CSE 511: Operating Systems Design

Lecture 11

Local APIC and I/O APIC

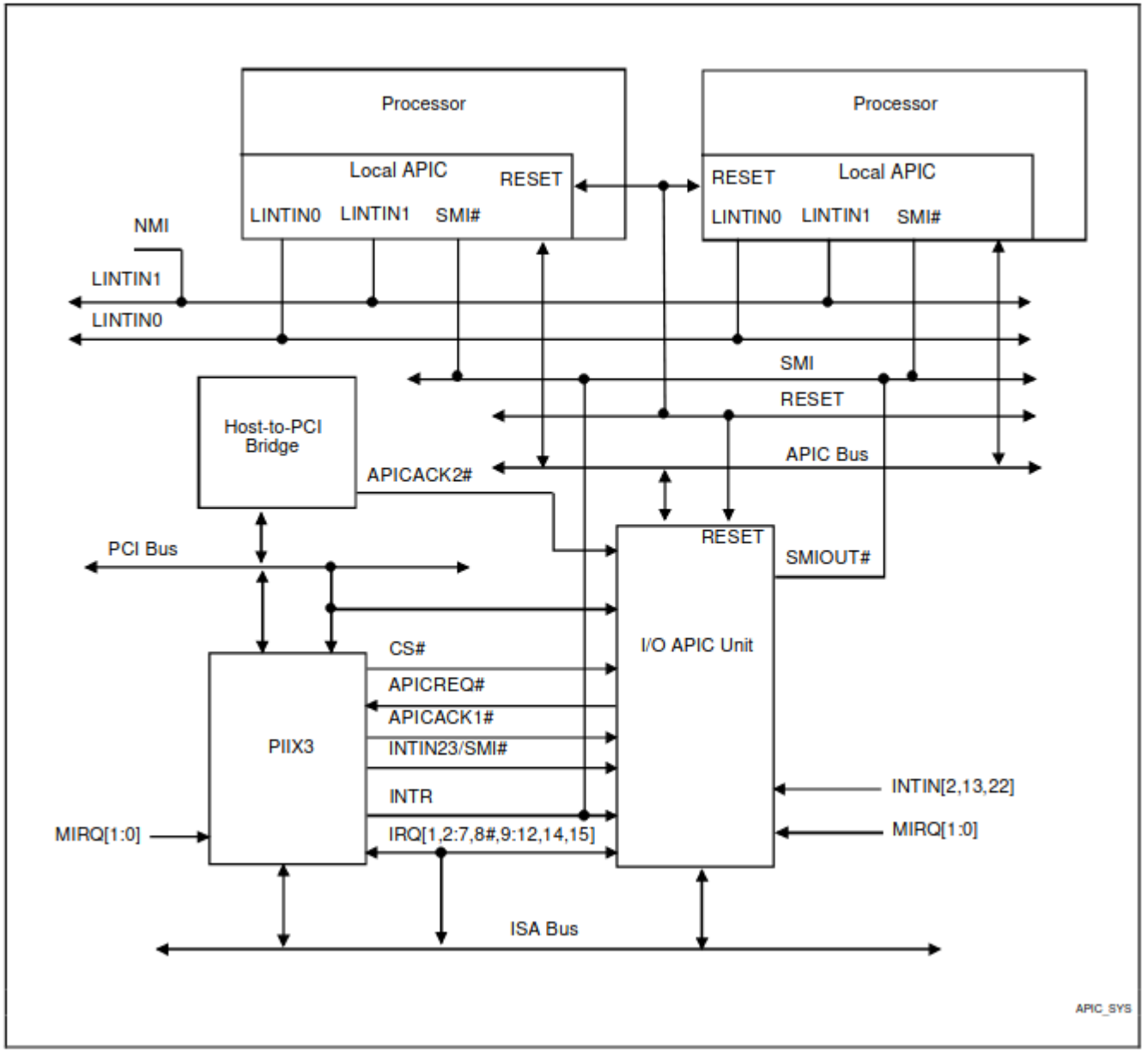
MSI Interrupts

APIC Timer

Advanced Programmable Interrupt Controller (APIC)

- PIC is not suitable for SMP and also has limits on the number of IRQs
  - e.g., we can have IRQ conflicts for different devices
- In APIC systems, we have two (rather than one) controllers
  - Local APIC (LAPIC): handles CPU-specific interrupts
  - I/O APIC (IOAPIC): handles interrupts from devices, distributes interrupts across all CPUs
  - IOAPIC is wired to LAPICs

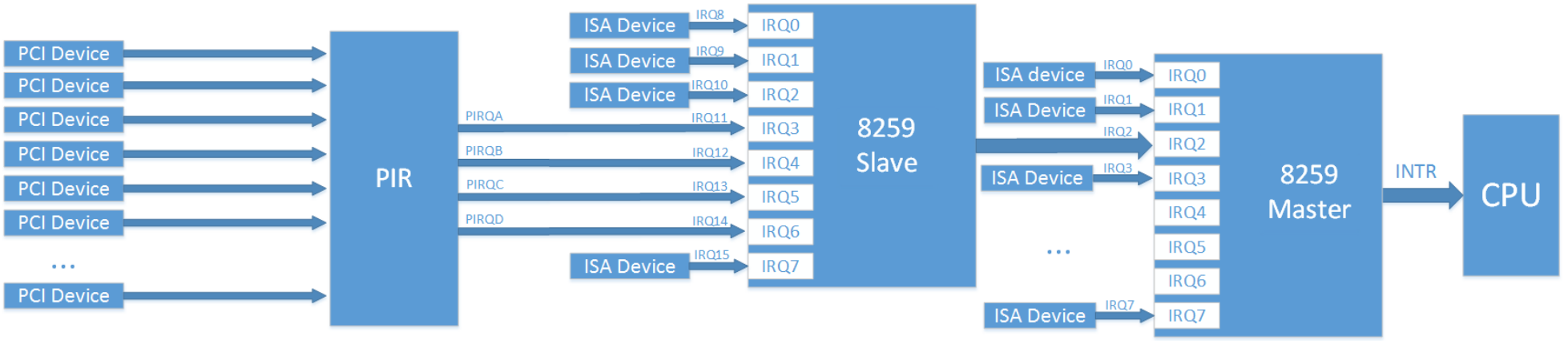
Advanced Programmable Interrupt Controller (APIC)



https://pdos.csail.mit.edu/6.828/2016/readings/ia32/ioapic.pdf

PIC Interrupt Sharing

- Limited number of IRQs
- Use Programmable Interrupt Requests (PIRQ)
  - e.g., 20 devices, each group of 5 devices shares the same IRQ (4 IRQs in total)
  - The CPU needs to inquire which device issued an interrupt for each PIRQ group by asking *all* devices for this PIRQ



* The picture is taken from https://habr.com/en/post/446312/

PIC Interrupt Sharing

- Each PCI devices has INTA, INTB, INTC, INTD lines
  - Which line is actually used determined by configuration (e.g., the firmware can give us these mappings)
- Even with these workarounds for PCI devices, this does not work for SMP systems
- Local APIC and I/O APIC solve this problem
  - One Local APIC for each CPU
  - I/O APIC is wired to each Local APIC

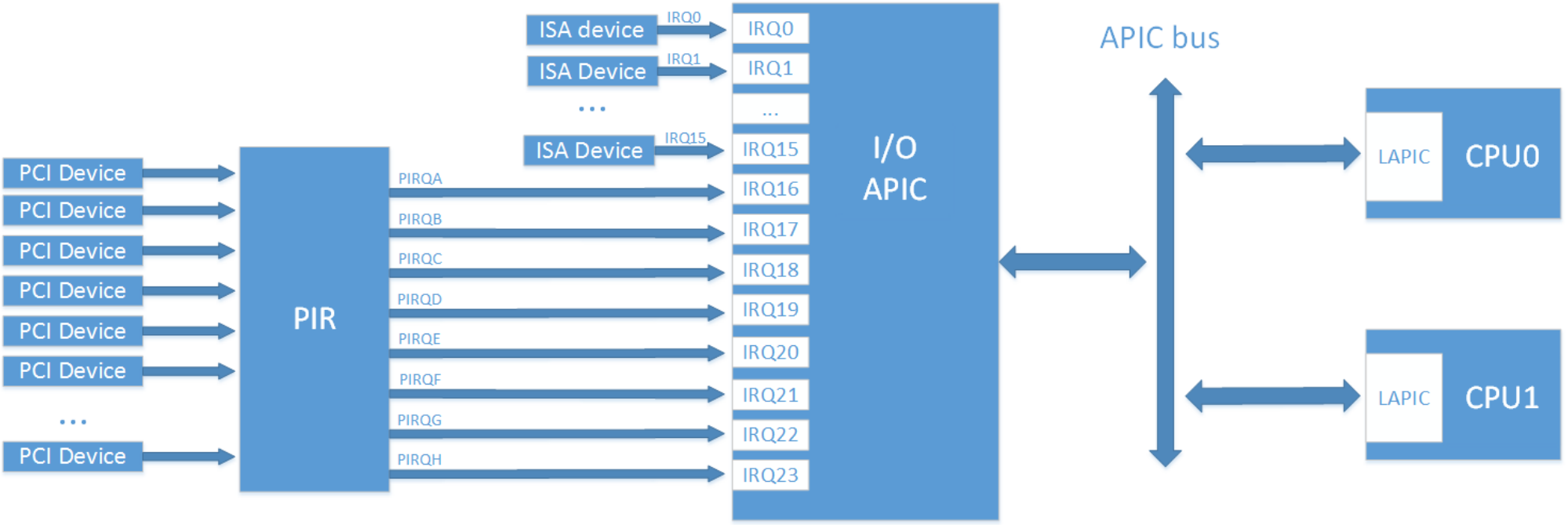
Local APIC Variants

- Original APIC
  - i486 processors
  - Supports at most 16 CPUs
- xAPIC
  - Pentium 4+
  - Fully backward compatible
  - Supports at most 256 CPUs
- x2APIC
  - The Nehalem architecture (2008)
  - Supports 2^32 CPUs
  - Need to enable the x2APIC mode

Local APIC and I/O APIC

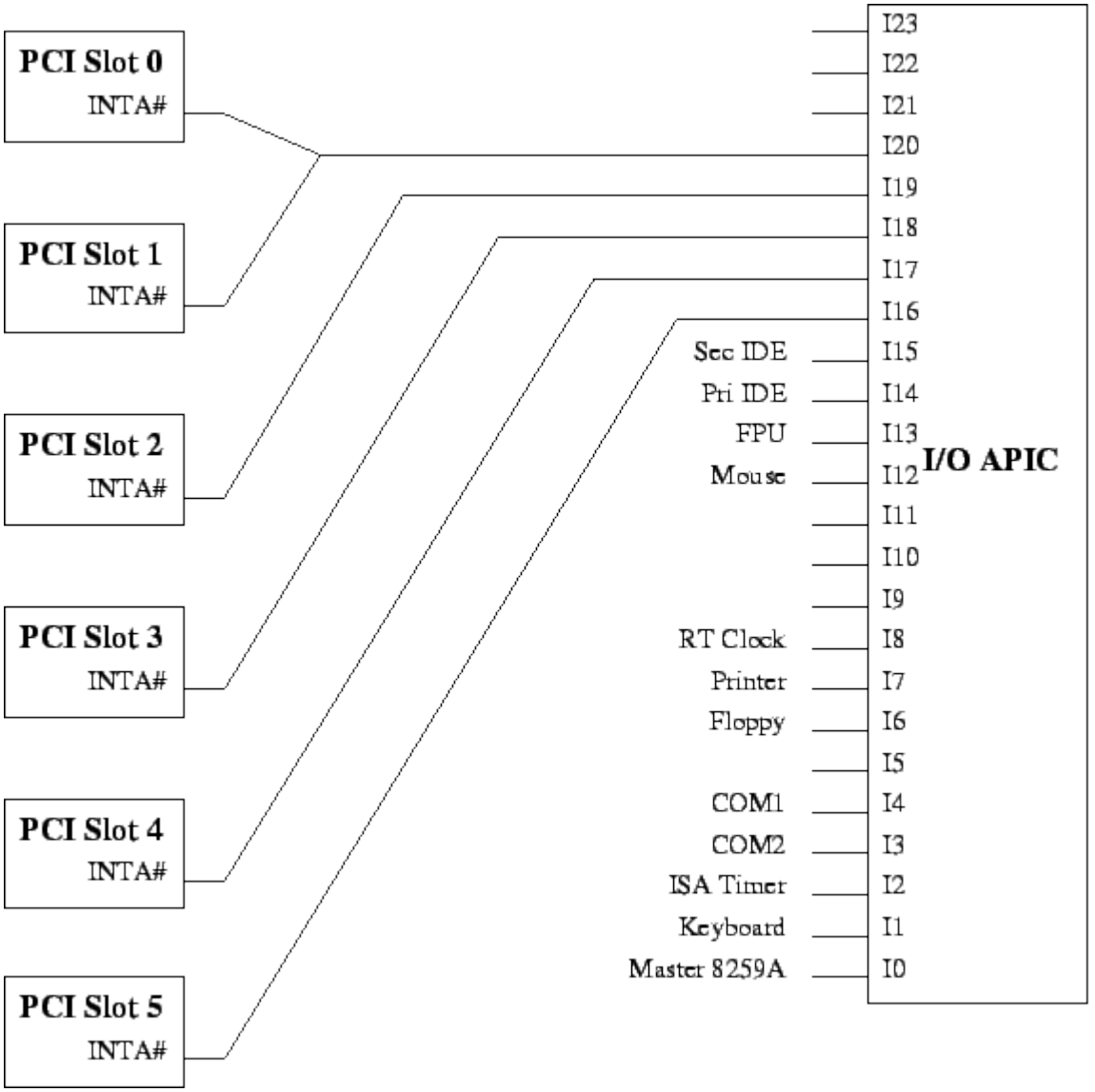
- With I/O APIC, we have 24 IRQs
  - 16 IRQs for legacy ISA devices
  - 8 IRQs for PCI devices
- We can have 8 PIRQs rather than 4 PIRQs
- I/O APIC forwards interrupts to Local APICs
  - Interrupts can be spread across different CPUs
- Can have several I/O APICs in the system
  - The total number of IRQs can be higher than 24

Local APIC and I/O APIC



* The picture is taken from https://habr.com/en/post/446312/

Local APIC and I/O APIC



* The picture is taken from https://people.freebsd.org/~jhb/papers/bsdcan/2007/article/node4.html

Local APIC

- Registers are memory-mapped
- Local APIC can be used to wake up CPUs
  - e.g., when we want to initially bring up all CPUs in the SMP system
- Local APIC delivers hardware interrupts to a CPU
  - Either through I/O APIC or using MSI interrupts

Local APIC

ruslan@ruslan-ThinkPad-T470p: ~/librettos-src/rumprun-smp

```
static void
x86_lapic_enable(void)
{
    uint32_t eax, ebx, ecx, edx;
    void *lapic_base;
    uint32_t lapic_id;
    uint32_t x2apic = 0;

    x86_cpuid(1, &eax, &ebx, &ecx, &edx);

    if (!(edx & (1U << 9))) { /* No APIC. */
        bmk_set_cpu(lapic_base, NULL);
        bmk_set_cpu(lapic_id, 0);
        return;
    }

    if (((eax >> 8) & 0x0FU) < 6) { /* CPU family: before P6. */
        lapic_base = x86_mp_lapic_base;
    } else {
        if ((ecx & (1U << 21)) != 0)
            x2apic = X86_MSR_APIC_X2APIC;
        x86_rdmsr(X86_MSR_APIC, &eax, &edx);
        x86_x2apic_barrier();
        x86_wrmsr(X86_MSR_APIC, eax | X86_MSR_APIC_ENABLE | x2apic, edx);
        if (!x2apic) {
#ifdef __i386__
            lapic_base = (void *) (eax & 0xFFFFF000U);
#else
            lapic_base = (void *) (((uint64_t) (edx & 0x0FU) << 32) | (eax & 0xF
FFFF000U));
#endif
        } else {
            lapic_base = X86_LAPIC_X2APIC;
        }
    }

    bmk_set_cpu(lapic_base, lapic_base);
    lapic_id = x86_lapic_read_id();
    bmk_set_cpu(lapic_id, lapic_id);

    if (x2apic) {
        bmk_printf("x2APIC enabled for #%u\n", lapic_id);
    } else {
        bmk_printf("APIC enabled for #%u, BASE: 0x%p\n",
            lapic_id, lapic_base);
    }

    /* Spurious interrupt: EN=1. */
    x86_lapic_write(X86_LAPIC_SPURIOUS,
        x86_lapic_read(X86_LAPIC_SPURIOUS) | (0x1U << 8));
}
```

301,0-1 75%

ruslan@ruslan-ThinkPad-T470p: ~/librettos-src/rumprun-smp

```
#define X86_MSR_APIC        0x01BU
#define X86_MSR_APIC_ENABLE 0x800U
#define X86_MSR_APIC_X2APIC 0x400U

#define X86_MSR_X2APIC_BASE 0x800U

#define X86_LAPIC_X2APIC    ((void *) (-1L))

#define X86_LAPIC_ID        0x02U
#define X86_LAPIC_SPURIOUS  0x0FU
#define X86_LAPIC_ICR       0x30U
```

8,1 28%

Local APIC has a base memory address, can be determined by CPUID

Local APIC

ruslan@ruslan-ThinkPad-T470p: ~/librettos-src/rumprun-smp

```
static inline uint32_t
x86_lapic_read(uint32_t offset)
{
    void *lapic_base = bmk_get_cpu(lapic_base);
    if (lapic_base == X86_LAPIC_X2APIC) {
        uint32_t eax, edx;
        x86_rdmsr(X86_MSR_X2APIC_BASE + offset, &eax, &edx);
        return eax;
    }
    return *(volatile uint32_t *) (lapic_base + (offset << 4));
}

static inline uint32_t
x86_lapic_read_id(void)
{
    void *lapic_base = bmk_get_cpu(lapic_base);
    if (lapic_base == X86_LAPIC_X2APIC) {
        uint32_t eax, edx;
        x86_rdmsr(X86_MSR_X2APIC_BASE + X86_LAPIC_ID, &eax, &edx);
        return eax;
    }
    return (*(volatile uint32_t *) (lapic_base + (X86_LAPIC_ID << 4))) >> 24;
}

static inline void
x86_x2apic_barrier(void)
{
    /* Seems we need *both* mfence and lfence per Intel (10.12.3) */
    __asm__ __volatile("mfence; lfence" :::
                       "memory");
}
```

196,2-5 53%

Local APIC

ruslan@ruslan-ThinkPad-T470p: ~/librettos-src/rumprun-smp

```
static inline void
x86_lapic_write(uint32_t offset, uint32_t value)
{
    void *lapic_base = bmk_get_cpu(lapic_base);
    if (lapic_base == X86_LAPIC_X2APIC) {
        x86_x2apic_barrier();
        x86_wrmsr(X86_MSR_X2APIC_BASE + offset, value, 0);
        return;
    }
    *(volatile uint32_t *) (lapic_base + (offset << 4)) = value;
}

static inline void
x86_x2apic_write_icr(uint32_t cmd_low, uint32_t cmd_high)
{
    x86_x2apic_barrier();
    x86_wrmsr(X86_MSR_X2APIC_BASE + X86_LAPIC_ICR, cmd_low, cmd_high);
}

static inline void
x86_lapic_write_icr(uint32_t cmd)
{
    void *lapic_base = bmk_get_cpu(lapic_base);
    if (lapic_base == X86_LAPIC_X2APIC) {
        x86_x2apic_write_icr(cmd, 0);
        return;
    }
    *(volatile uint32_t *) (lapic_base + (X86_LAPIC_ICR << 4)) = cmd;
    while ((*(volatile uint32_t *) (lapic_base + (X86_LAPIC_ICR << 4)))
                & 0x1000U) ; /* Wait until the command is completed. */
}

static void
                                                    229,2-8       62%
```

ICR:
Interrupt Command Register

Enabling CPUs using Local APIC

ruslan@ruslan-ThinkPad-T470p: ~/librettos-src/rumprun-smp

```
    x86_lapic_enable();

    if (total_cpus != 1) {
        unsigned long trampoline_size = (unsigned long)
            ((char *) &trampoline_32 - (char *) &trampoline);

        /* Initialize number of CPUs > 1. */
        bmk_numcpus = total_cpus;
        /* Copy the trampoline code. */
        backup = bmk_xmalloc_bmk(trampoline_size);
        bmk_memcpy(backup, (void *) 0x7000, trampoline_size);
        bmk_memcpy((void *) 0x7000, &trampoline, trampoline_size);
        /* INIT: DSH=11 00 TM=0 LV=1 0 DS=0 DM=0 DMODE=101
                 VECTOR=0000. */
        x86_lapic_write_icr(0x000C4500U);
        x86_mp_delay(10000000);
        /* SIPI: DSH=11 00 TM=0 LV=1 0 DS=0 DM=0 DMODE=110
                 VECTOR=0007 (i.e., the trampoline 0x7000). */
        x86_lapic_write_icr(0x000C4607U);
        x86_mp_delay(200000);
        /* SIPI: DSH=11 00 TM=0 LV=1 0 DS=0 DM=0 DMODE=110
                 VECTOR=0007 (i.e., the trampoline 0x7000). */
        x86_lapic_write_icr(0x000C4607U);
        x86_mp_delay(200000);
        /* Wait until all CPUs wake up. */
        while (boot_num_cpus != total_cpus) ;
        /* Restore the trampoline area. */
        bmk_memcpy((void *) 0x7000, backup, trampoline_size);
        bmk_memfree(backup, BMK_MEMWHO_WIREDBMK);
    }
    bmk_printf("all %u CPUs are awake\n", total_cpus);
}
                                                    358,2-8       98%
```

MSI Interrupts

- PCI Express eliminated interrupt lines
  - INTx are emulated with messages
  - No physical wire connection
- We do not have to use INTx
  - Message Signaled Interrupts (MSI)
  - A device writes into the memory-mapped region of Local APIC
  - No need for I/O APIC
  - No longer restricted to 4 interrupts per a PCI device
    - Up to 32 interrupts with MSI and 2048 with MSI-X

MSI Interrupts

* The picture is taken from https://habr.com/en/post/446312/

Local APIC

- xAPIC added support for 256 CPUs, it is backward compatible with the initial version of APIC
- x2APIC added support for 2^32 CPUs, not compatible with xAPIC but uses a very similar register layout

Table 2-2. Local APIC Register Address Map Supported by x2APIC (Contd.)

MMIO Offset (xAPIC mode)	MSR Offset (x2APIC mode)	Register Name	R/W Semantics	Comments
0320H	032H	LVT Timer Register	Read/Write.	
0330H	033H	LVT Thermal Sensor Register	Read/Write.	
0340H	034H	LVT Performance Monitoring Register	Read/Write.	

* Taken from https://software.intel.com/content/dam/develop/public/us/en/documents/64-architecture-x2apic-specification.pdf

Useful Resources

- APIC general description, Chapter 10, Volume 3A
  - https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html
- x2APIC changes
  - https://software.intel.com/content/dam/develop/public/us/en/documents/64-architecture-x2apic-specification.pdf

Accessing xAPIC/x2APIC registers

```
#define X86_MSR_X2APIC_BASE 0x800U // apic.h

static void *lapic_base = NULL; // initialized in x86_lapic_enable()

static inline uint32_t
x86_lapic_read(uint32_t offset)
{
    if (lapic_base == X86_LAPIC_X2APIC) {
        uint64_t msr = rdmsr(X86_MSR_X2APIC_BASE + offset);
        return (uint32_t) msr;
    }
    return *(volatile uint32_t *) (lapic_base + (offset << 4));
}
```

EOI (End of Interrupt)

- When completing an interrupt, you should write to a special APIC register
- You must write “0”:
  - “In xAPIC mode, the EOI register is written by an interrupt service routine to indicate that the current interrupt service has completed”
  - “In the x2APIC mode, the write of a zero value to EOI register is enforced. Writes of a non-zero value to the EOI register in x2APIC mode will raise a GP fault. System software continues to have to perform the EOI write to indicate interrupt service completion. But in x2APIC mode, the EOI write is with a value of zero”

APIC Timer

- Provides a special mechanism for periodic or one-shot interrupts

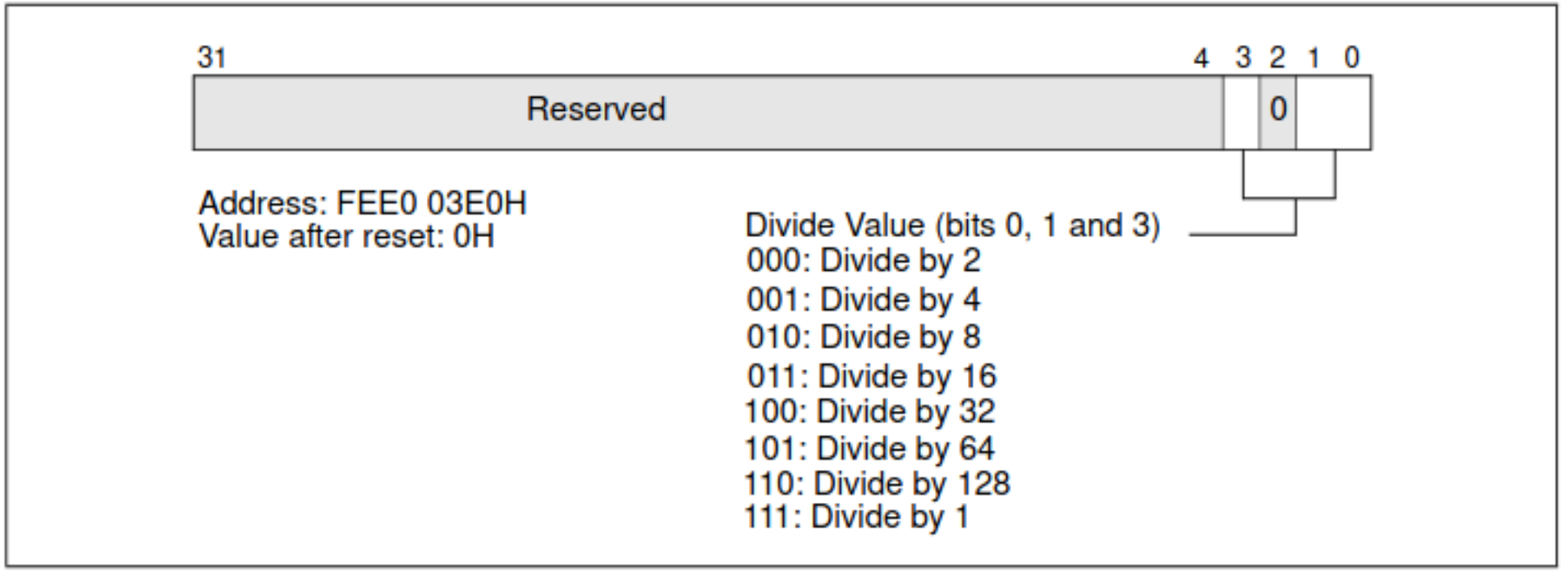


Figure 10-10. Divide Configuration Register

APIC Timer

- The timer can be configured through the timer LVT entry for one-shot or periodic operation
  - “In one-shot mode, the timer is started by programming its initial-count register. The initial count value is then copied into the current-count register and count-down begins. After the timer reaches zero, an timer interrupt is generated and the timer remains at its 0 value until reprogrammed”
  - “In periodic mode, the current-count register is automatically reloaded from the initial-count register when the count reaches 0 and a timer interrupt is generated, and the count-down is repeated. If during the count-down process the initial-count register is set, counting will restart, using the new initial-count value”

Interrupt Priority

- Each interrupt vector is an 8-bit value
  - The interrupt-priority class is the value of bits 7:4: the lowest interrupt-priority class is 1 and the highest is 15
  - Interrupts with vectors in the range 0–15 (with interrupt-priority class 0) are illegal and are never delivered
    - Because vectors 0–31 are reserved, software should configure interrupt vectors to use interrupt-priority classes in the range 2–15
- ***Example***: any interrupt using vectors 0x40 through 0x4F have the same priority
  - If you block at 0x43, you block ***all*** interrupts 0x40 through 0x4F

Interrupt Priority

- The task priority register allows to set a priority threshold for interrupting the processor
  - This mechanism enables the OS to temporarily block low priority interrupts from disturbing high-priority work that the processor is doing
  - ***Example:*** if you want to block all interrupts through 0x3F, just write 0x30, 0x31, … or 0x3F to the task priority register

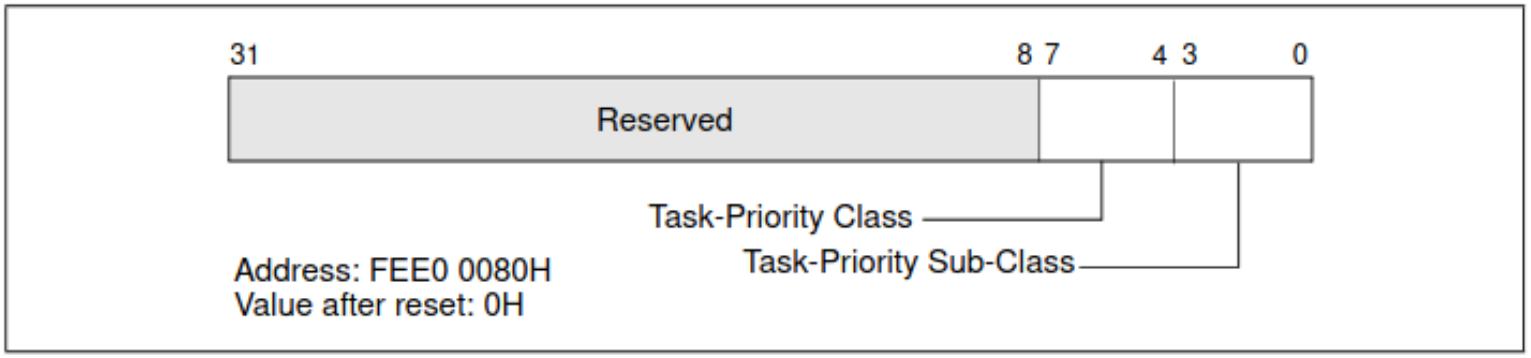


Figure 10-18. Task-Priority Register (TPR)

* Taken from https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html