Interrupciones en Modo Protegido

Organización del Computador II

Sofía Massobrio

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

15/10/2019

IDT

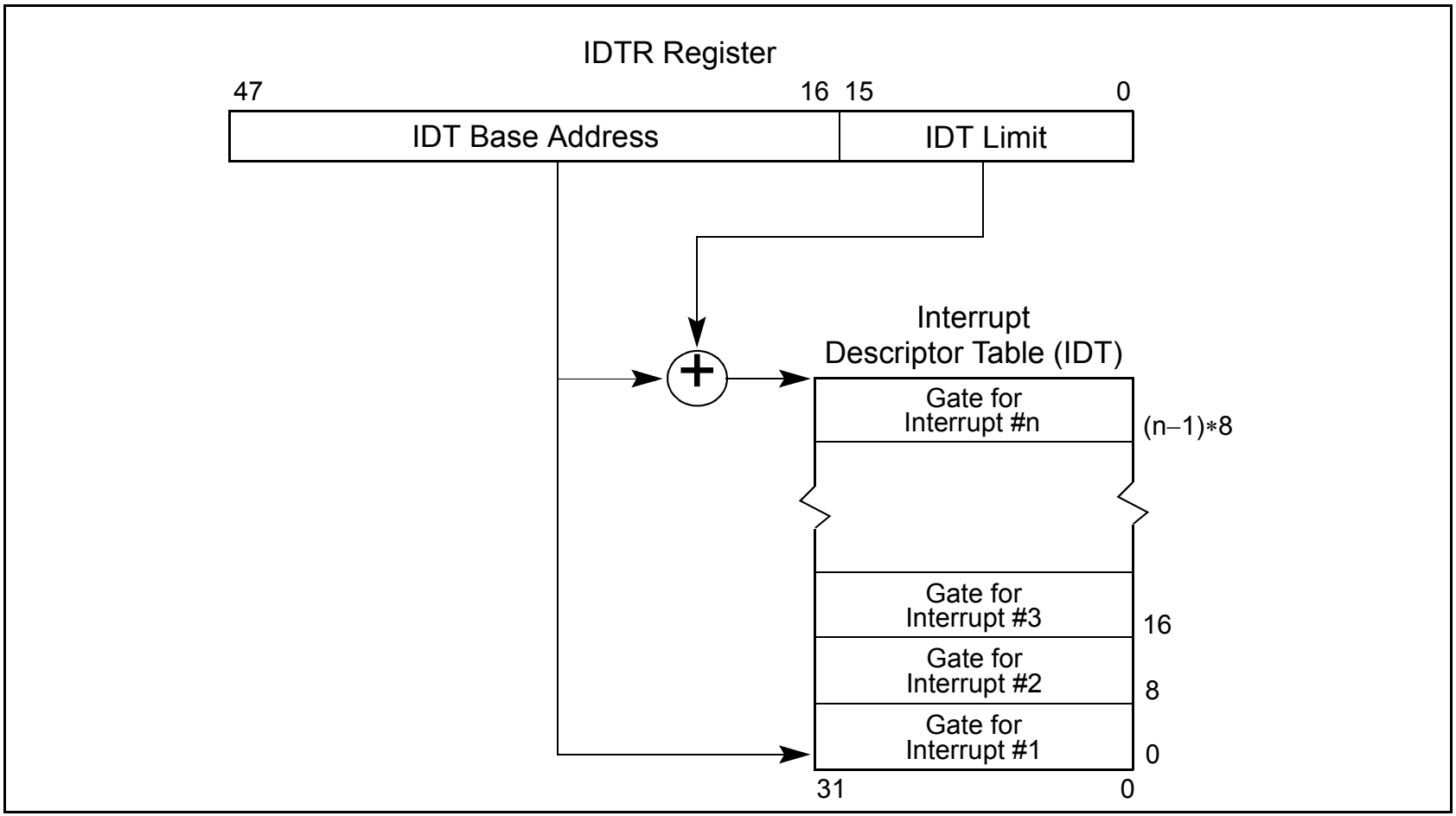
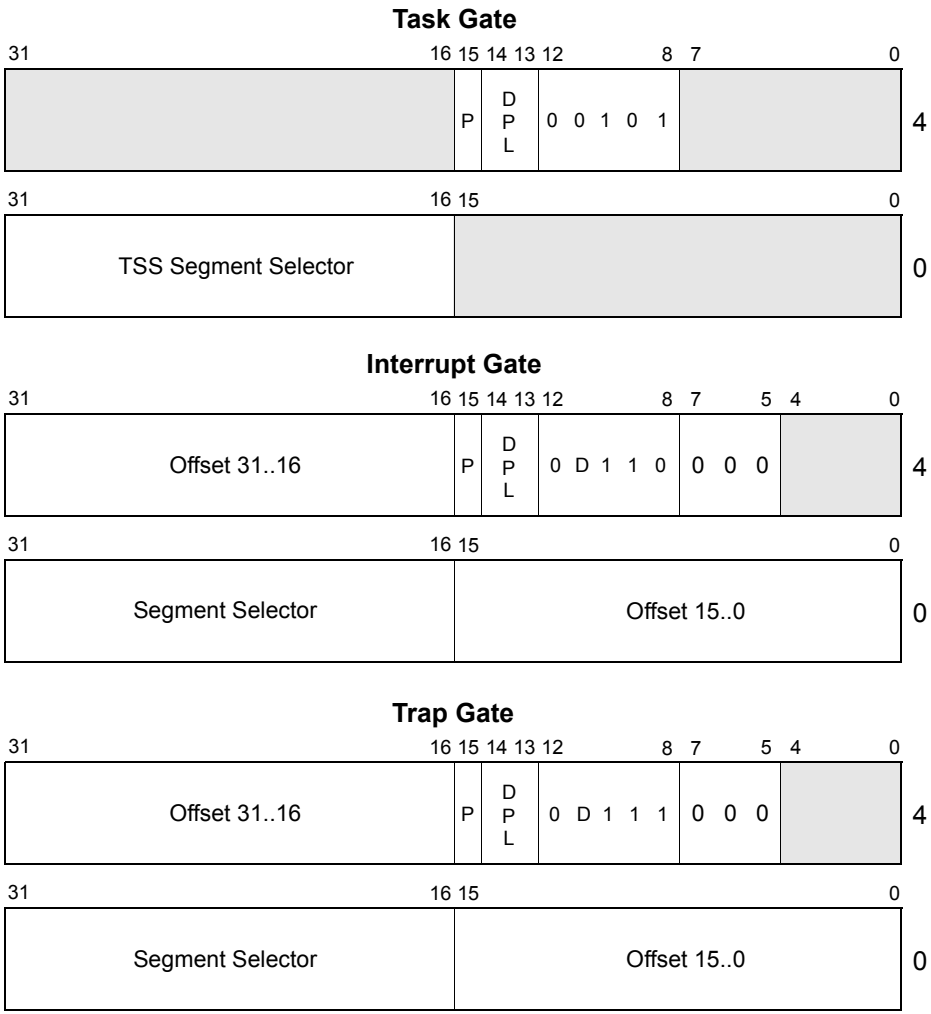


Figure 5-1. Relationship of the IDTR and IDT

Descriptores

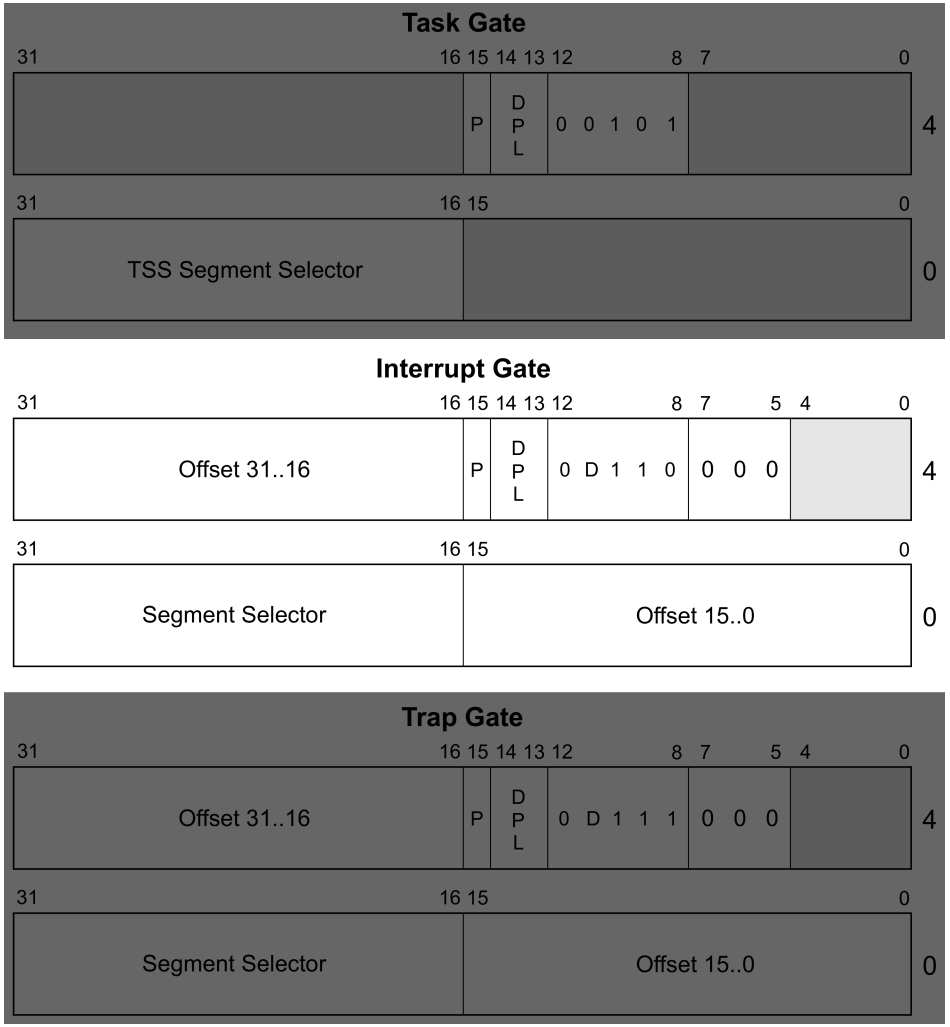


DPL	Descriptor Privilege Level
Offset	Offset to procedure entry point
P	Segment Present flag
Selector	Segment Selector for destination code segment
D	Size of gate: 1 = 32 bits; 0 = 16 bits

Reserved

Figure 5-2. IDT Gate Descriptors

Descriptores



DPL	Descriptor Privilege Level
Offset	Offset to procedure entry point
P	Segment Present flag
Selector	Segment Selector for destination code segment
D	Size of gate: 1 = 32 bits; 0 = 16 bits

Reserved

Figure 5-2. IDT Gate Descriptors

Rutina de Atención de Interrupción

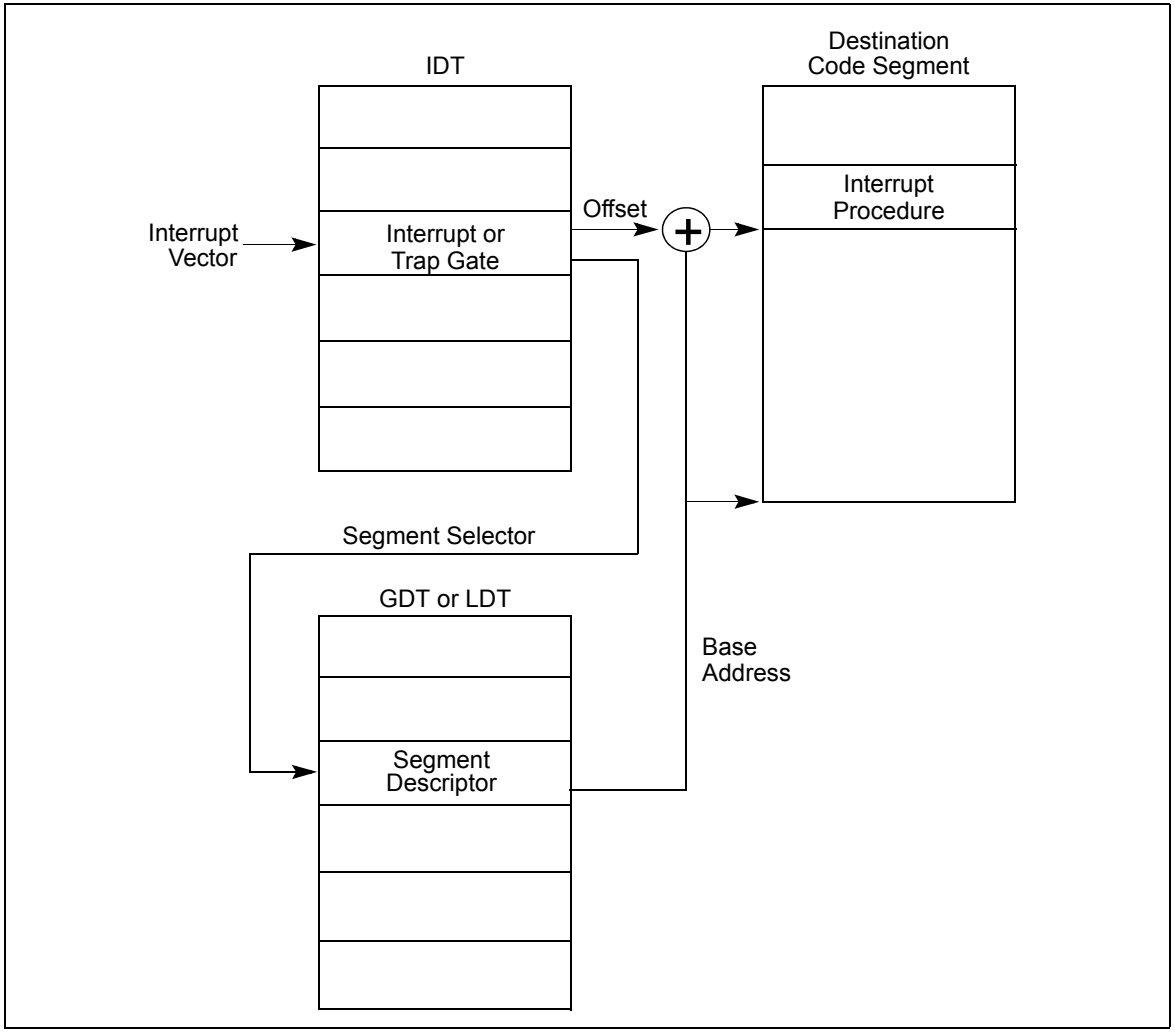


Figure 5-3. Interrupt Procedure Call

Ejercicio 2

a. Completar las entradas necesarias en la `IDT` para asociar diferentes rutinas a todas las excepciones del procesador. Cada rutina de excepción debe indicar en pantalla qué problema se produjo e interrumpir la ejecución. Posteriormente se modificarán estas rutinas para que se continúe la ejecución, resolviendo el problema y desalojando a la tarea que lo produjo.

b. Hacer lo necesario para que el procesador utilice la `IDT` creada anteriormente. Generar una excepción para probarla.

Archivos

1. `idt.h`: Descripción de las estructuras
2. `idt.c`: Estructura de la IDT con cada una de sus entradas

Estructuras

- Struct de descriptor de IDT

```
typedef struct str_idt_descriptor {
unsigned short idt_length;
unsigned int idt_addr;
} __attribute__((__packed__)) idt_descriptor;
```

- Struct de una entrada de la IDT

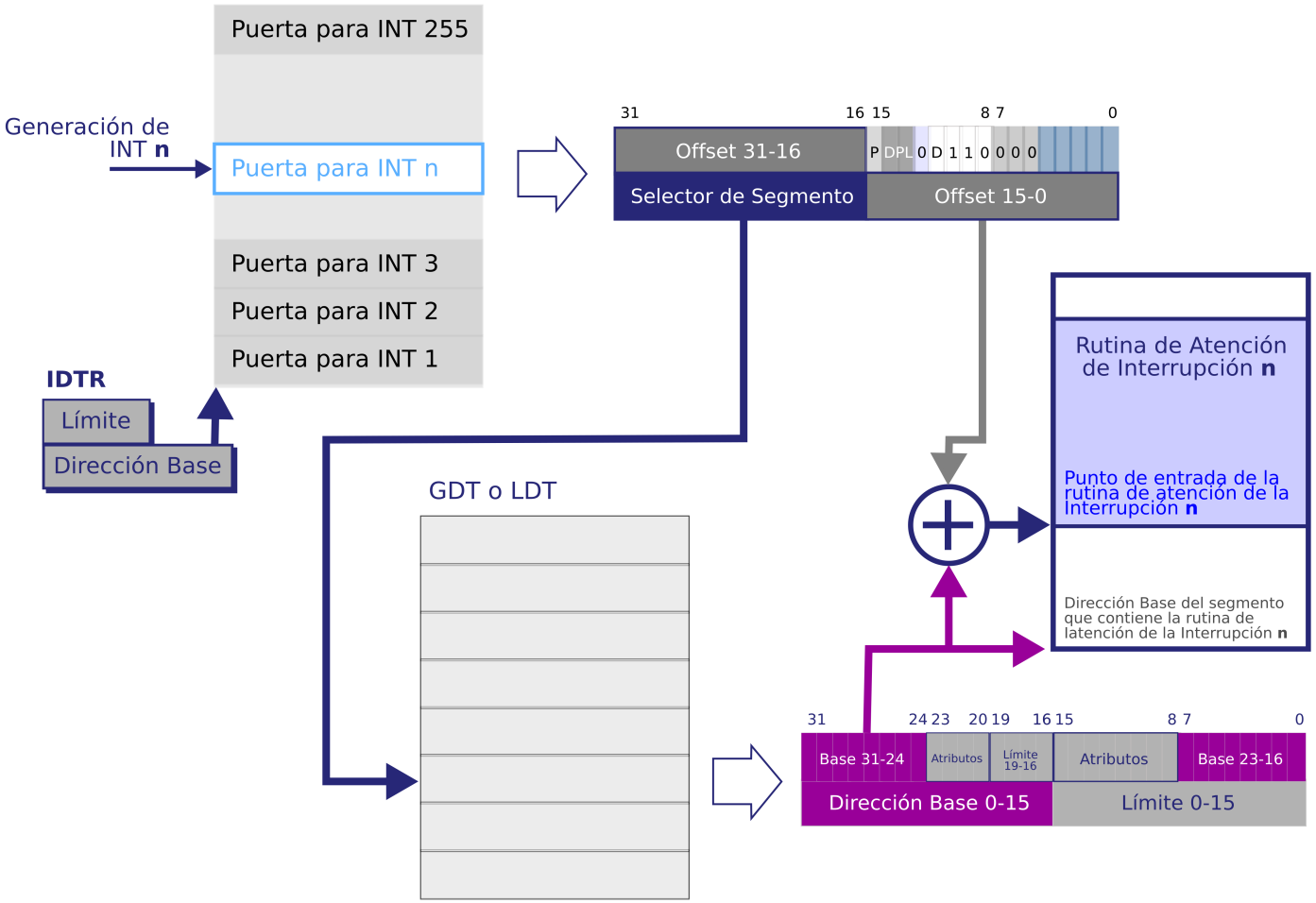
```
typedef struct str_idt_entry_fld {
unsigned short offset_0_15;
unsigned short segsel;
unsigned short attr;
unsigned short offset_16_31;
} __attribute__((__packed__, aligned (8))) idt_entry;
```

Otros

- `idt_entry idt[255] = { };`
- `IDT_ENTRY(numero)`: Permite declarar una entrada en la IDT, para la utilizar el handler de nombre `_isr`*NUMERO*

- `void inicializar_idt()`: Función llamada desde el kernel para inicializar las entradas en la idt

Rutinas de Atención de Interrupción



Rutinas de Atención de Interrupción

Qué debo hacer para manejar correctamente una interrupción

1. Preservar los registros que vayamos a romper
   (*¡la interrupción debe ser transparente!*)
2. Realizar la tarea correspondiente a la interrupción
3. Restaurar los registros
4. Retornar de la interrupción

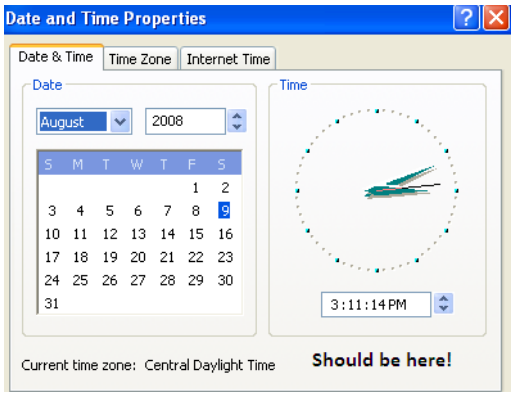
Archivos

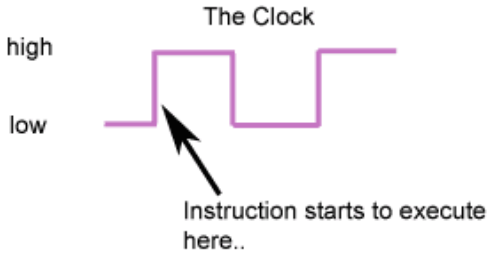
1. `isr.h`: Descripción de las funciones handlers
2. `isr.asm`: Código de los handlers

Macro

- `ISR numero`: Crea la etiqueta con el código de la RAI `_isr`*NUMERO*

¿Qué es un clock?





El clock

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

El teclado

También veremos cómo capturar las interrupciones generadas por el teclado al presionar una tecla.

Cómo acceder al teclado

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

Scan code:

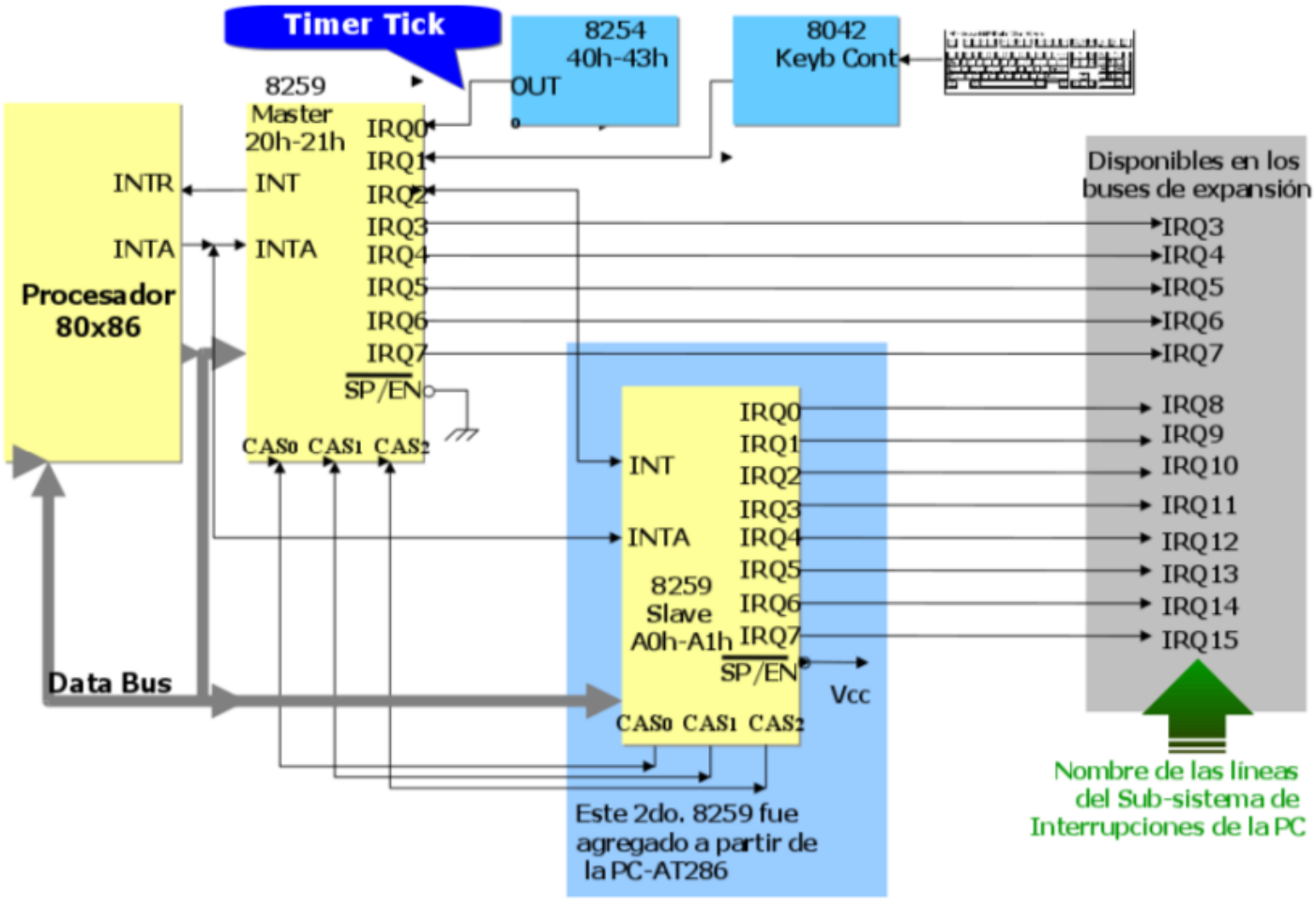
Es lo que genera el teclado luego de presionar una tecla.
El teclado reconoce cuando se está presionando una tecla y cuando se la está soltando. Genera diferentes códigos en cada caso, denominados **make** *codes y* **break** *codes, respectivamente.*
A cada tecla le corresponde un *scan code*.

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.

Scan codes: `http://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html` (sección: "**1.4 Ordinary scancodes**").

¿Cómo las manejamos?



Habilitando/Deshabilitando PIC

¿Cómo lo hacemos?

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

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

Además...

Cuando atendemos una interrupción del PIC debemos notificarle que ya atendimos dicha interrupción. Para eso tenemos la función **pic_finish1**

Ejercicio 3

a. Completar las entradas necesarias en la `IDT` para asociar una rutina a la interrupción del reloj y otra a la interrupción de teclado. Además crear una entrada adicional para la interrupción de software `47`.
b. Escribir la rutina asociada a la interrupción del reloj, para que por cada tick llame a la función `nextClock`. 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 `nextClock` está definida en `isr.asm`.
c. Escribir la rutina asociada a la interrupción de teclado de forma que si se presiona cualquiera de `0` a `9`, se presente la misma en la esquina superior derecha de la pantalla.
d. Escribir la rutina asociada a la interrupción `47` para que modifique el valor de `eax` por *0x42*. Posteriormente este comportamiento va a ser modificado para atender uno de los servicios del sistema.