

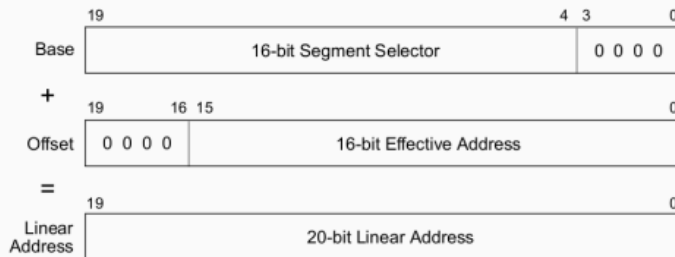
Figure 2-1. IA-32 System-Level Registers and Data Structures

## Modo real

- Trabaja con 16 bits
- 1MB de memoria
- No hay protección ni privilegios en memoria

Las direcciones en modo real (20 bits) se forman con 2 componentes de 16 bits:

- **Dirección Base:** valor de un **registro de segmento** (CS,DS,ES,SS) shifteado 4 bits a la izquierda
- **Offset:** el valor de un **registro** (AX, BX, CX, DX, SP, BP, SI y DI)



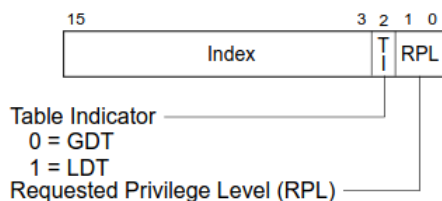
Ejemplo: *Dirección Física* = *Segmento* 16 + Offset\* *Segmento* = 0x12F3. Offset = 0x4B27  
 0x17A57 = 0x12F30 + 0x4B27

### Modo protegido

- 32/64 bits
- 4GB de memoria para 32 bits
- 4 niveles de privilegio, atención a interrupciones con privilegio.

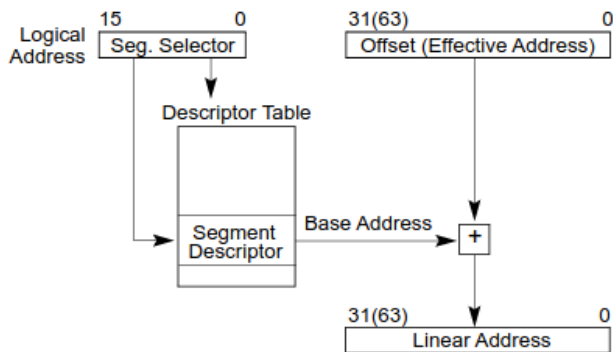
### Segmentación

- **Linear Address space:** memoria direccionable por el procesador. Generalmente se divide en segmentos.
- **Dirección lógica:** Consiste en un **selector de segmento** y un **offset**, y sirve para direccionar a un byte dentro de un segmento.
  - **Segment selector:** Es un identificador único para un segmento en un descriptor table (por ejemplo la GDT, Global Descriptor Table) que dá a lugar a un descriptor de segmento. Consiste en un índice, un bit indicador (0 GDT, 1 LDT) y 2 bits para el privilegio.



**Figure 3-6. Segment Selector**

- **Offset:** Se suma el offset con el base adress para obtener el byte deseado dentro del segmento.

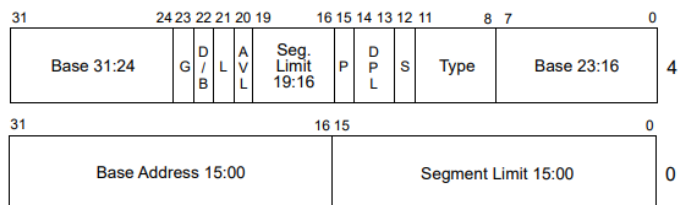


**Figure 3-5. Logical Address to Linear Address Translation**

- Si no se usa paginación, la dirección lineal se mapea directamente con la dirección física (la linear se manda directo al bus). Si se usa paginación se hace una segunda traducción de dirección lineal a dirección física.
- **Segmentación Flat:** El sistema operativo y las aplicaciones tiene acceso a una memoria contigua, en general se usa un segmento para todo el espacio de direccionamiento de 4gb, se define uno para datos de nivel 0 y 3, y código de nivel 0 y 3 que se solapan. Con esto se esconde la segmentación.
- **GDTR (Global Descriptor Table Register):** Consiste en un registro de 48 bits, los 32 mas significativos guarda la dirección de la GDT en el espacio de direccionamiento lineal. Los restantes 16 definen el limite en bytes de la tabla. Entonces máximo la tabla puede tener  $2^{16} / 2^3$  (el tamaño de un **gate** de la gdt) =  $2^{13} = 8192$  descriptores de segmento. El byte limite se incluye en el tamaño, es decir que si el limite = 0, entonces la GDT tiene solo un byte de tamaño, por lo que el límite tiene que ser uno menos un múltiplo de 8 ( $8N - 1$ ) para tener al menos una entry.

Se carga con la instrucción lgdt [mem], que recibe una dirección en la que toma 48 bits

- **GDT:** Es un arreglo de descriptores de segmento global. El primer descriptor de la tabla no se usa (*Null descriptor*). Cuando un registro de segmento (DS, ES, FS, GS) carga el segmento nulo no genera una excepción, pero cuando se intenta acceder se genera un #GP (general-purpose exception).
- **Segment descriptor:** Especifica el tamaño, los permisos y los privilegios, el tipo, la ubicación del primer byte en la linear address space del segmento (**base address**).



L — 64-bit code segment (IA-32e mode only)  
 AVL — Available for use by system software  
 BASE — Segment base address  
 D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)  
 DPL — Descriptor privilege level  
 G — Granularity  
 LIMIT — Segment Limit  
 P — Segment present  
 S — Descriptor type (0 = system; 1 = code or data)  
 TYPE — Segment type

**Figure 3-8. Segment Descriptor**

Table 3-1. Code- and Data-Segment Types

| Type Field |    |          |          |          | Descriptor Type | Description                        |
|------------|----|----------|----------|----------|-----------------|------------------------------------|
| Decimal    | 11 | 10<br>E  | 9<br>W   | 8<br>A   |                 |                                    |
| 0          | 0  | 0        | 0        | 0        | Data            | Read-Only                          |
| 1          | 0  | 0        | 0        | 1        | Data            | Read-Only, accessed                |
| 2          | 0  | 0        | 1        | 0        | Data            | Read/Write                         |
| 3          | 0  | 0        | 1        | 1        | Data            | Read/Write, accessed               |
| 4          | 0  | 1        | 0        | 0        | Data            | Read-Only, expand-down             |
| 5          | 0  | 1        | 0        | 1        | Data            | Read-Only, expand-down, accessed   |
| 6          | 0  | 1        | 1        | 0        | Data            | Read/Write, expand-down            |
| 7          | 0  | 1        | 1        | 1        | Data            | Read/Write, expand-down, accessed  |
|            |    | <b>C</b> | <b>R</b> | <b>A</b> |                 |                                    |
| 8          | 1  | 0        | 0        | 0        | Code            | Execute-Only                       |
| 9          | 1  | 0        | 0        | 1        | Code            | Execute-Only, accessed             |
| 10         | 1  | 0        | 1        | 0        | Code            | Execute/Read                       |
| 11         | 1  | 0        | 1        | 1        | Code            | Execute/Read, accessed             |
| 12         | 1  | 1        | 0        | 0        | Code            | Execute-Only, conforming           |
| 13         | 1  | 1        | 0        | 1        | Code            | Execute-Only, conforming, accessed |
| 14         | 1  | 1        | 1        | 0        | Code            | Execute/Read, conforming           |
| 15         | 1  | 1        | 1        | 1        | Code            | Execute/Read, conforming, accessed |

Table 3-2. System-Segment and Gate-Descriptor Types

| Type Field |    |    |   |   | Description            |                        |
|------------|----|----|---|---|------------------------|------------------------|
| Decimal    | 11 | 10 | 9 | 8 | 32-Bit Mode            | IA-32e Mode            |
| 0          | 0  | 0  | 0 | 0 | Reserved               | Reserved               |
| 1          | 0  | 0  | 0 | 1 | 16-bit TSS (Available) | Reserved               |
| 2          | 0  | 0  | 1 | 0 | LDT                    | LDT                    |
| 3          | 0  | 0  | 1 | 1 | 16-bit TSS (Busy)      | Reserved               |
| 4          | 0  | 1  | 0 | 0 | 16-bit Call Gate       | Reserved               |
| 5          | 0  | 1  | 0 | 1 | Task Gate              | Reserved               |
| 6          | 0  | 1  | 1 | 0 | 16-bit Interrupt Gate  | Reserved               |
| 7          | 0  | 1  | 1 | 1 | 16-bit Trap Gate       | Reserved               |
| 8          | 1  | 0  | 0 | 0 | Reserved               | Reserved               |
| 9          | 1  | 0  | 0 | 1 | 32-bit TSS (Available) | 64-bit TSS (Available) |
| 10         | 1  | 0  | 1 | 0 | Reserved               | Reserved               |
| 11         | 1  | 0  | 1 | 1 | 32-bit TSS (Busy)      | 64-bit TSS (Busy)      |
| 12         | 1  | 1  | 0 | 0 | 32-bit Call Gate       | 64-bit Call Gate       |
| 13         | 1  | 1  | 0 | 1 | Reserved               | Reserved               |
| 14         | 1  | 1  | 1 | 0 | 32-bit Interrupt Gate  | 64-bit Interrupt Gate  |
| 15         | 1  | 1  | 1 | 1 | 32-bit Trap Gate       | 64-bit Trap Gate       |

- **Registros segmento:** guardan las ultimas traducciones hechas de direcciones lineales. Es decir que solo pueden usarse hasta 6 segmentos a la vez. Se modifican con un jmp far, mov y otras instrucciones. CS: codigo, SS: stack, el resto datos.

| Visible Part     |   | Hidden Part |    |
|------------------|---|-------------|----|
| Segment Selector | Base Address, Limit, Access Information |             | CS |
|                  |   |             | SS |
|                  |   |             | DS |
|                  |   |             | ES |
|                  |   |             | FS |
|                  |   |             | GS |

Figure 3-7. Segment Registers

## Interrupciones

- Se define una identidad numérica para cada interrupción (vectorización) y utiliza una tabla de descriptores donde para cada índice, o identidad, se decide:
  - Donde se encuentra la rutina que lo atiende (dirección de memoria)
  - En qué contexto se va a ejecutar (segmento y nivel de privilegio).
  - De qué tipo de interrupción se trata

### Tipos de interrupciones

- Excepciones que van a ser generadas por el procesador cuando se cumpla una condición, por ejemplo si se quiere acceder a una dirección de memoria a través de un selector cuyo segmento tiene el bit P apagado.
  - *Fault*: Excepción que podría corregirse para que el programa continúe su ejecución. El procesador guarda en la pila la dirección de la instrucción que produjo la falla. Algunos faults suman un código de error a la pila.
  - *Traps*: Excepción producida al terminar la ejecución de una instrucción de trap. El procesador guarda en la pila la dirección de la instrucción a ejecutarse luego de la que causó el trap.
  - *Aborts*: Excepción que no siempre puede determinar la instrucción que la causa, ni permite recuperar la ejecución de la tarea que la causó. Reporta errores severos de hardware o inconsistencias en tablas del sistema.
- Interrupciones:
  - *Externas*: de un dispositivo externo (reloj o teclado)
  - *Internas*: generado por una llamada a la instrucción INT por parte de un proceso.

**Table 6-1. Protected-Mode Exceptions and Interrupts**

| Vector | Mnemonic | Description                                | Type        | Error Code | Source  |
|--------|----------|--|-------------|------------|---|
| 0      | #DE      | Divide Error                               | Fault       | No         | DIV and IDIV instructions.  |
| 1      | #DB      | Debug Exception                            | Fault/ Trap | No         | Instruction, data, and I/O breakpoints; single-step; and others.    |
| 2      | —        | NMI Interrupt                              | Interrupt   | No         | Nonmaskable external interrupt.                                     |
| 3      | #BP      | Breakpoint                                 | Trap        | No         | INT3 instruction.   |
| 4      | #OF      | Overflow                                   | Trap        | No         | INTO instruction.   |
| 5      | #BR      | BOUND Range Exceeded                       | Fault       | No         | BOUND instruction.  |
| 6      | #UD      | Invalid Opcode (Undefined Opcode)          | Fault       | No         | UD instruction or reserved opcode.                                  |
| 7      | #NM      | Device Not Available (No Math Coprocessor) | Fault       | No         | Floating-point or WAIT/FWAIT instruction.                           |
| 8      | #DF      | Double Fault                               | Abort       | Yes (zero) | Any instruction that can generate an exception, an NMI, or an INTR. |
| 9      |          | Coprocessor Segment Overrun (reserved)     | Fault       | No         | Floating-point instruction. <sup>1</sup>                            |
| 10     | #TS      | Invalid TSS                                | Fault       | Yes        | Task switch or TSS access.  |
| 11     | #NP      | Segment Not Present                        | Fault       | Yes        | Loading segment registers or accessing system segments.             |
| 12     | #SS      | Stack-Segment Fault                        | Fault       | Yes        | Stack operations and SS register loads.                             |
| 13     | #GP      | General Protection                         | Fault       | Yes        | Any memory reference and other protection checks.                   |
| 14     | #PF      | Page Fault                                 | Fault       | Yes        | Any memory reference.   |

**Table 6-1. Protected-Mode Exceptions and Interrupts (Contd.)**

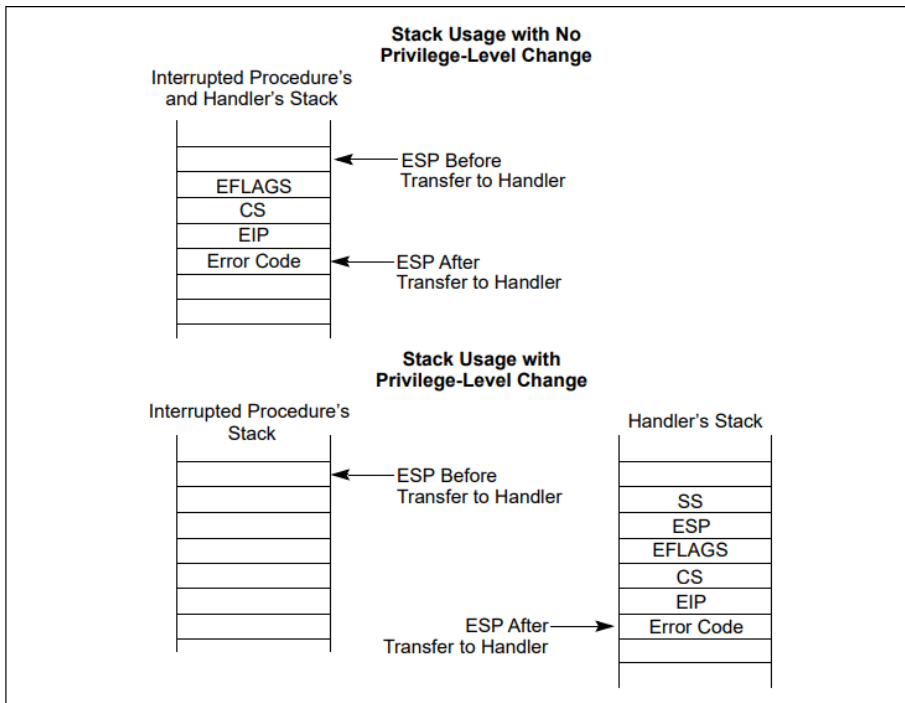
| Vector | Mnemonic | Description                               | Type      | Error Code | Source  |
|--------|----------|---|-----------|------------|---|
| 15     | —        | (Intel reserved. Do not use.)             |           | No         |   |
| 16     | #MF      | x87 FPU Floating-Point Error (Math Fault) | Fault     | No         | x87 FPU floating-point or WAIT/FWAIT instruction.   |
| 17     | #AC      | Alignment Check                           | Fault     | Yes (Zero) | Any data reference in memory. <sup>2</sup>  |
| 18     | #MC      | Machine Check                             | Abort     | No         | Error codes (if any) and source are model dependent. <sup>3</sup>   |
| 19     | #XM      | SIMD Floating-Point Exception             | Fault     | No         | SSE/SSE2/SSE3 floating-point instructions <sup>4</sup>  |
| 20     | #VE      | Virtualization Exception                  | Fault     | No         | EPT violations <sup>5</sup>   |
| 21     | #CP      | Control Protection Exception              | Fault     | Yes        | RET, IRET, RSTORSSP, and SETSSBSY instructions can generate this exception. When CET indirect branch tracking is enabled, this exception can be generated due to a missing ENDBRANCH instruction at target of an indirect call or jump. |
| 22-31  | —        | Intel reserved. Do not use.               |           |            |   |
| 32-255 | —        | User Defined (Non-reserved) Interrupts    | Interrupt |            | External interrupt or INT <i>n</i> instruction.   |

NOTES:

### Atención a interrupciones

Antes de la atención, el procesador pushea a la pila EFLAGS, CS, EIP y el código de error si hay según la interrupción. Luego el handler hace IRET que saca de la pila los primeros 3 mencionados (el error code depende del programador poner en la posición correcta el esp, ya que algunas interrupciones no tienen error code).

Según si el handler de la interrupción tiene un privilegio mayor que la tarea ejecutándose al momento de producirse dicha interrupción, se puede generar un “cambio” de stack. Éste stack con mayor privilegio está definido en la TSS de dicha tarea.



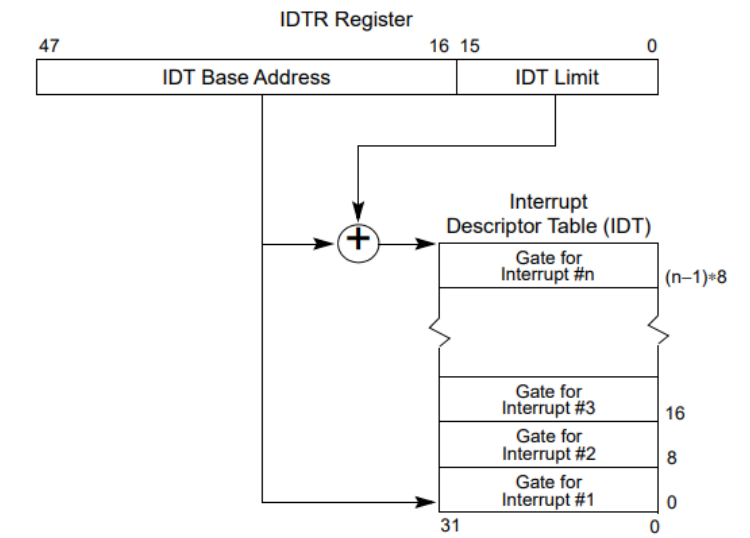
**Figure 6-4. Stack Usage on Transfers to Interrupt and Exception-Handling Routines**

- La pila se alinea a 4 bytes (no en 16 como se hacía para la convención con C).

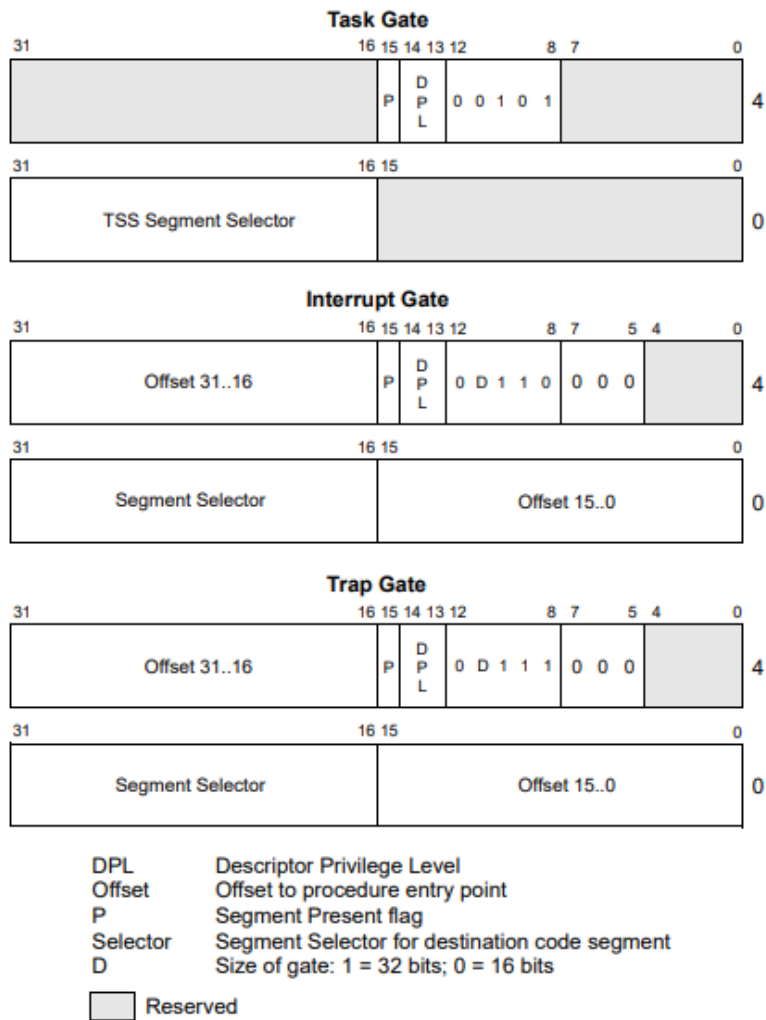
## Estructura de la IDT

Parecida a la GDT, guarda por cada interrupción posible (son 255. Pueden haber menos, se dejan el bit p en 0) se define en que segmento y offset (dentro del segmento) están los handlers, además de algunos atributos.

Con lidt [mem] se carga el idtr. Consiste en 48 bits, el del base address indica la dirección de la IDT y el límite el tamaño en bytes.



**Figure 6-1. Relationship of the IDTR and IDT**



**Figure 6-2. IDT Gate Descriptors**

los bits 12 a 8 indican el type, ver tabla de gdt para types de sistema. El que usamos para interrupciones de 32 bits es el 1110

Se pueden implementar syscalls con interrupciones. Las syscalls es el mecanismo que se tiene para dar funcionamiento de nivel de privilegio alto para las aplicaciones de usuario de forma segura.

## PIC

Controlador que maneja las interrupciones externas. En un principio hay que configurarlo con un protocolo definido. Como los codigos de interrupciones se pisan con los del sistema, se debe hacer un remapeo.





```
}  
  
void pic_disable() {  
    outb(PIC1_PORT + 1, 0xFF);  
    outb(PIC2_PORT + 1, 0xFF);  
}
```

## Paginación

## Tareas