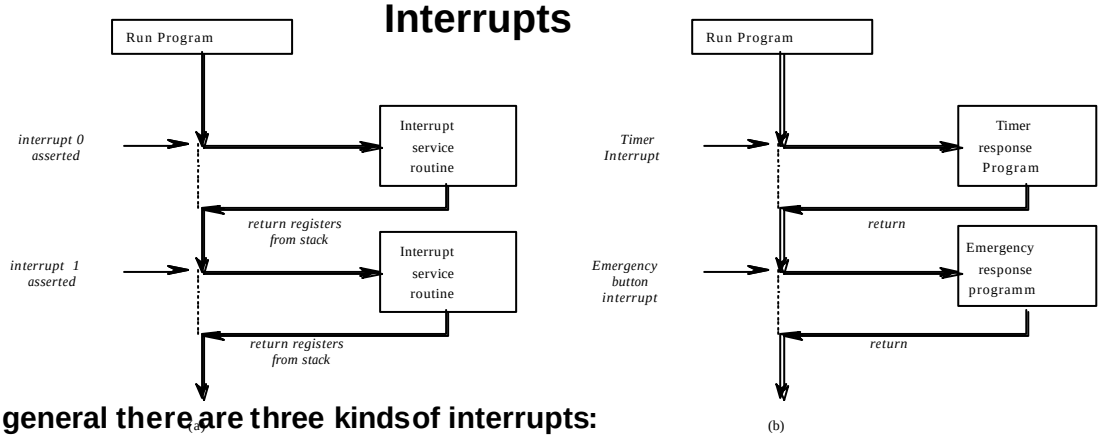
Polling

- must be written in the main program (by a programmer, in advance)

Interrupts provide a mechanism for quickly changing program environment. Transfer of program control is initiated by the occurrence of either an event internal to the MPU or an event in its external hardware. The section of program to which control is passed is
called the ***interrupt service routine.(ISR)***



Interrupts



In general there are three kinds of interrupts:

Software interrupts – initiated by the INT instruction

internal interrupts: – occur in response to error states in the processor

Hardware interrupts – originate in peripheral hardware

In the 8086/88 there are a total of 256 interrupts: INT 00, INT 01, …, INT FF

Regardless of source, they are handled the same

–Each interrupt has a unique interrupt number from 0 to 255. These are called interrupt vectors.

–For each interrupt vector, there is an entry in the interrupt vector table.

–The interrupt vector table is simply a jump table containing segment:offset addresses of procedures to handle each interrupt

–These procedures are called *interrupt handlers* or *interrupt service routines (ISR's)*

When an interrupt is executed, the microprocessor automatically saves the flag register (FR), the instruction pointer (IP), and the code segment register (CS) on the stack, and goes to a fixed memory location

Interrupts: a) Non-vectored (Microchip microcontrollers)

- uses fixed address in the memory at which the interrupt service routine (ISR) is located
- when an interrupt occurs – a Program Counter register is loaded with this address
- on completion, the last value in the Program Counter is restored and the control is handed over back to the CPU

b) Vectored (8086)

- Interrupt Vector:
- Is a pointer to a memory location whose content is a start address of the ISR
- The address of the vector itself is fixed but its content may be any address value
- It offers flexibility to a programmer
- *Vector* is a ***pointer*** (address) into *Interrupt Vector Table*, *IVT*
  - IVT is stored in memory from **`0000:0000`** to **`0000:03ffh`**

The address pointer table is located at the lowaddress end of the memory address space. It starts at 0000016 and ends at 003FE16. This represents the first 1 Kbytes of memory.

Sofware interrupts

If an ISR is called upon as result of the execution of an 80x86 instruction such as ‘’**INT nn”**, it is referred to as a software interrupt since it was invoked from saftware, not from external hardware. IN IBM PC many of the interrupts in this category are used by the MS DOS operating system and BIOS to perform essential tasks that evey computer must provide to the system and user.Within this group of interrups There are also same predefined functions associated with some of the interrupts. They are **‘’INT 00’’ (divide error), ‘’INT 01’’ (single step), ‘’INT 03’’ (breakpoint), and ‘’INT 04’’ (signed number overflow).**

So INT 00 to INT 04 hich have predefined functions, the rest of the interrupts from INT 05 to INT FF can be used to implement either software or hardware interrupts.

Interrupts and flag register:

In flag register, there are two bits that are associated with the interrup: IF (interrupt enable flag) and TF (trap flag). OF(overflow flag) canbe used by interrupt

Mnemonic	Meaning	Format	Operation	Flags affected
CLI	Clear interrupt flag	CLI	0 → (IF)	IF
STI	Set interrupt flag	STI	1 → (IF)	IF
INT n	Type n software interrupt	INT n	(Flags) → ((SP)-2) 0 → TF, IF (CS) → ((SP) – 4) (2+4xn) → (CS) (IP) → ((SP) – 6) (4xn) → (IP)	TF, IF
IRET	Interrupt return	IRET	((SP)) → (IP) ((SP)+2) →(CS) ((SP)+4) →(Flags) (SP) + 6 → (SP)	All
INTO	Interrupt on overflow	INTO	INT 4 steps	TF, IF

INT 01 (single step)

After execution of each instruction, the 8086/88 automatically jumps to physical location 00004 to fetch the 4 bytes for CS:IP of the interrupt service routine, whose job is, among other things, to dump the registers onto the screen.

Trap flag set or reset?

PUSHF	
POP	AX
AND	AX; FEFFH
PUSH	AX
POPF	

In 80x86 PCs, the memory location to which an interrupt goes is always four times the value of the interrupt number. For example, INT 03 will go to adress 0000CH (4 x 3) = 12 = 0CH). Table is a parital list of the interrupt vactor table.

For every interrupt there are allocated four bytes of memory in the interrupt vector table. Two bytes are for the IP and the other two are for the CS of the ISR. These four memory locations provide addresses of the interrupt service routine for which the interrupt was invoked. Thus the lowest 1024 bytes (256 x 4 = 1024) of memory space are set aside for the interrupt vector table and must not be used for any other function.

INT NUMBER	Physical Adress	Logical Adress
INT 00	00000	0000:0000
INT 01	00004	0000:0004
INT 02	00008	0000:0008
INT 03	0000C	0000:000C
INT 04	00010	0000:0010
INT 05	00014	0000:0014
…	…	…
INT FF	003FC	0000:03FC

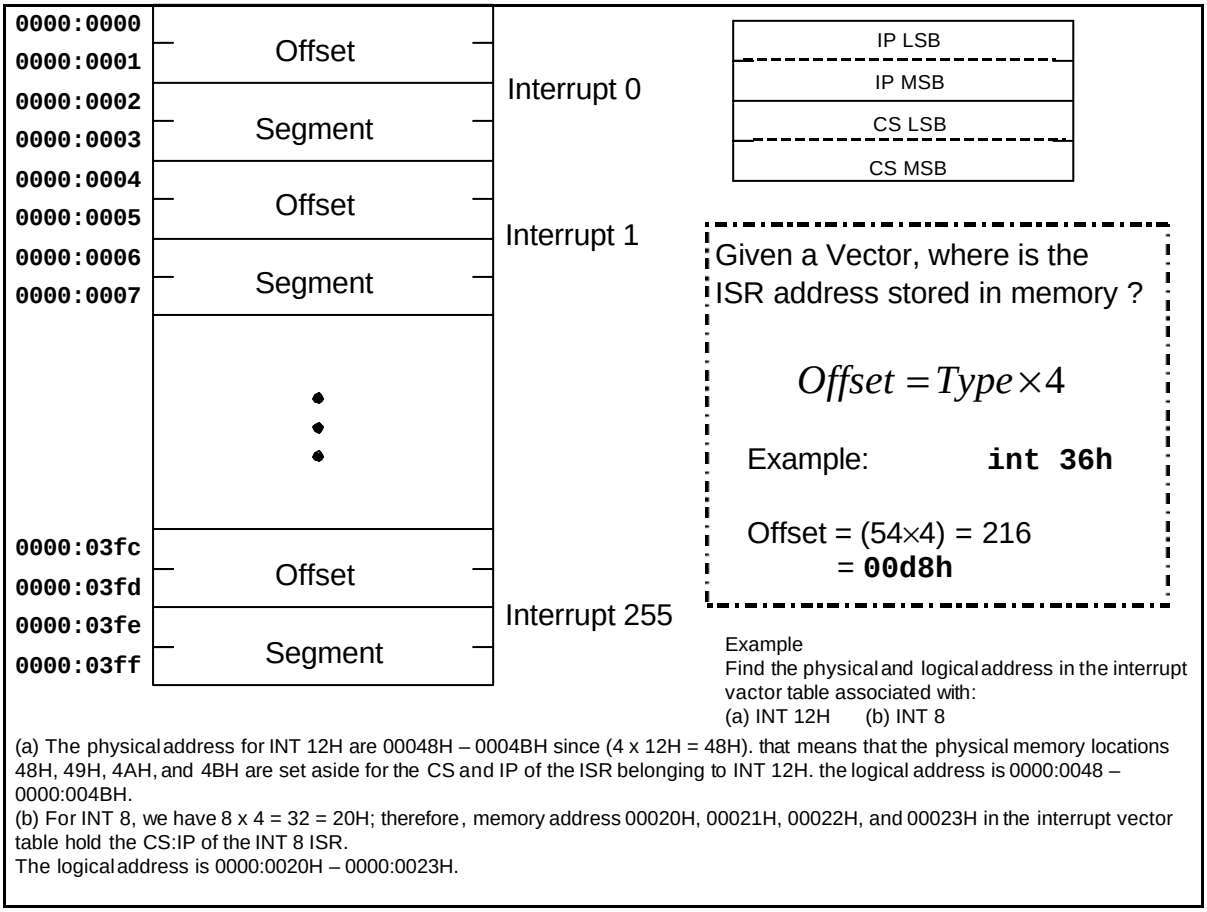
Regardless of source, they are handled the same

Each interrupt has a unique interrupt number from 0 to 255. These are called interrupt vectors.

For each interrupt vector, there is an entry in the interrupt vector table.

The interrupt vector table is simply a jump table containing segment:offset addresses of procedures to handle each interrupt

These procedures are called *interrupt handlers* or *interrupt service routines (ISR's)*



Example
Find the physical and logical address in the interrupt vactor table associated with:
(a) INT 12H (b) INT 8

(a) The physical address for INT 12H are 00048H – 0004BH since (4 x 12H = 48H). that means that the physical memory locations 48H, 49H, 4AH, and 4BH are set aside for the CS and IP of the ISR belonging to INT 12H. the logical address is 0000:0048 – 0000:004BH.
(b) For INT 8, we have 8 x 4 = 32 = 20H; therefore, memory address 00020H, 00021H, 00022H, and 00023H in the interrupt vector table hold the CS:IP of the INT 8 ISR.
The logical address is 0000:0020H – 0000:0023H.

Hardware interrupts.

There pins in the 80x86 that are associated with hardware interrupts.

They are **INTR (interrupt request),**

NMI (nonmaskable interrupt),

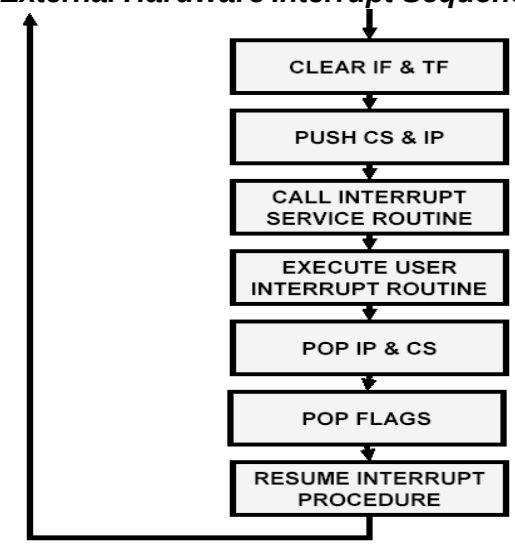
INTA (interrupt acknowledge)

INTR is an input signal into the CPU which can be masked (ignored) and unmasked through the use of instructions CLI and STI.

NMI, which is also an input signal into the CPU, cannot be masked and unmasked using instructions CLI and STI, and for this reason it is called a nonmaskable interrupt.

When either of these interrupts is activated, the 80x86 finishes the instruction which it is executing, pushes FR and the CS:IP of the next instruction onto the stack, then jumps to a fixed location in the interrupt vector table and fetches the CS:IP for the interrupt service routine (ISR) associated with that interrupt. At the end of the ISR, the IRET instruction causes the CPU to get (pop) back its original FR and CS: IP from the stack, thereby forcing the CPU to continue at the instruction where it left off when the interrupt came in.

External Hardware-Interrupt Sequence



Interrupt service routine

To save registers and parameters on the stack { PUSH XX, PUSH YY, PUSH ZZ

Main body of the service routine { . . . . .

To restore register and parameters from the stack { POP ZZ, POP YY, POP XX

Return to main program { IRET

Sequences of hardware interrupts with the 8259

1. After an IR is activated, the 8259 will respond by putting a high on INTR, thereby signaling the CPU for an interrupt request.

2. The 8288 issues the first INTA to the 8259

3. The 8259 receives the first INTA and does internal housekeeping, which includes resolution of priority (if more than one IR has been activated) and resolution of cascading.

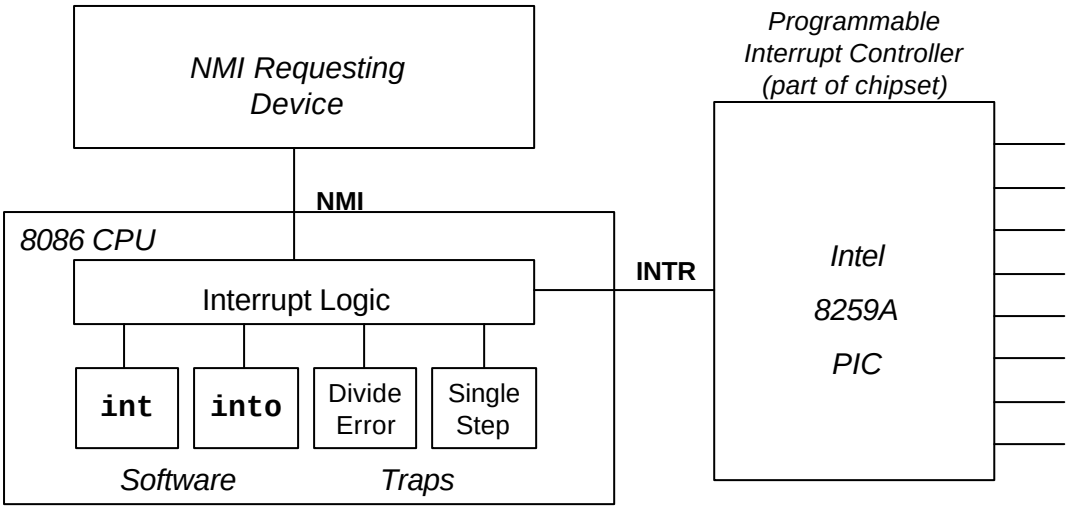
4. The second INTA pulse makes the 8259 to put a single interrupt vector byte on the data bus which the 8088/86 will latch in.

5. The 8088/86 uses this byte to calculate the vector location, which is four times the value of the INT type.

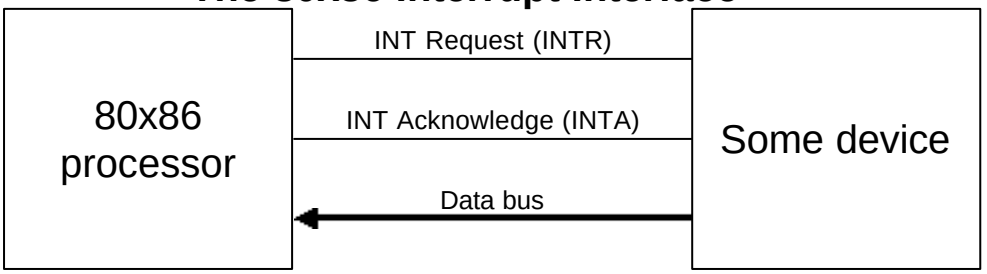
INTR and NMI:

The 80x86 processors have just two hardware interrupt pins. These are labeled INTR and NMI. NMI is nonmaskable interrupt, which means it cannot be blocked-the processor must respond to it. For this reason the NMI input is usually reserved for critical system functions, for example, saving the processor state when a power failure is eminent. INTR, on the other hand, is maskable via the IF flag.

The 82C59A is known as a ***programmable interrupt controller*** or ***PIC***. The operation of the PIC is programmable under software control. The 82C59A can be cascaded to expand from 8 to 64 interrupt inputs.



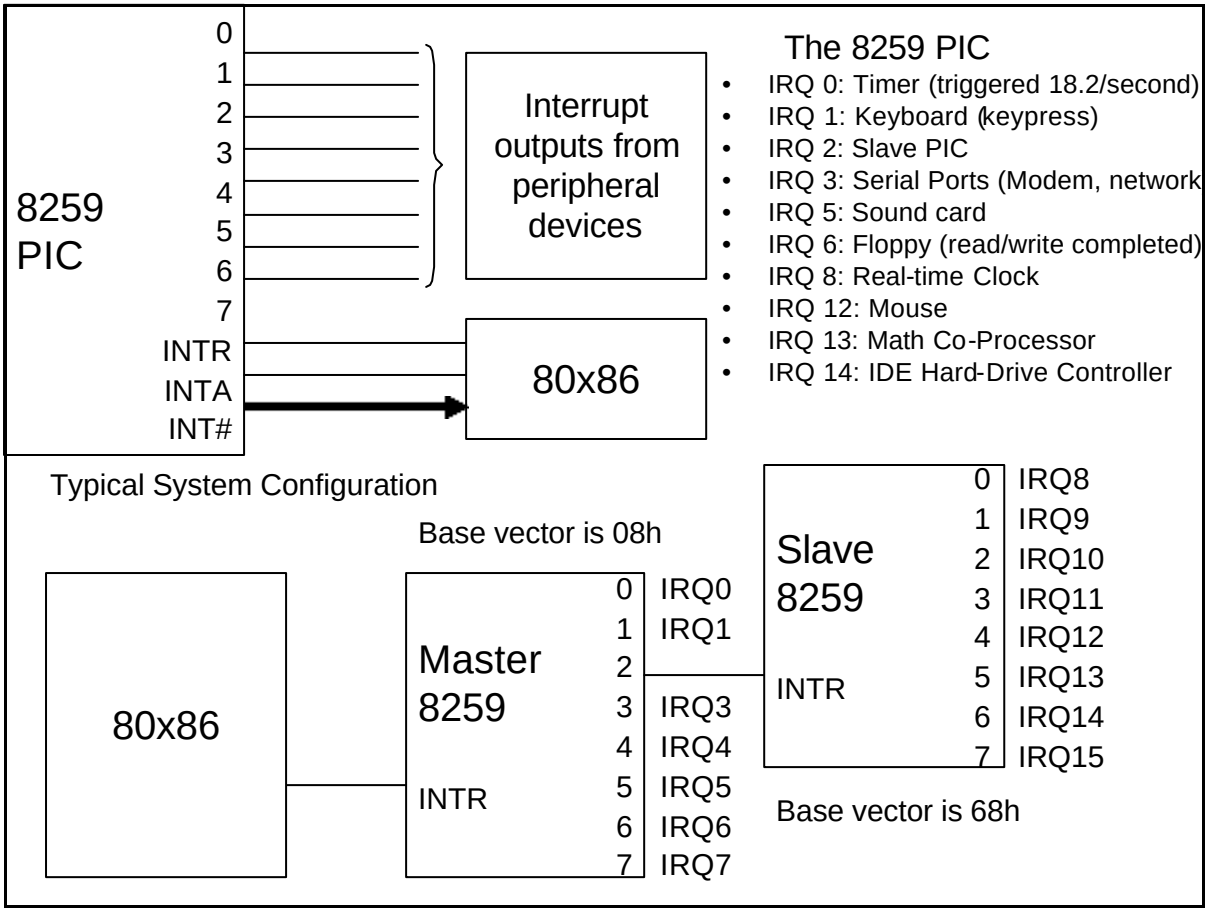
8086 External Interrupt Connections and The 80x86 Interrupt Interface



INTR and NMI. The 80x86 processors have just two hardware interrupt pins.

INTR is used to expand the number of hardware interrupts and should be allowed to use any ''INT nn'' which has not been previously assigned. **The 8259 programmable interrupt controller (PIC) chip can be connected to INTR to expand the number of hardware interrupts up to 64.**

- Device generates request signal
- Device supplies interrupt vector number on data bus
- Processor completes the execution of current instruction and executes ISR corresponding to the interrupt vector number on the data bus
- ISR upon completion acknowledges the interrupt by asserting the INTA signal


8259
PIC
0
1
2
3
4
5
6
7
INTR
INTA
INT#
Interrupt outputs from peripheral devices
80x86
The 8259 PIC
• IRQ 0: Timer (triggered 18.2/second)
• IRQ 1: Keyboard (keypress)
• IRQ 2: Slave PIC
• IRQ 3: Serial Ports (Modem, network
• IRQ 5: Sound card
• IRQ 6: Floppy (read/write completed)
• IRQ 8: Real-time Clock
• IRQ 12: Mouse
• IRQ 13: Math Co-Processor
• IRQ 14: IDE Hard-Drive Controller
Typical System Configuration
Base vector is 08h
80x86
Master
8259
INTR
0 IRQ0
1 IRQ1
2
3 IRQ3
4 IRQ4
5 IRQ5
6 IRQ6
7 IRQ7
Slave
8259
INTR
0 IRQ8
1 IRQ9
2 IRQ10
3 IRQ11
4 IRQ12
5 IRQ13
6 IRQ14
7 IRQ15
Base vector is 68h

Basic Operation.Of 829A

Before the PIC can be used, the interrupt type numbers must be programmed. In addition, the operating mode and priority scheme must be selected. These software codes are writen via the 8-bit bidirectioal data bus lines.

Ýf an interrupt request occurs that is of a higher priority than that currently being serviced (if any), the PIC drives its INT output high. Assuming IF (the processor's interrupts enabled flag) is set, the processor finishes the current imstruction and responds by outputting the first of two INTA pulses. This pulse freezes (stores) all interrupt requests within the PIC in a special interrupt request register (IRR). The interrupt signal can now be removed.

When the second INTA pulse is received by the PIC.

The PIC then outputs a type number corresponding to the active IR input. This number is multiplied by four within the processor and then used a pointer into the interrupt vector table located.

Servicing an Interrupt

- Complete current instruction
- Preserve current context
  - **PUSHF Store flags to stack**
  - **Clear Trap Flag (TF) & Interrupt Flag (IF)**
  - **Store return address to stack PUSH CS, PUSH IP**
- Identify Source
  - Read 8259 PIC status register
  - Determine which device (N) triggered interrupt
- Activate Interrupt Service Routine
  - Use N to index vector table
  - Read CS/IP from table
  - Jump to instruction
- Execute Interrupt Service Routine
  - usually the handler immediately re-enables the interrupt system (to allow higher priority interrupts to occur) (STI instruction)
  - process the interrupt
- Indicate End-Of-Interrupt (EOI) to 8259 PIC

```
mov   al, 20h
out   20h, al
```

;transfers the contents of AL to I/O port 20h

- Return (IRET)
  - POP IP (Far Return)
  - POP CS
  - POPF (Restore Flags)

Sequence of operation in interrupt

1- Normal processing

2- Interrupt occurs

3- Finish current instruction

4- Push flags, CS, and IP onto stack

5- Branch to ISR

6- Execute ISR

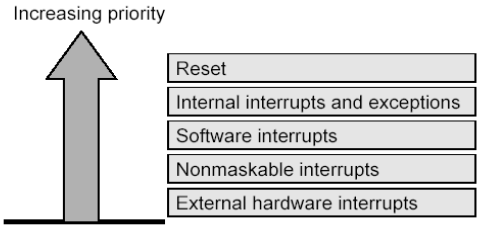
7- Pop flags, CS, and IP from stack

8- Normal processing resumes

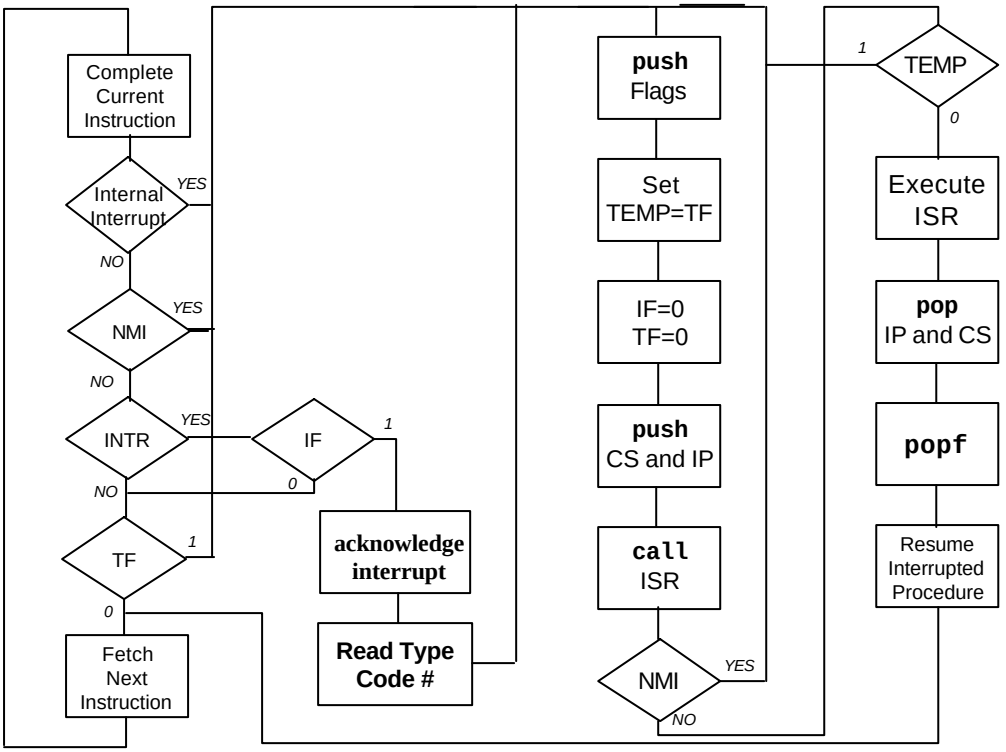
Interrupt Priority

- Each interrupts is given a different priority level by assigning it a ***type number***. Type 0 identifies the highest-priority interrupt, and type 255 identifies the
- lowest-priority interrupt.

- Lower interrupt vectors have higher priority
- Lower priority interrupts normally cannot interrupt higher priority interrupts
  - ISR for INT 21h is running
    - Computer gets request from device attached to IRQ8 (INT 78h)
    - INT 21h procedure must finish before IRQ8 device can be serviced
  - ISR for INT 21h is running
    - Computer gets request from Timer 0 IRQ0 (INT 8h)
    - Code for INT 21h gets interrupted, ISR for timer runs immediately, INT21h finishes afterwards
    - Priority. On the PC, the PIC is operated in the fully nested mode. This means that the lowest numbered IRQ input has highest priority. Interrupts of a lower priority will not be acknowledged by the PIC (and therefore not forwarded to the processor) until the higher priority interrupts have been serviced.



What Happens During an Interrupt ?



Difference between INT and CALL instructions

1. A ''CALL '' instruction can jump to any location within the 1 megabyte address range of the 8088/86 CPU, but ''INT nn'' goes to a fixed memory location in the interrupt vector table to get the address of the interrupt service routine.

2. A ''CALL'' instruction is used by the programmer in the sequence of instructions in the program but an externally activated hardware interrupt can come in at any time, requesting the attention of the CPU.

3. A ''CALL'' instruction cannot be masked (disabled), but ''INT nn'' belonging to externally activated hardware interrupts can be masked. This is discussed in a later section.

4. A ''CALL'' instruction automatically saves only CS:IP of the next instruction on the stack, while ''INT nn'' saves FR (flag register) in addition to CS:IP of the next instruction.

5. At the end of the subroutine that has been called by the ''CALL'' instruction, the RETF (return) is the last instruction, whereas the last instruction in the interrupt service routine (ISR) for ''INT nn'' is the instruction IRET (interrupt return). The difference is that RETF pops CS, IP off the stack but the IRET pops off the FR (flag register) in addition to CS and IP.

Interrupt Instructions

Mnemonic	Meaning	Format	Operation	Flags affected
CLI	Clear interrupt flag	CLI	0 → (IF)	IF
STI	Set interrupt flag	STI	1 → (IF)	IF
INT n	Type n software interrupt	INT n	(Flags) → ((SP)-2) 0 → TF, IF (CS) → ((SP) – 4) (2+4xn) → (CS) (IP) → ((SP) – 6) (4xn) → (IP)	TF, IF
IRET	Interrupt return	IRET	((SP)) → (IP) ((SP)+2) →(CS) ((SP)+4) →(Flags) (SP) + 6 → (SP)	All
INTO	Interrupt on overflow	INTO	INT 4 steps	TF, IF