

IDT Initialisation

8259A PIC Initialisation

The x86_64 architecture supports hardware interrupts. When the CPU is interrupted, it expects the currently running software to handle them using Interrupt Service Routines (ISRs). In Protected Mode, they are stored in a table of 256 entries called an Interrupt Descriptor Table. These Interrupt ReQuests (IRQs) and exceptions are sent through specialised hardware.

The 8259A Programmable Interrupt Controller (PIC) is such hardware. It must be properly initialised before handling kernel code because it will accidentally trigger Protected Mode exceptions otherwise.

By default, the PIC possesses 2 parts but can include more depending on processors.

The Master PIC is communicated with using the command port 0x20 and the data port 0x21.

The Slave PIC is communicated with using the command port 0xA0 and the data port 0xA1.

Each of them controls Interrupt ReQuests (IRQ) ranging from 0-7 and 8-15 respectively.

When the CPU switches into Protected Mode, they therefore overlap with the interrupt vectors of the exceptions present in this mode because the first 0x1F (32) values are reserved for them.

As such, we must remap them to unused IDT entries (we chose entry 33 through 48).

To interface with the PIC, communication must be established using Initialisation Control Words (ICW) represented by a byte value. Each bit corresponds to parameters according to the tables located [here](#).

The following code has been written in the file `./src/boot/main64.asm`.

```
; Initialisation control word 1 (ICW1)
mov al, 0x11 ; Bit 0 set means ICW4 is expected during
```

initialisation.

```
        ; Bit 4 set means initialisation will be made.
out 0x20, al ; Sends the initialise command (0x10) to Master PIC
and makes it wait for additional information (0x01)
out 0xA0, al ; Sends the initialise command (0x10) to Slave PIC
and makes it wait for additional information (0x01)

; ICW2 (In 80x86 mode, it defines the interrupt vector addresses).
mov al, 0x20 ; Sets IRQ0 to the 33rd entry of the IDT (It
automatically sets up IRQ1-7 to subsequent entries).
out 0x21, al
mov al, 0x28 ; Sets IRQ8 to the 41st entry of the IDT (It
automatically sets up IRQ9-15 to subsequent entries).
out 0xA1, al

; ICW3
mov al, 4    ; Tells Master PIC that there's a slave PIC at IRQ2
(It uses bit notation to determine the IRQ Line. 4 is 0b100 in
binary, which means the second bit is flipped. So, it is IRQ2).
out 0x21, al; IRQ2 was chosen because the x86 architecture uses
IRQ2 for communication between the master and the slave.
mov al, 2    ; Tells Slave PIC its cascade identity. Only bit 0 to
2 are used to communicate.
out 0xA1, al ; To pass the IR line that the master must
communicate on, 3bit value ranging from 0 to 7 must be passed.
Since IR line 2 is used for communication, we send 2 to the
master.

; ICW4
mov al, 1    ; Bit 0 enables 8086 mode.
out 0x21, al ; By default, it works in 8080 mode which sends 3
bytes of data to the data bus equivalent to a call instruction in
8080 processors.
out 0xA1, al ; In 8086 mode, it sends an interrupt number.
```

Interrupt Service Routines (ISRs)

When an interruption/exception occurs, the CPU will read the IDT Table loaded in the IDT Register using the LIDT instruction. It will then load the address of the instruction which triggered the interruption in Control Register 2 and it will start execution of the instruction located at the address written in the IDT Table for the corresponding interrupt or exception.

C code in the file `./src/kernel/idt.c` is used to give each entries an handler.

The handlers themselves are written in assembly in the file `./src/boot/idt.asm`.

While we tried writing C code for the handlers, the compiler adds instructions which interfere with the handlers leading to Triple Faults which force a system reset.

These ASM functions first push a dummy value (if the interrupt/exception doesn't return any error code) and an identifier to the stack before jumping to the main procedure called **interrupt_resolver**. The latter first backs up all the registers to the stack because the CPU has to return to the same state it had before being interrupted after resolving the interruption.

This routine then calls a C function called **resolve_interrupt** written in `./src/kernel/idt.c`.

Using the calling convention of C, we add a pointer to the top of the stack into the rdi register (first argument). The **cpu_stack_t** structure defined in `./include/kernel/idt.h` allow us to retrieve the values from the top of the stack as well as the identifier and error pushed to the stack.

Using the identifier of the interrupt, we can redirect the error to the proper handler.

If the interrupt had no error, we can show a DEBUG message.

On return from this function, we force the rax register to hold the CPU state passed as argument to this function.

Finally, we pass the content of the rax register to the rsp register.

Doing this restores the values pushed to the stack before entering the **resolve_interrupt** function.

Popping the values from the stack in reverse order will then restore the registers to what they were before the interruption was triggered. Since the last two values still on the stack can't be passed to any registers, we add 16 to the rsp register to remove them as both of them are 8 byte values since the CPU is in 64bit mode.

The IRETQ instruction (Interrupt RETurn Quad) is the standard way to return from an interruption as it forces a far return and pops the IRET frame from the stack.

The IRET frame consists of values from the rip and rsp registers, the code and stack segment descriptors and the cpu flags and they are automatically pushed to the stack before the interruption is handled.

Once all the handlers are properly initialised, the LIDT instruction can then be used to load the IDT Table defined in the **idt_init** function from the **./src/kernel/idt.c** file into the IDT register.

Since we had deactivated the interrupts during 64bit mode initialisation, the STI instruction will re-enable them.

External Sources:

- [8259A PIC Documentation](#)
- [8259A PIC Initialisation Procedure](#)
- [8259A PIC OSDev](#)