# Interrupciones de Reloj y Teclado.

Manuel Ferreria $\longrightarrow$ Pablo Somodi

DC - FCEyN - UBA

27 de Octubre de 2015

# Volviendo a interrupciones...

# Volviendo a interrupciones...

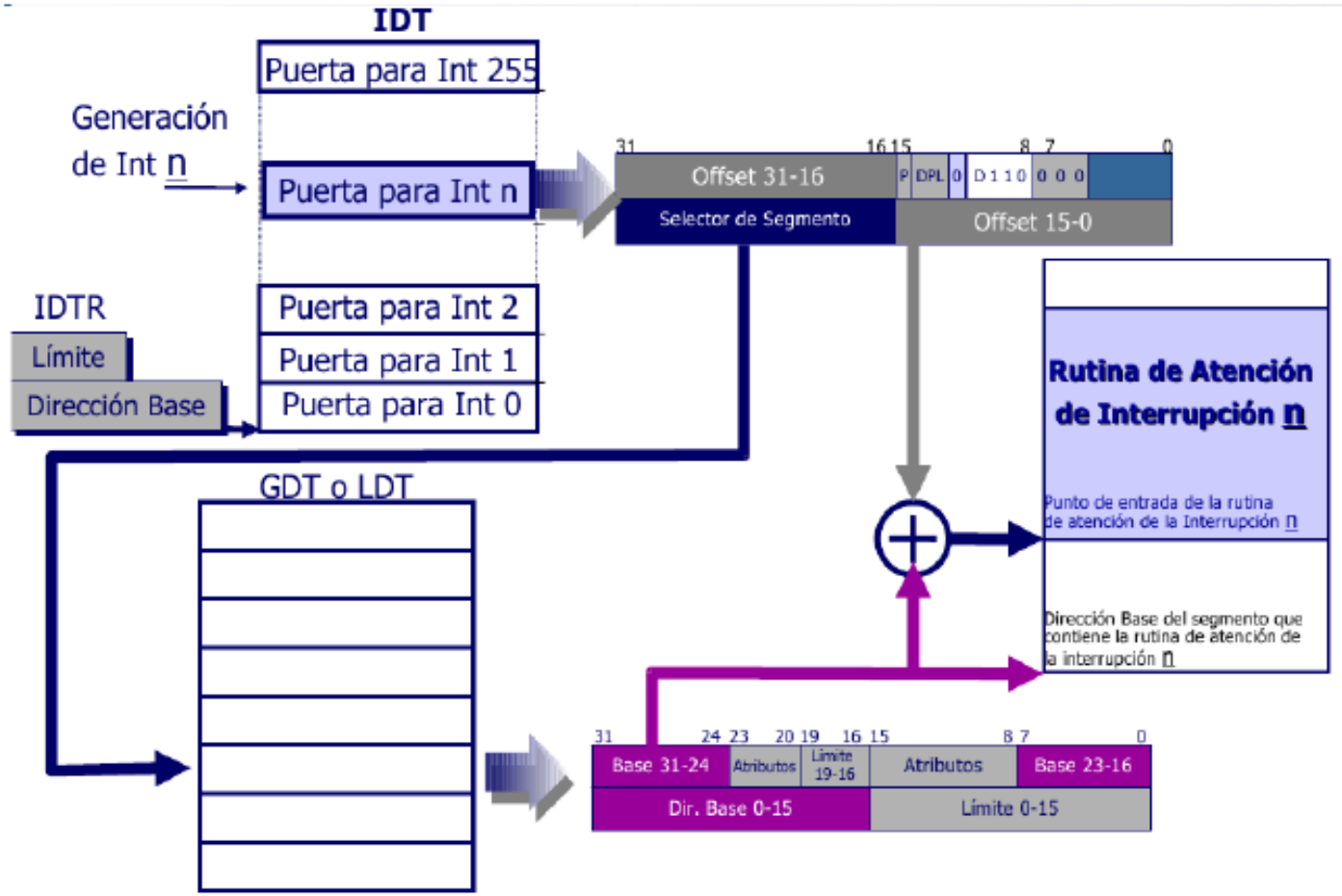
IDT:

- Almacena descriptores de interrupción.
- Su dirección se almacena en el registro IDTR.

# Volviendo a interrupciones...

IDT:

- Almacena descriptores de interrupción.
- Su dirección se almacena en el registro IDTR.



| Tipo | Mnemónico | Descripción | Clase | Código de Error | Fuente |
|---|---|---|---|---|---|
| 0 | #DE | Error de División | Fault | NO | Instrucciones DIV e IDIV |
| 1 | #DB | Reservada | Fault/Trap | NO | Solo para uso de Intel |
| 2 | - | NMI | Interrupción | NO | Interrupción No enmascarable. Pin NMI |
| 3 | #BP | BreackPoint | Trap | NO | Opcode 0xCC |
| 4 | #OF | Overflow | Trap | NO | Instrucción INTO |
| 5 | #BR | BOUND Range Exceeded | Fault | NO | Instrucción BOUND |
| 6 | #UD | Invalid Opcode | Fault | NO | Instrucción UD2 u Opcode Reservado |
| 7 | #NM | Coprocesador No disponible | Fault | NO | Instrucciones de Punto Flotante o WAIT / FWAIT |
| 8 | #DF | Doble Fault | Abort | SI (Cero) | Cualquier instrucción capaz de generar una excepción, una señal en NMI o en INTR |
| 9 | | Coprocessor Segment Overrun (reservada) | Fault | SI | Instrucciones de Punto Flotante |
| 10 | #TS | TSS Inválido | Fault | SI | Task switch o acceso a un TSS |
| 11 | #NP | Segmento No Presente | Fault | SI | Carga o acceso a un registro de segmento |
| 12 | #SS | Falta en el Stack Segment | Fault | SI | Operacioes de Pila y Carga del registro SS |
| 13 | #GP | General Protection | Fault | SI | Cualquier referencia a memoria y otros chequeos de protección |
| 14 | #PF | Page Fault | Fault | SI | Cualquier referencia a memoria |
| 15 | - | Reservada por Intel (no usar) | NO | | |
| 16 | #MF | X-87 FPU Error de Punto Flotante | Fault | NO | Instrucción de la FPU o WAI/FWAIT |
| 17 | #AC | Alignment Check | Fault | SI (Cero) | Cualquier referencia de datos a memoria |
| 18 | #MC | Machine Check | abort | NO | Los Códigos de error si hubiese, así como su fuente, depende del modelo de procesador |
| 19 | #XF | Excepción SIMD Floating Point | Fault | NO | Cualquier instrucción SSEx |
| 20-31 | - | Reservada por Intel (no usar) | | | |
| 32-255 | - | A definir por el usuario | Interrupción | | Interrupciones externas o Instrucción INTn |

| Tipo | Mnemónico | Descripción | Clase | Código de Error | Fuente |
|---|---|---|---|---|---|
| 0 | #DE | Error de División | Fault | NO | Instrucciones DIV e IDIV |
| 1 | #DB | Reservada | Fault/Trap | NO | Solo para uso de Intel |
| 2 | - | NMI | Interrupción | NO | Interrupción No enmascarable. Pin NMI |
| 3 | #BP | BreackPoint | Trap | NO | Opcode 0xCC |
| 4 | #OF | Overflow | Trap | NO | Instrucción INTO |
| 5 | #BR | BOUND Range Exceeded | Fault | NO | Instrucción BOUND |
| 6 | #UD | Invalid Opcode | Fault | NO | Instrucción UD2 u Opcode Reservado |
| 7 | #NM | Coprocesador No disponible | Fault | NO | Instrucciones de Punto Flotante o WAIT / FWAIT |
| 8 | #DF | Doble Fault | Abort | SI (Cero) | Cualquier instrucción capaz de generar una excepción, una señal en NMI o en INTR |
| 9 | | Coprocessor Segment Overrun (reservada) | Fault | SI | Instrucciones de Punto Flotante |
| 10 | #TS | TSS Inválido | Fault | SI | Task switch o acceso a un TSS |
| 11 | #NP | Segmento No Presente | Fault | SI | Carga o acceso a un registro de segmento |
| 12 | #SS | Falta en el Stack Segment | Fault | SI | Operacioes de Pila y Carga del registro SS |
| 13 | #GP | General Protection | Fault | SI | Cualquier referencia a memoria y otros chequeos de protección |
| 14 | #PF | Page Fault | Fault | SI | Cualquier referencia a memoria |
| 15 | - | Reservada por Intel (no usar) | NO | | |
| 16 | #MF | X-87 FPU Error de Punto Flotante | Fault | NO | Instrucción de la FPU o WAI/FWAIT |
| 17 | #AC | Alignment Check | Fault | SI (Cero) | Cualquier referencia de datos a memoria |
| 18 | #MC | Machine Check | abort | NO | Los Códigos de error si hubiese, así como su fuente, depende del modelo de procesador |
| 19 | #XF | Excepción SIMD Floating Point | Fault | NO | Cualquier instrucción SSEx |
| 20-31 | - | Reservada por Intel (no usar) | | | |
| 32-255 | - | A definir por el usuario | Interrupción | | Interrupciones externas o Instrucción INTn |

Lo que vimos

| Tipo | Mnemónico | Descripción | Clase | Código de Error | Fuente |
|---|---|---|---|---|---|
| 0 | #DE | Error de División | Fault | NO | Instrucciones DIV e IDIV |
| 1 | #DB | Reservada | Fault/Trap | NO | Solo para uso de Intel |
| 2 | - | NMI | Interrupción | NO | Interrupción No enmascarable. Pin NMI |
| 3 | #BP | BreackPoint | Trap | NO | Opcode 0xCC |
| 4 | #OF | Overflow | Trap | NO | Instrucción INTO |
| 5 | #BR | BOUND Range Exceeded | Fault | NO | Instrucción BOUND |
| 6 | #UD | Invalid Opcode | Fault | NO | Instrucción UD2 u Opcode Reservado |
| 7 | #NM | Coprocesador No disponible | Fault | NO | Instrucciones de Punto Flotante o WAIT / FWAIT |
| 8 | #DF | Doble Fault | Abort | SI (Cero) | Cualquier instrucción capaz de generar una excepción, una señal en NMI o en INTR |
| 9 | | Coprocessor Segment Overrun (reservada) | Fault | SI | Instrucciones de Punto Flotante |
| 10 | #TS | TSS Inválido | Fault | SI | Task switch o acceso a un TSS |
| 11 | #NP | Segmento No Presente | Fault | SI | Carga o acceso a un registro de segmento |
| 12 | #SS | Falta en el Stack Segment | Fault | SI | Operacioes de Pila y Carga del registro SS |
| 13 | #GP | General Protection | Fault | SI | Cualquier referencia a memoria y otros chequeos de protección |
| 14 | #PF | Page Fault | Fault | SI | Cualquier referencia a memoria |
| 15 | - | Reservada por Intel (no usar) | NO | | |
| 16 | #MF | X-87 FPU Error de Punto Flotante | Fault | NO | Instrucción de la FPU o WAI/FWAIT |
| 17 | #AC | Alignment Check | Fault | SI (Cero) | Cualquier referencia de datos a memoria |
| 18 | #MC | Machine Check | abort | NO | Los Códigos de error si hubiese, así como su fuente, depende del modelo de procesador |
| 19 | #XF | Excepción SIMD Floating Point | Fault | NO | Cualquier instrucción SSEx |
| 20-31 | - | Reservada por Intel (no usar) | | | |
| 32-255 | - | A definir por el usuario | Interrupción | | Interrupciones externas o Instrucción INTn |

*Lo que vemos hoy*

# Lo que vemos hoy:

## Interrupciones Externas (Enmascarables)

**Reloj:** La máquina posee un reloj interno que genera interrupciones a intervalos regulares de tiempo. Hoy veremos cómo capturar esa interrupción y hacer que se ejecute una rutina cada vez que esto sucede.

# Lo que vemos hoy:

## Interrupciones Externas (Enmascarables)

**Reloj:** La máquina posee un reloj interno que genera interrupciones a intervalos regulares de tiempo. Hoy veremos cómo capturar esa interrupción y hacer que se ejecute una rutina cada vez que esto sucede.

**Teclado:** También veremos cómo capturar las interrupciones generadas por el teclado, al presionar una tecla.

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.

   ```
   void _isrXX()
   ```

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.

   `void _isrXX()`

2. Se declaran en la IDT. ¿Cómo?

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.

   `void _isrXX()`

2. Se declaran en la IDT. ¿Cómo?

   `JMP clase3`

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.

   `void _isrXX()`

2. Se declaran en la IDT. ¿Cómo?

   `JMP clase3`

3. Se remapea el PIC. ¿Remapear que?

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.
   `void _isrXX()`
2. Se declaran en la IDT. ¿Cómo?
   `JMP clase3`
3. Se remapea el PIC. ¿Remapear que?
   Ya vamos a eso.

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

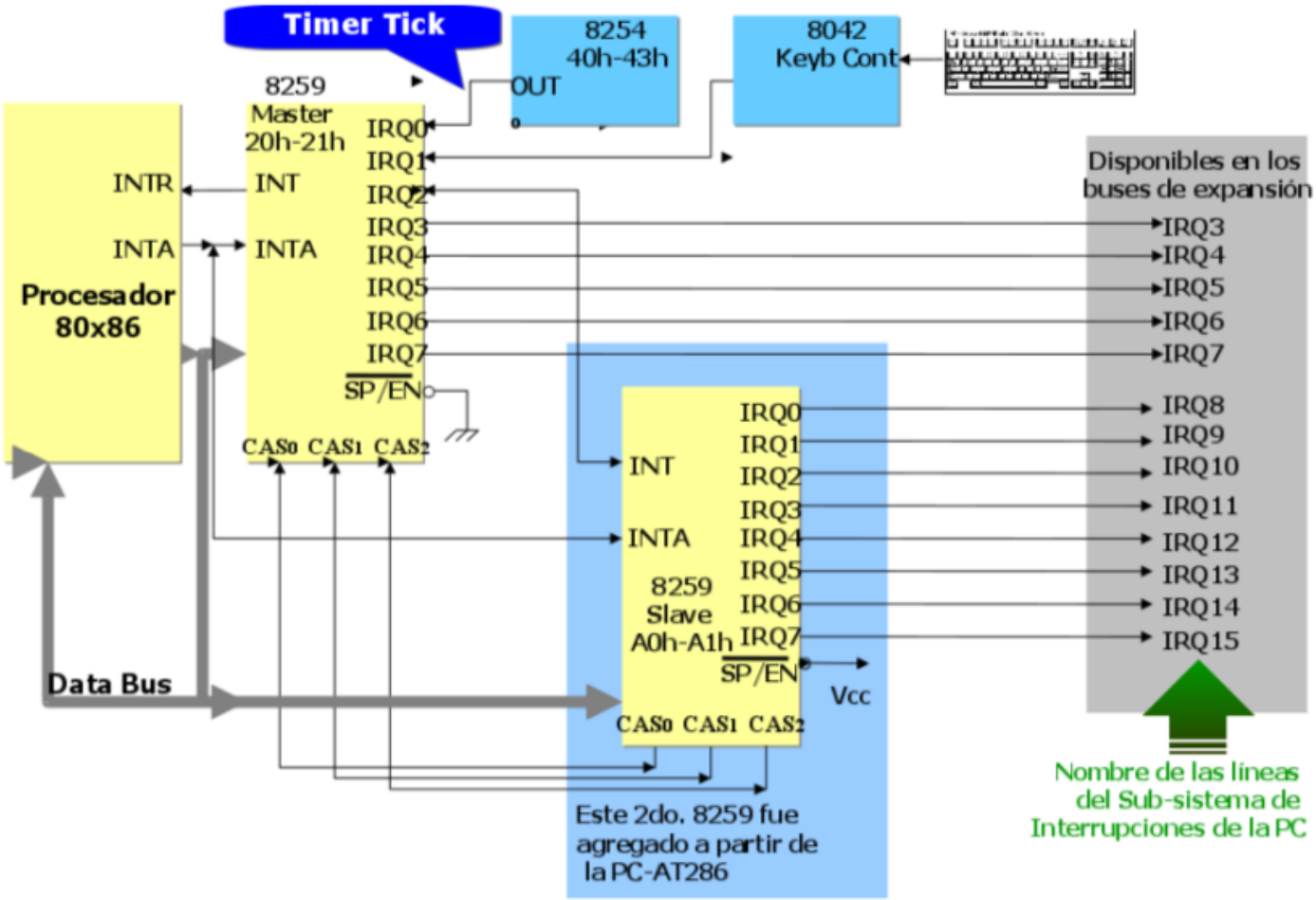
1. Se definen las rutinas de atención para cada interrupción.
   `void _isrXX()`
2. Se declaran en la IDT. ¿Cómo?
   `JMP clase3`
3. Se remapea el PIC. ¿Remapear que?
   Ya vamos a eso.
4. Se activan las interrupciones. ¿Cómo?

# ¿Cómo las manejamos?

## Igual que manejamos las excepciones:

1. Se definen las rutinas de atención para cada interrupción.
   `void _isrXX()`
2. Se declaran en la IDT. ¿Cómo?
   `JMP clase3`
3. Se remapea el PIC. ¿Remapear que?
   Ya vamos a eso.
4. Se activan las interrupciones. ¿Cómo?
   Se activa el flag **IF** del registro **EFLAGS**.
   Buscar la instrucción **sti** en el manual.

# ¿Que se remapea qué?



# El PIC

- Es un dispositivo (chip) al que le llegan las interrupciones de los demás dispositivos de la máquina.

# El PIC

- Es un dispositivo (chip) al que le llegan las interrupciones de los demás dispositivos de la máquina.
- Administra las interrupciones por prioridad, y las manda al procesador (i.e. la interrupción del reloj y la del teclado van primero).

# El PIC

- El PIC puede atender 15 interrupciones (IRQ0 - IRQ15, la IRQ2 no cuenta ya que es donde se conecta otro PIC en cascada). Por defecto, estas IRQs están mapeadas a las interrupciones 0x8 a 0xF (PIC1) y de 0x70 a la 0x77 (PIC2).

# El PIC

- El PIC puede atender 15 interrupciones (IRQ0 - IRQ15, la IRQ2 no cuenta ya que es donde se conecta otro PIC en cascada). Por defecto, estas IRQs están mapeadas a las interrupciones 0x8 a 0xF (PIC1) y de 0x70 a la 0x77 (PIC2).

- Como vimos, las interrupciones de la 0 a la 31 están reservadas para el procesador y, en particular, de la 8 a la 15 ya están ocupadas por las excepciones del mismo... Si se produce la interrupción, se llama al handler de la *excepción*.

# El PIC

- El PIC puede atender 15 interrupciones (IRQ0 - IRQ15, la IRQ2 no cuenta ya que es donde se conecta otro PIC en cascada). Por defecto, estas IRQs están mapeadas a las interrupciones 0x8 a 0xF (PIC1) y de 0x70 a la 0x77 (PIC2).

- Como vimos, las interrupciones de la 0 a la 31 están reservadas para el procesador y, en particular, de la 8 a la 15 ya están ocupadas por las excepciones del mismo... Si se produce la interrupción, se llama al handler de la *excepción*.

- Hay un conflicto, y para solucionarlo hay que "remapearlas".

# ¡Sorpresa!

- Las rutinas para hacer esto ya están hechas y listas para usar en el archivo **pic.h**. Éstas son: **resetear_pic** (remapeo), **habilitar_pic** y **deshabilitar_pic**.

# ¡Sorpresa!

- Las rutinas para hacer esto ya están hechas y listas para usar en el archivo **pic.h**. Éstas son: **resetear_pic** (remapeo), **habilitar_pic** y **deshabilitar_pic**.

- Por lo tanto:
  - Después de remapear el PIC y habilitarlo, tenemos que la interrupción de reloj está mapeada a la interrupción **32** y el teclado, a la **33**.
  - Resta habilitar las interrupciones utilizando la instrucción **sti**.

# Rutinas de atención - Esquema general

1. Preservar los registros que vayamos a romper (*¡la interrupción debe ser transparente!*)
2. Comunicar al PIC que ya se atendió la interrupción (EOI) (rutina **fin_intr_pic1**) del archivo **pic.h**.
3. Realizar la tarea correspondiente a la interrupción.
4. Restaurar los registros.
5. Retornar de la interrupción. (**iret**)

# Rutinas de atención - Esquema general

1. Preservar los registros que vayamos a romper (*¡la interrupción debe ser transparente!*)
2. Comunicar al PIC que ya se atendió la interrupción (EOI) (rutina **fin_intr_pic1**) del archivo **pic.h**.
3. Realizar la tarea correspondiente a la interrupción.
4. Restaurar los registros.
5. Retornar de la interrupción. (**iret**)

- NOTA: No es necesario deshabilitar las interrupciones al comienzo de la rutina, ya que el procesador lo hace automáticamente (si definimos el descriptor como un *interrupt gate*). Tampoco es necesario volver a habilitarlas al finalizar.

# Rutinas de atención del teclado

# Rutinas de atención del teclado

- Leemos del teclado a través del puerto **0x60** (**in al, 0x60**).
- Obtenemos un **scan code**.

# Rutinas de atención del teclado

- Leemos del teclado a través del puerto **0x60** (**in al, 0x60**).
- Obtenemos un **scan code**.

**Scan code:**

*Es lo que genera el teclado luego de presionar una tecla; es decir, a cada tecla le corresponde uno.*
*El teclado reconoce cuando se está presionando una tecla y cuando se la está soltando y genera diferentes códigos en cada caso. Estos códigos son denominados* **make** *codes y* **break** *codes, respectivamente.*

# Rutinas de atención del teclado

- Leemos del teclado a través del puerto **0x60** (**in al, 0x60**).
- Obtenemos un **scan code**.

## Scan code:

*Es lo que genera el teclado luego de presionar una tecla; es decir, a cada tecla le corresponde uno.*
*El teclado reconoce cuando se está presionando una tecla y cuando se la está soltando y genera diferentes códigos en cada caso. Estos códigos son denominados* **make** *codes y* **break** *codes, respectivamente.*

## Por ejemplo:

La tecla **a** tiene asociado el scan code **0x1E**, la tecla **b** el **0x30**, etc.
Cuando se suelta la tecla **a** se genera el break code **0x9E** ($=$ 0x1E $+$ 0x80), es decir, se suma **0x80** al valor del make code.

# Rutinas de atención del teclado

- Leemos del teclado a través del puerto **0x60** (**in al, 0x60**).
- Obtenemos un **scan code**.

## Scan code:

*Es lo que genera el teclado luego de presionar una tecla; es decir, a cada tecla le corresponde uno.*
*El teclado reconoce cuando se está presionando una tecla y cuando se la está soltando y genera diferentes códigos en cada caso. Estos códigos son denominados* **make** *codes y* **break** *codes, respectivamente.*

## Por ejemplo:

La tecla **a** tiene asociado el scan code **0x1E**, la tecla **b** el **0x30**, etc.
Cuando se suelta la tecla **a** se genera el break code **0x9E** (= 0x1E + 0x80), es decir, se suma **0x80** al valor del make code.

Pueden ver los scan codes correpondientes a cada tecla en:
`http://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html` (sección: "**1.4 Ordinary scancodes**").

# Listo!

¿Preguntas?

# ¡TP!

1. Completar las entradas necesarias en la `IDT` para asociar una rutina a la interrupción del reloj, otra a la interrupción de teclado y por último una a la interrupción de software `0x46`.
2. Escribir la rutina asociada a la interrupción del reloj, para que por cada *tick* llame a la función `game_tick`. La misma se encarga de mostrar cada vez que se llame, la animación de un cursor rotando en la esquina inferior derecha de la pantalla. La función `screen_actualizar_reloj_global` está definida en `screen.h`.
3. Escribir la rutina asociada a la interrupción de teclado de forma que si se presiona cualquiera de las teclas a utilizar en el juego, se presente la misma en la esquina superior derecha de la pantalla.
4. Escribir la rutina asociada a la interrupción `0x46` para que modifique el valor de `eax` por $0x42$. Posteriormente este comportamiento va a ser modificado para atender el servicio del sistema.