9 Interrupts & 8259

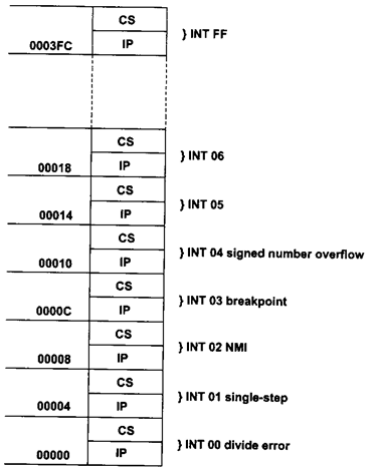
Interrupts

- **The main function of Interrupts**: CPU does not need to check I/O device frequently. If I/O is ready, then an interrupt will be sent to CPU to inform the CPU the information.
- Totally 256 **Interrupt Types** (`INT 00H ~ INT 0FFH`)
  - `0 ~ 4`: dedicated (reserved for special purposes);
  - `5 ~ 31`: reserved for system use, where `08H ~ 0FH` is for 8259A and `10H ~ 1FH` is for BIOS;
  - `32 ~ 255` reserved for users, where `20H ~ 3FH` is for DOS and `40H ~ FFH` is open.
- **Interrupt vector** （中断向量）(CS, IP), points the entrance address of the corresponding *interrupt service routine* (ISR, to handle the interrupts). Therefore, an interrupt vector needs 4 bytes (CS: 2 bytes and IP 2 bytes). Notes: the "physical address" and "logical address" is the address of interrupt vector, not CS and IP.

Table 14-1: Interrupt Vector

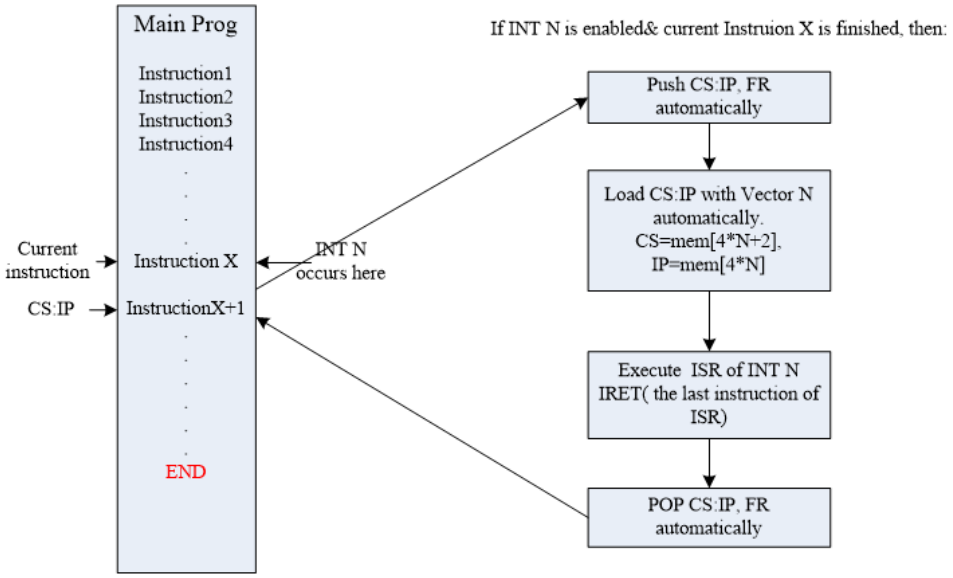
INT Number	Physical Address	Logical Address
INT 00	00000	0000:0000
INT 01	00004	0000:0004
INT 02	00008	0000:0008
INT 03	0000C	0000:000C
INT 04	00010	0000:0010
INT 05	00014	0000:0014
...	...	...
INT FF	003FC	0000:03FC

- **Interrupt vector table** （中断向量表）: the storage the store the interrupt vector.



- **Categories of Interrupts**
  - **Hardware interrupts** (external interrupts)
    - Maskable（可屏蔽的） (from INTR)
    - Non-maskable（不可屏蔽的） (from NMI)
  - **Software interrupts** (internal interrupts)
    - Using `INT` instructions;
    - Predefined *conditional* (*exception*) interrupts

ISR and Main Programs: An ISR is launched by an interrupt event (internal 'INT xx' or external NMI and INTR). So ISR is **separated** from main.



Hardware Interrupts

- Non-maskable interrupt:
  - Trigger: NMI pin, input-signal, rising edge and two-cycle high activate;
  - **TYPE**: `INT 02`;
  - not affected by the `IF` (interrupt flag)
  - Reasons: Usually severe errors, such as parity error.
- Maskable interrupt:
  - Trigger: INTR pin, input-signal, high activate;
  - **TYPE**: No predefined type;
  - `IF = 1`, enable; `IF = 0`, disable; (use `STI` to set `IF`, and use `CLI` to clear `IF`);
  - Reasons: Interrupt requests of external I/O devices.

Process Maskable Interrupts: CPU responds to INTR interrupt requests:

- External I/O devices send interrupt requests to CPU;
- CPU will check `INTR` pin on the last cycle of an instruction: if the `INTR` is high and `IF = 1`, CPU responds to the interrupt request;
- CPU sends **two negative pulses** on the `~INTA` pin to the I/O device;
- After receiving the second `~INTA`, I/O device sends the interrupt type *N* on the data bus.

How CPU executes the ISR of `INT N`

- CPU reads N from data bus;
- Push the `FR` in the stack;
- Clear `IF` (automatically does not allow nested interrupts) and `TF` (trap flag);
- Push the `CS` and `IP` of the next instruction in stack;
- Load the ISR entrance address and moves to the ISR;
- At the end of ISR, `IRET` will pop `IP` and `CS` and `FR` in turn, CPU returns to previous program and proceeds.

Software Interrupts

- `INT xx` instruction.
  - An ISR is called upon instruction such as `INT xx`;
  - CPU always responds and goes execute the corresponding ISR, not affected by `IF`;

- You can call an ISR by using `INT` instructions.

  > Difference between `INT` and `CALL`
  >
  > - `CALL FAR` can jump anywhere within 1MB v.s. `INT` jumps to a fix location (finding the corresponding ISR);
  > - `CALL FAR` is in the sequence of instructions v.s. an external interrupt can come in at any time
  > - `CALL FAR` cannot be masked (disabled) v.s. an external interrupt can be masked
  > - `CALL FAR` saves CS:IP of next instruction v.s. `INT` saves `FR + CS:IP` of next instruction
  > - last instruction: `RETF` v.s. `IRET`

- Predefined conditional interrupts:
  - `INT 00`: (divide error) dividing a number by zero, or quotient is too large.
  - `INT 01`: (single step) If `TF = 1`, CPU will generates an `INT 1` interrupt after executing each instruction for debugging.

    > How to clear `TF`?
    >
    > ```
    > PUSHF
    > POP AX
    > AND AX, 0FEFFH
    > PUSH AX
    > POPF
    > ```

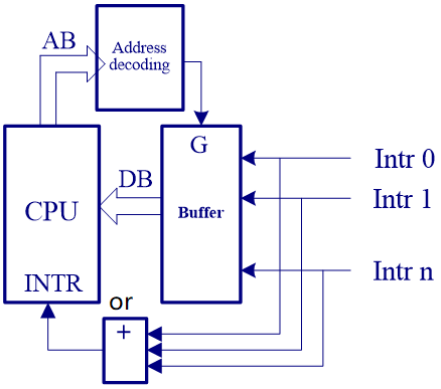
  - `INT 03`: (breakpoints) When CPU hits the breakpoint set in the program, CPU generates `INT 3` interrupt for debugging.
  - `INT 04`: (signed number overflow): use `INTO` instruction to check the `OF` after an arithmetic instruction, and it may generate an interrupt according to whether the result overflows.

Process Non-maskable & Software Interrupts

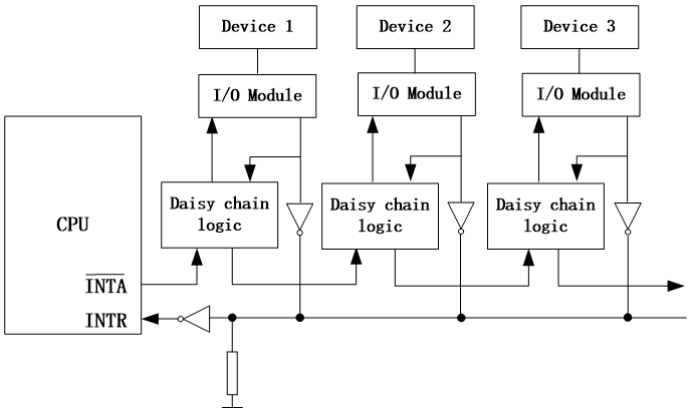
- For NMI: CPU checks NMI, generates `INT 02` interrupt, automatically regardless of `IF` and execute the ISR;
- For internal interrupts: CPU generates `INT N` interrupt automatically and executes to the corresponding ISR.

Interrupt Priority

- `INT` instruction has a higher priority than INTR and NMI;
- NMI has higher priority than INTR;
- For different external interrupt requests, different strategies can be used to determine their priorities.
  - **Software polling**: the sequence of checking determines the priority.



- **Hardware checking**: the location in the daisy chain counts.

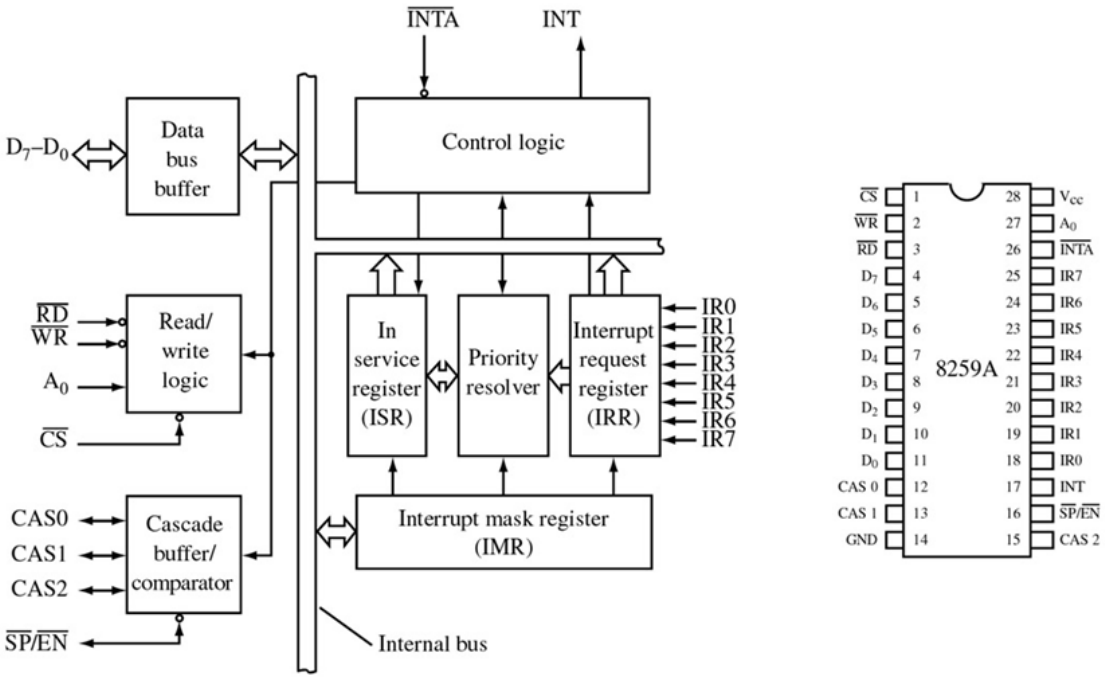


- **Interrupt Controller**: 8259.

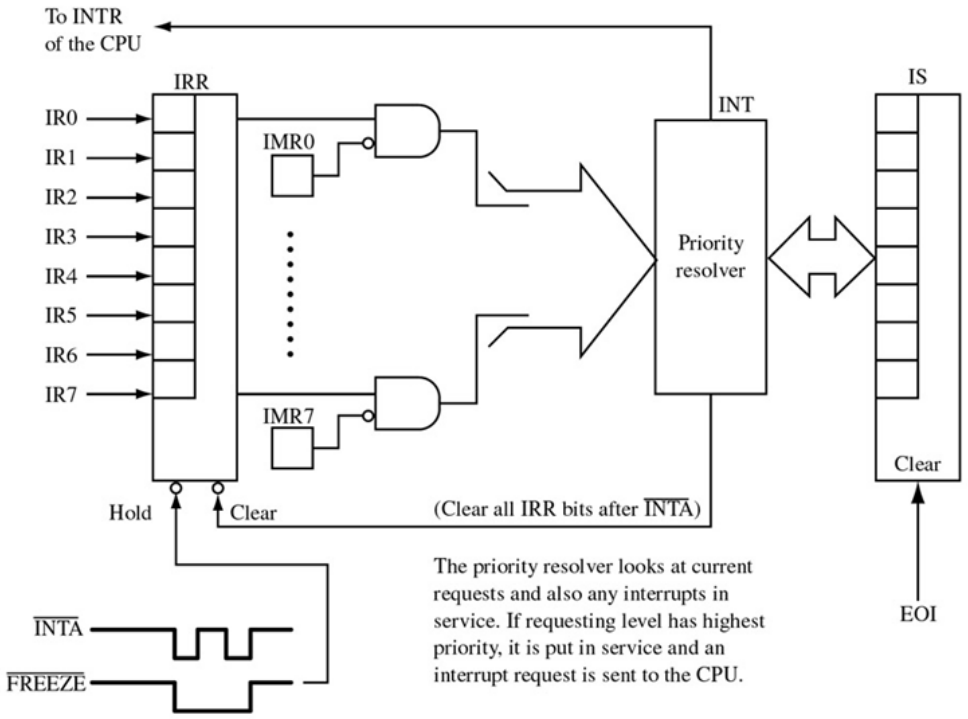
8259 Programmable Interrupt Controller (PIC): I/O device for managing the interrupt requests. (It can be a separate circuits, but nowadays PIC usually sits inside the CPU.)

- PIC can deal with up to 64 interrupt inputs;
- interrupts can be masked;
- various priority schemes can also programmed.

8259 Interface

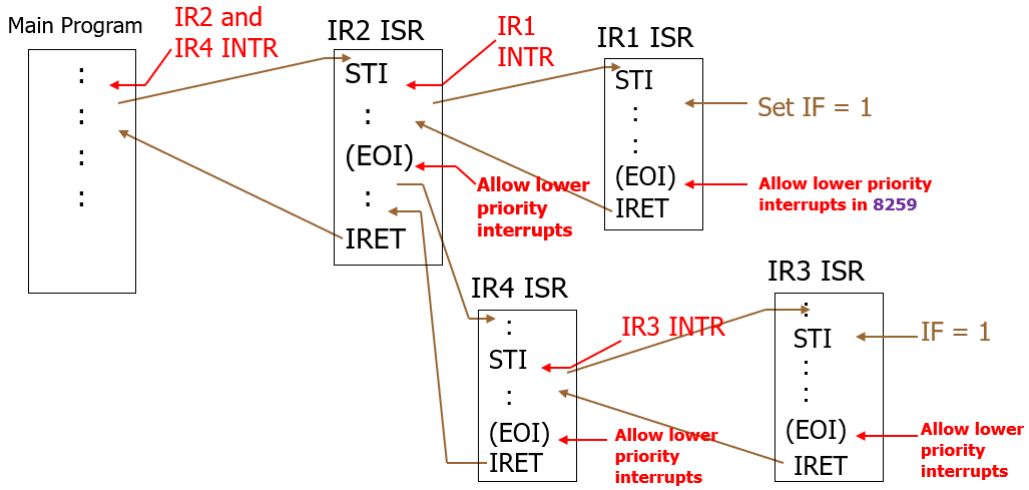


Registers in 8259



- **IRR** (Interrupt Request Register): store the Interrupt Request.
- **IMR** (Interrupt Mask Register): `IMR = 1`, 8259 masks this signal, `IMR = 0`, allows the signal.
- **Priority Resolver**: define different priorities to different interrupts to handle the interrupt.
- **ISR** (Interrupt Service / In-Service Register): the result of priority resolver.
- ISR will respond to CPU, and CPU will know the priorities.

Interrupt Nesting Higher priority interrupts can interrupt lower interrupts. Suppose smaller number indicates the higher priority.



where, `STI` is used to re-open the interrupt flag which is closed by ISR (Interrupt Service Routine). `EOI` will clear the flags in the `ISR` (Interrupt Service Register) , which allows lower priority interrupts to come in.