

The K Project

Interrupt and Exception Handling

LSE Team

EPITA

May 06, 2019

- *Exception* : Synchronous with program execution (e.g. division by zero, accessing an invalid address)
- *Interrupt* : Asynchronous with program execution. Generated by devices external to the CPU

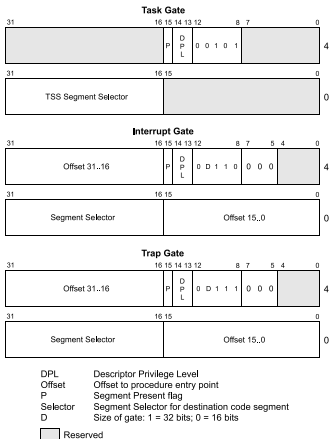


Figure: IDT

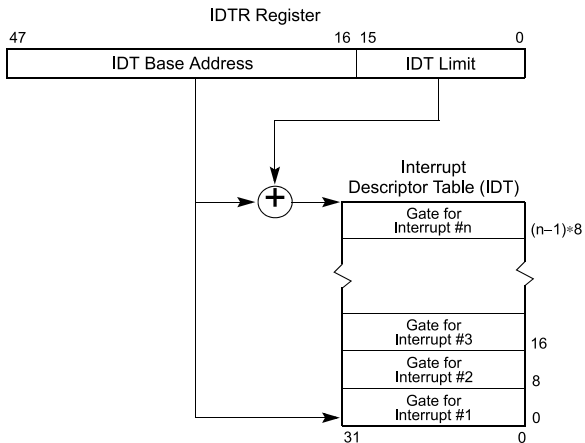


Figure: IDTR

```
idt_r idtr;

idtr.base = idt;           /* idt base address */
idtr.limit = sizeof(idt) - 1; /* idt size - 1 */

__asm__ volatile("lidt %0\n"
                 : /* no output */
                 : "m" (idtr)
                 : "memory");
```

Int.	Description
0	Divide by 0
1	Debug
2	NMI
3	Breakpoint
4	Overflow
5	Bound Range Exceeded
6	Invalid Opcode

The K Project

LSE Team

Introduction

Interrupt
Descriptor
TableInterrupt
Request

Keyboard

Timer

Conclusion

Int.	Description
7	Device Not Available
8	Double Fault
9	Coprocessor
10	Invalid TSS
11	Segment not present
12	Stack Segment Fault
13	General Protection
14	Page Fault

Int.	Description
15	(Intel reserved)
16	x87 FPU (Math Fault)
17	Alignment Check
18	Machine Check
19	SMD Floating Point Ex.
20	Virtualization Ex.
21-31	(Intel reserved)
32-255	User Defined


```
int $3      ; Breakpoint
int $0x80   ; Syscall
```

Intel's definition, Vol2a

"The INT n instruction generates a call to the interrupt or exception handler specified with the destination operand"

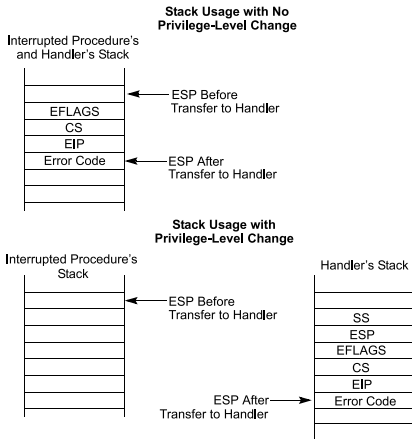


Figure: Stack Usage

```
isr:
    ;<save registers>
    pushl %esp
    call generic_c_handler
    add $4, %esp
    ;<restore registers>
    add $8, %esp
    iret
```

```
.global isr_keyboard
isr_keyboard:
    pushl $0
    pushl $33 ; IRQ1
    jmp isr
```

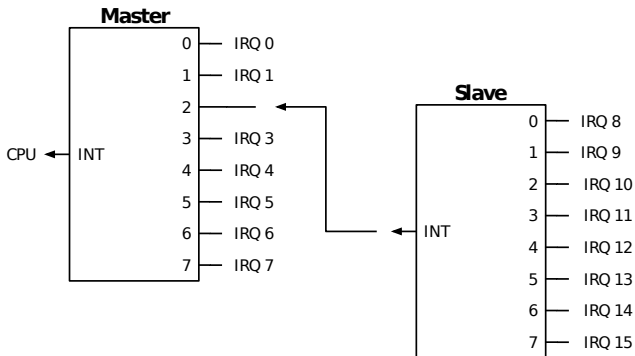


Figure: PIC

- IRQ0 - PIT
- IRQ1 - Keyboard
- IRQ2 - Not assigned in PC/XT; cascaded to slave 8256
- IRQ3 - UART (COM2 and COM4)
- IRQ4 - UART (COM1 and COM3)
- IRQ5 - Hard disk in PC/XT; Parallel port LPT2 in PC/AT
- IRQ6 - Floppy disk controller
- IRQ7 - Parallel port LPT1

- IRQ8 - RTC
- IRQ9 -
- IRQ10 -
- IRQ11 -
- IRQ12 - PS/2 mouse controller
- IRQ13 - Math coprocessor
- IRQ14 - Hard disk controller 1
- IRQ15 - Hard disk controller 2

- 0x20, the master PIC's port A
- 0x21, the master PIC's port B
- 0xA0, the slave PIC's port A
- 0xA1, the slave PIC's port B

```

0 0 0 1 x 0 x x
      |  | |
      |  | +----- (1)
      |  +----- (2)
      +----- (3)

```

- 1 ICW4 **present** (set) or not (clear)
- 2 single controller (set) or **cascade** mode (clear)
- 3 level triggered mode (set) or **edge triggered** mode (clear)


```
x x x x x 0 0 0
| | | | | | | |
+----- (1)
```

1 Interrupt vector base address

Master PIC

```

x x x x x x x x
| | | | | | | |
+----- (1)

```

Slave PIC

```

0 0 0 0 0 x x x
          | | |
          +----- (2)

```

- 1** For each bit, indicate whether a slave PIC is connected to this pin (set) or not (clear)
- 2** Indicate to the slave his slave ID (which pin of the master it is connected to)

```

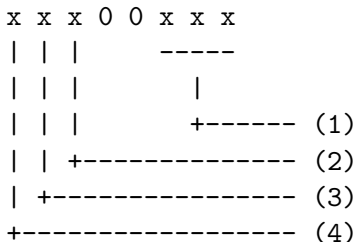
0 0 0 x x x x 1
      | --- |
      | | +----- (1)
      | +----- (2)
      +----- (3)

```

- 1 Automatic (set) EOI or **normal** (clear) EOI
- 2 Buffering mode (**no buffering**)
- 3 Special mode fully nested (set) or **not** (clear)

```
x x x x x x x x
| | | | | | | |
+----- (1)
```

- 1 For each bit, indicate whether the corresponding IRQ is masked (set) or not (clear)



- 1** Interrupt level to be acted upon when sending a specific command
- 2** Send an EOI (end of interrupt command) (set)
- 3** Send a specific (set) or a non-specific (clear) command
- 4** Rotate priorities (set) or not (clear)

- Write IDT management functions
 - allocate/clean IDT
 - set an interrupt gate in the IDT
 - set a trap gate in the IDT
- Write the context saving/restoring routines in assembly code
- Implement the exceptions and interrupts wrappers
- Write a function which builds the IDT and loads it
- Initialize the PIC
 - send ICWs to both master and slave PICs
 - mask all interrupts
- Write very simple debug handlers like:
 - printing an error message when executing a division by zero
 - printing a string when a key is pressed
- Do not forget to enable hardware interrupts using `sti`

- You forgot to save or restore some registers in your context
- You did not consider that gcc automatically generates the prolog/epilog for C functions
- The stack is misaligned when `iret` is executed
- PICs are not acknowledged after returning from an ISR (Interrupt SubRoutine)

The K Project

LSE Team

Introduction

Interrupt
Descriptor
Table

Interrupt
Request

Keyboard

Timer

Conclusion

- 0x60: I/O buffer
- 0x64: Status register

The K Project

LSE Team

Introduction

Interrupt
Descriptor
TableInterrupt
Request

Keyboard

Timer

Conclusion

```

X X X X X X X X
| | | | | | | |
| | | | | | | +----- Output buffer full
| | | | | | | +----- Input buffer full
| | | | | | +----- System flag
| | | | | +----- Command/Data
| | | +----- Keyboard inhibit
| | +----- Auxiliary device output buffer
| +----- General purpose time-out
+----- Parity error

```

The K Project

LSE Team

Introduction

Interrupt
Descriptor
TableInterrupt
Request

Keyboard

Timer

Conclusion

X X X X X X X X

| -----

| |

| +----- Key number

+----- Key press (clear) or release (set)

- Ensure that the event manager is working
- Initialize driver
 - Add the keyboard interrupt gate to the IDT
 - Update the PIC to unmask the IRQ line 1
- Write a very simple keyboard handler:
 - Read scancodes from I/O port 0x60
 - Extra features such as modifier keys or simultaneous key strikes support are not mandatory
- Implement `getkey()`
- You **must** implement a queue for the keyboard buffer

- Execution context may be corrupted.
- Hardware interrupts may have been disabled with `cli` or may not have been enabled with `sti`
- The keyboard interrupt may be masked in the PIC
- The previous IRQ has not been ACKED
- The previous key has not been fetched

- **Counter 0:** fire an interrupt at a user-defined frequency.
- Counter 1: historically used in order to periodically refresh the RAM, but it not used anymore.
- Counter 2: linked with the PC speaker, so you can use it in order to generate sound

- Mode 0: Interrupt on terminal count
- Mode 1: hardware retriggerable one-shot
- **Mode 2**: rate generator
- Mode 3: square generator
- Mode 4: Software Triggered Strobe
- Mode 5: Hardware Triggered Strobe

c	c	io	io	m	m	m	b	
----		-----		-----				
							+-	(1)
				+-----				(2)
		+-----						(3)
+-----								(4)

- 1 Binary counter (**unset**) or BCD counter (set)
- 2 Mode to use
- 3 Registers read/write policy (LSB first then MSB)
- 4 Counter to setup

The K Project

LSE Team

Introduction

Interrupt
Descriptor
Table

Interrupt
Request

Keyboard

Timer

Conclusion

- 0x40 : Counter 0
- 0x41 : Counter 1
- 0x42 : Counter 2
- 0x43 : Control Register

- Write into control register
- Write the divider on the counter register

$$\text{divider} = \frac{\text{internal_frequency}}{\text{desired_frequency}}$$

- Internal frequency : 1193182

```
unsigned long gettick(void);
```

- Counter 0
- Mode 2
- Interrupt rate : 100 Hz

- Execution context may be corrupted.
- Hardware interrupts may have been disabled with `cli` or may not have been enabled with `sti`
- The timer interrupt may be masked in the PIC.

The K Project

LSE Team

Introduction

Interrupt
Descriptor
Table

Interrupt
Request

Keyboard

Timer

Conclusion

- Use X Macro

The K Project

LSE Team

Introduction

Interrupt
Descriptor
Table

Interrupt
Request

Keyboard

Timer

Conclusion

- `k[at]lse.epita.fr`
- `labos.lse` with `[K]` tag
- `#k` (`irc.rezosup.org`)
- `guillaume.pagnoux[at]lse.epita.fr`
- `tom.decrette[at]lse.epita.fr`