

```

linux.git/drivers/kvm/i8259.c
01 void kvm_pic_set_irq(void *opaque, int irq, int level)
02 {
03     struct kvm_pic *s = opaque;
04
05     pic_set_irq1(&s->pics[irq >> 3], irq & 7, level);
06     .....
07 }
08
09 static inline void pic_set_irq1(struct kvm_kpic_state *s,
10 int irq, int level)
11 {
12     int mask;
13     mask = 1 << irq;
14     if (s->elcr & mask) /* level triggered */
15         ...
16     else /* edge triggered */
17         if (level) {
18             if ((s->last_irr & mask) == 0)
19                 s->irr |= mask;
20             s->last_irr |= mask;
21         } else
22             s->last_irr &= ~mask;
23 }

```

信号有边缘触发和水平触发，在物理上可以理解为，8329A 在前一个周期检测到管脚信号是 0，当前周期检测到管脚信号是 1，如果是上升沿触发模式，那么 8259A 就认为外设有了请求了，这种触发模式就是边缘触发。对于水平触发，以高电平触发为例，当 8259A 检测到管脚处于高电平，则认为外设来请求了。

在虚拟 8259A 的结构体 `kvm_kpic_state` 中，寄存器 `elcr` 是用来记录 8259A 被设置的触发模式的，我们以边缘触发为例进行讨论，即代码第 16 ~ 22 行。参数 `level` 即相当于硬件层面的电信号，0 表示低电平，1 表示高电平。当管脚收到一个低电平时，即 `level` 的值为 0，代码进入 `else` 分支，即代码第 21、22 行，结构体 `kvm_kpic_state` 中的字段 `last_irr` 会清除该 IRQ 对应 IRR 的位，即相当于设置该中断管脚为低电平状态。当管脚收到高电平时，即 `level` 的值为 1，代码进入 `if` 分支，即代码第 17 ~ 20 行，此时 8259A 将判断之前该管脚的状态，即代码第 18 行，也就是判断结构体 `kvm_kpic_state` 中的字段 `last_irr` 中该 IRQ 对应 IRR 的位，如果之前管脚为低电平，而现在管脚是高电平，那么显然管脚电平有一个跳变，说明中断源发出了中断请求，8259A 在字段 `irr` 中记录下中断请求。当然，同时需要在字段 `last_irr` 记录下当前该管脚的状态。

3.2.4 设置待处理中断标识

当 8259A 将中断请求记录到 IRR 中后，下一步就是开启一个中断评估（`evaluate`）过程，包括评估中断是否被屏蔽、多个中断请求的优先级等，最后将通过管脚 INTA 通知