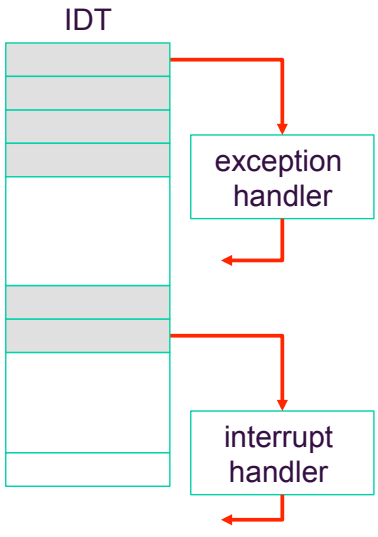
Exceptions and Interrupts on the x86

- Handling Intel Processor Exceptions
  - the Interrupt Descriptor Table (IDT)
- Exception Handler Framework
- Handling Hardware Device Interrupts old-fashioned style
  - IRQs and the 8259 Programmable Interrupt Controller.
- Interrupt Handler Framework

Interrupt Descriptor Table

- Interrupt Vector Table x86-style:
  - processor exceptions, hardware interrupts, software interrupts
- 256 entries: Each entry contains address of interrupt handler (**interrupt service routine**).
- The first 32 entries reserved for processor **exceptions** (division by zero, page fault, etc.)
- **Hardware interrupts** can be mapped to any of the other entries using the Programmable Interrupt Controller (e.g. 8259 PIC, see later)



Processor Exceptions

Exception #	Description	Error Code?
0	Division By Zero Exception	No
1	Debug Exception	No
2	Non Maskable Interrupt Exception	No
3	Breakpoint Exception	No
4	Into Detected Overflow Exception	No
5	Out of Bounds Exception	No
6	Invalid Opcode Exception	No
7	No Coprocessor Exception	No
8	Double Fault Exception	**Yes**
9	Coproc. Segment Overrun Exception	No
10	TSS Exception	**Yes**
11	Segment Not Present Exception	**Yes**
12	Stack Fault Exception	**Yes**
13	General Protection Fault Exception	**Yes**
14	Page Fault Exception	**Yes**
15	Unknown Interrupt Exception	No
16	Coprocessor Fault Exception	No
17	Alignment Check Exception (486+)	No
18	Machine Check Exception (Pentium/586+)	No
19 to 31	Reserved Exceptions	No

Handling Exceptions

```
global _isr0
global _isr1
global _isr2

...

global _isr8

…

global _isr30
global _isr31
```

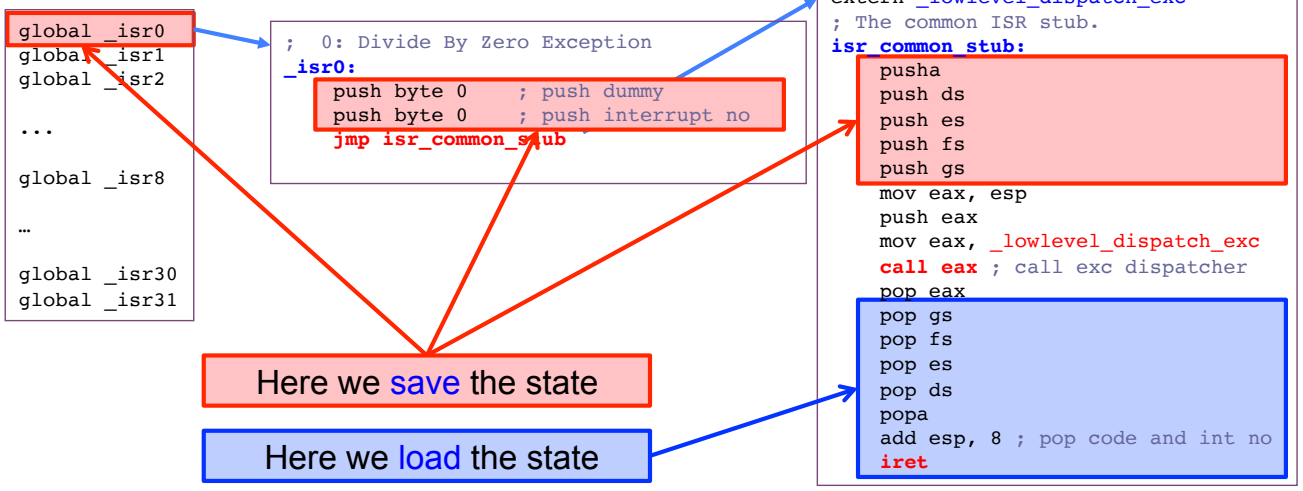
```
;  0: Divide By Zero Exception
_isr0:
    push byte 0     ; push dummy
    push byte 0     ; push interrupt no
    jmp isr_common_stub
```

```
;  8: Double Fault Exception (Error Code!)
_isr8:
    ; don't push dummy!
    push byte 8       ; push interrupt no
    jmp isr_common_stub
```

```
extern _lowlevel_dispatch_exc
; The common ISR stub.
isr_common_stub:
    pusha
    push ds
    push es
    push fs
    push gs
    mov eax, esp
    push eax
    mov eax, _lowlevel_dispatch_exc
    call eax
    pop eax
    pop gs
    pop fs
    pop es
    pop ds
    popa
    add esp, 8 ; pop code and int no
    iret
```

```
/* This defines what the stack looks like when the
   exception/interrupt reaches the exception dispatcher. */
typedef struct regs
{
    unsigned int gs, fs, es, ds;      /* pushed in common stub */
    unsigned int edi, esi, ebp, esp,
                 ebx, edx, ecx, eax; /* pushed by pusha      */
    unsigned int int_no, err_code;    /* pushed in _isrXX     */
    unsigned int eip, cs, eflags;     /* pushed by processor */
} REGS;
```

Handling Exceptions



High-Level Exception Handler

```
extern "C" void lowlevel_dispatch_exception (REGS * _r) {
    ExceptionHandler::dispatch_exception(_r);
}
```

```
class ExceptionHandler {
  static ExceptionHandler * handler_table[EXCEPTION_TABLE_SIZE];
  static void register_handler(unsigned int _isr_code,/* POPULATE DISPTCHER TABLE */
                               ExceptionHandler * _handler);
  static void dispatch_exception(REGS * _r);          /* DISPATCHER */
  virtual void handle_exception(REGS * _regs);        /* HANDLE EXCEPTION (abstract)*/
};
```

```
void ExceptionHandler::dispatch_exception(REGS * _r) {
    unsigned int exc_no = _r->int_no;
    ExceptionHandler * handler = handler_table[exc_no];
    if (handler == NULL) {
      Console::puts("NO EXC. HANDLER REGISTERED\n");
      panic();
    }
    handler->handle_exception(_r);
}
```

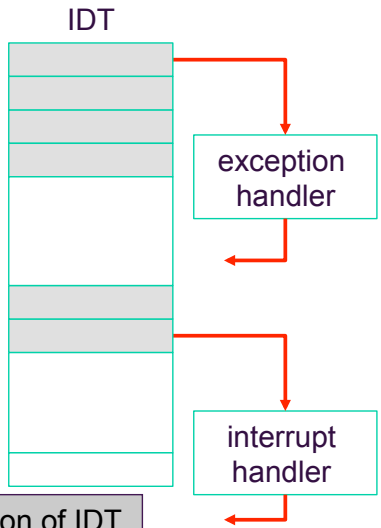
Example: Divide-by-Zero Exception Handler

```
/* -- EXAMPLE OF AN EXCEPTION HANDLER: Divide-by-Zero  -- */

class DBZ_Handler : public ExceptionHandler {
  /* We derive Divide-by-Zero handler from ExceptionHandler
     and overload the method handle_exception. */
  public:
    virtual void handle_exception(REGS * _regs) {
        Console::puts("DIVISION BY ZERO!\n");
        panic();
    }
};
```

```
/* -- IN THE KERNEL ... */
/* Define DBZ handler */
DBZ_Handler dbz_handler;

/* Register DBZ handler for Exception #0
   with the exception dispatcher. */
ExceptionHandler::register_handler(0, &dbz_handler);
```

Initializing the IDT

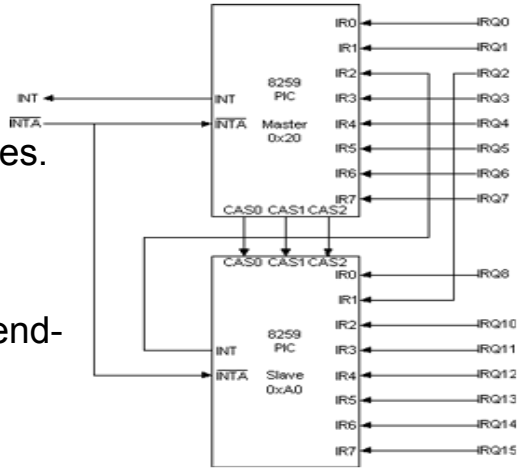
```
extern "C" void isr0(); /* _isr0 in assembly */
extern "C" void isr1();
/*…*/
extern "C" void isr31();

static void ExceptionHandler::init_dispatcher() {
  /* Add any new ISRs to the IDT here. */
  idt_set_gate( 0, (unsigned) isr0, 0x08, 0x8E);
  idt_set_gate( 1, (unsigned) isr1, 0x08, 0x8E);
  /* … */
  idt_set_gate(31, (unsigned) isr31, 0x08, 0x8E);

  /* Initialize high-level exception handler */
  for(int i = 0; i < EXCEPTION_TABLE_SIZE; i++) {
    handler_table[i] = NULL;
  }
}
```



For details of **`idt_set_gate()`** and installation of IDT check out the code (file **`idt.C`**).

Hardware Interrupts: The Programmable Interrupt Controller (PIC)

- Interrupt control on PC/AT by cascaded (master/slave) pair of 8259 PICs.
- Interrupts (IRQs) are mapped to IDT entries.
  - Typically inconvenient ones, can be remapped.
- Interrupt service routine must send EOI (end-of-interrupt) signal to PIC
  - to both PICs if interrupt from slave



Assigned Interrupt Lines in the PIC

Master 8259
- IRQ0 – Intel 8253 or Intel 8254 PIT, aka the system timer
- IRQ1 – Intel 8042 keyboard controller
- IRQ2 – not assigned in PC/XT; cascaded to slave 8259 INT line in PC/AT
- IRQ3 – 8250 UART serial port COM2 and COM4
- IRQ4 – 8250 UART serial port COM1 and COM3
- IRQ5 – hard disk controller in PC/XT; Intel 8255 parallel port LPT2 in PC/AT
- IRQ6 – Intel 82072A floppy disk controller
- IRQ7 – Intel 8255 parallel port LPT1 / spurious interrupt

Slave 8259 (PC/AT and later only)
- IRQ8 – real-time clock (RTC)
- IRQ9 – no common assignment
- IRQ10 – no common assignment
- IRQ11 – no common assignment
- IRQ12 – Intel 8042 PS/2 mouse controller
- IRQ13 – math coprocessor
- IRQ14 – hard disk controller 1
- IRQ15 – hard disk controller 2

Handling Hardware Interrupts

```
global _irq0
global _irq1
global _irq2
global _irq3

/*… */



global _irq14
global _irq15
```

```
; 32: IRQ0 – start at IDT entry 32
_irq0:
    push byte 0
    push byte 32
    jmp irq_common_stub
```

```
; 33: IRQ1
_irq1:
    push byte 0
    push byte 33
    jmp irq_common_stub
```

```
extern _lowlevel_dispatch_interrupts
/* similar to ISRs */
irq_common_stub:
    pusha
    push ds
    push es
    push fs
    push gs
    mov eax, esp
    push eax
    mov eax, _lowlevel_dispatch_interrupts
    call eax
    pop eax
    pop gs
    pop fs
    pop es
    pop ds
    popa
    add esp, 8
    iret
```

High-level Interrupt Handler

```
extern _lowlev
/* similar to
irq_common_stu
    pusha
    push ds
    push es
    push fs
    push gs
    mov eax, e
    push eax
    mov eax, _
    call eax
    pop eax
    pop gs
    pop fs
    pop es
    pop ds
    popa
    add esp, 8
    iret
```

```
static void InterruptHandler::dispatch_interrupt(REGS * _r) {
  unsigned int int_no = _r->int_no - IRQ_BASE;
  InterruptHandler * handler = handler_table[int_no];
  if (handler == NULL) {
     Console::puts("NO DEFAULT INTERRUPT HANDLER REGISTERED\n");
     panic();
  }
  handler->handle_interrupt(_r);
  /* This is an interrupt that was raised by the interrupt controller. We need
     to send and end-of-interrupt (EOI) signal to the controller after the
     interrupt has been handled. */
  /* Check if the interrupt was generated by the slave interrupt controller.
     If so, send an End-of-Interrupt (EOI) message to the slave controller. */
  if (generated_by_slave_PIC(int_no)) { /* i.e. int_no < 8 */
       outportb(0xA0, 0x20);
  }
  /* Send an EOI message to the master interrupt controller. */
  outportb(0x20, 0x20);
}
```

```
extern "C" void lowlevel_dispatch_interrupt(REGS * _r) {
    InterruptHandler::dispatch_interrupt(_r);
}
```

Interrupt Handler Example: Periodic Timer

```
class SimpleTimer : public InterruptHandler {
  int           ticks;   /* ticks since last time update. */
  unsigned long seconds; /* How long has the system been running? */
  /* At what frequency do we update the ticks counter? */
  int hz; /* By defaults it is 18.22Hz. In this way,
             a 16-bit counter wraps around every hour. */
  void set_frequency(int _hz);
  /* Set the interrupt frequency for the simple timer. */
public:
  SimpleTimer(int _hz);
  /* Initialize a simple timer, and set its frequency. */
  virtual void handle_interrupt(REGS *_r);
  /* This must be registered as the interrupt handler for the timer
     when the system gets initialized. (e.g. in "kernel.C") */
};
```

Interrupt Handler Example: Periodic Timer

```
void SimpleTimer::SimpleTimer(int _hz) {
    seconds = 0;
    ticks   = 0;
    set_frequency(_hz);
}
```

```
class SimpleTimer : public InterruptHandler {
  int           ticks;   /* ticks since last time u
  unsigned long seconds; /* How long has the systen
  /* At what frequency do we update the ticks counte
  int hz; /* By defaults it is 18.22Hz. In this way,
             a 16-bit counter wraps around every hour. */
  void set_frequency(int _hz);
  /* Set the interrupt frequency for the simple timer. */
public:
  SimpleTimer(int _
  /* Initialize a s
  virtual void hand
  /* This must be r
     when the syste
};
```

```
/* Set the interrupt frequency for the simple timer.                */
/* Preferably set this before installing the timer handler!         */
void set_frequency(int _hz) {
  hz = _hz;                          /* Remember the frequency.     */
  int divisor = 1193180 / _hz;       /* The input clock runs at 1.19MHz  */
  Machine::outportb(0x43, 0x34);     /* Set command byte to be 0x34.     */
  Machine::outportb(0x40, divisor & 0xFF); /* Set low byte of divisor.*/
  Machine::outportb(0x40, divisor >> 8); /* Set high byte of divisor. */
}
```

Interrupt Handler Example: Periodic Timer

```
class SimpleTimer : public
  int           ticks;   /
  unsigned long seconds; /
  /* At what frequency do
  int hz; /* By defaults i
             a 16-bit coun
  void set_frequency(int _
  /* Set the interrupt fre
public:
  SimpleTimer(int _hz);
  /* Initialize a simple t
  virtual void handle_interrupt(REGS *_r);
  /* This must be registered as the interrupt handler for the timer
     when the system gets initialized. (e.g. in "kernel.C") */
};
```

```
void handle_interrups(REGS *r) {
    /* Increment our "ticks" count */
    ticks++;
    /* Whenever a second is over, we update counter. */
    if (ticks >= hz ) {
        seconds++;
        ticks = 0;
        Console::puts("One second has passed\n");
    }
}
```

Interrupt Handler Example: Periodic Timer

```
class SimpleTimer : public InterruptHandler
  int           ticks;   /* ticks since las
  unsigned long seconds; /* How long has th
  /* At what frequency do we update the tic
  int hz; /* By defaults it is 18.22Hz. In
             a 16-bit counter wraps around
  void set_frequency(int _hz);
  /* Set the interrupt frequency for the si
public:
  SimpleTimer(int _hz);
  /* Initialize a simple timer, and set its
  virtual void handle_interrupt(REGS *_r);
  /* This must be registered as the interru
     when the system gets initialized. (e.g
};
```

```
void main() { /* Sample Kernel */
    GDT::init();
    Console::init();
    IDT::init();
    ExceptionHandler::init_dispatcher();
    IRQ::init();
    InterruptHandler::init_dispatcher();

    SimpleTimer timer(100);
    /* set timer ticks to 10ms. */
    reg_int_handler(0, &timer);

    Machine::enable_interrupts();
    Console::puts("Hello World!\n");
    /* continue here ... */
}
```

Summary: Interrupts and Exceptions on the x86

- Handling Intel Processor Exceptions
- the Interrupt Descriptor Table (IDT)
- Exception Handler Framework
- Example Exception Handler:
  - Division-by-Zero Handler
- Handling Hardware Interrupts “old-fashioned style”
  - IRQs and the 8259 Programmable Interrupt Controller.
- Interrupt Handler Framework
- Example Interrupt Handler:
  - Programmable Interval Timer