Práctica de Organización del Computador II

Paginación

Primer Cuatrimestre 2025

Organización del Computador II
DC - UBA

Introducción

Sobre la clase de hoy

En la clase de hoy vamos a ver:

Sobre la clase de hoy

En la clase de hoy vamos a ver:

- **Repaso sobre direcciones y memoria**

Sobre la clase de hoy

En la clase de hoy vamos a ver:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**

Sobre la clase de hoy

En la clase de hoy vamos a ver:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**
- **Ideas básicas de los mecanismos de paginación**

Sobre la clase de hoy

En la clase de hoy vamos a ver:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**
- **Ideas básicas de los mecanismos de paginación**
- **Proceso de traducción y estructuras involucradas (`CR3`, `page directory`, `page table`)**

Sobre la clase de hoy

En la clase de hoy vamos a ver:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**
- **Ideas básicas de los mecanismos de paginación**
- **Proceso de traducción y estructuras involucradas (`CR3`, `page directory`, `page table`)**

La idea es presentar la información necesaria para ayudarles a completar el taller.

Direcciones y memoria

¿Qué es una dirección (de memoria)?

Una dirección es **la información necesaria para, dado un contexto, identificar un recurso**.

En esta materia vamos a usar *dirección* para *dirección de memoria*.

¿Qué es una dirección (de memoria)?

Una dirección es **la información necesaria para, dado un contexto, identificar un recurso**.

En esta materia vamos a usar *dirección* para *dirección de memoria*.

Busquemos una definición específica entonces:

- ¿Cuál es el recurso?
- ¿Cuál es el contexto de una dirección física?
- ¿Y en una virtual?

¿Qué es una dirección (de memoria)?

Una dirección es **la información necesaria para, dado un contexto, identificar un recurso**.

En esta materia vamos a usar *dirección* para *dirección de memoria*.

Busquemos una definición específica entonces:

- ¿Cuál es el recurso? Una *porción* de la memoria
- ¿Cuál es el contexto de una dirección física?
- ¿Y en una virtual?

¿Qué es una dirección (de memoria)?

Una dirección es **la información necesaria para, dado un contexto, identificar un recurso**.

En esta materia vamos a usar *dirección* para *dirección de memoria*.

Busquemos una definición específica entonces:

- ¿Cuál es el recurso? Una *porción* de la memoria
- ¿Cuál es el contexto de una dirección física? La *computadora*
- ¿Y en una virtual?

¿Qué es una dirección (de memoria)?

Una dirección es **la información necesaria para, dado un contexto, identificar un recurso**.

En esta materia vamos a usar *dirección* para *dirección de memoria*.

Busquemos una definición específica entonces:

- ¿Cuál es el recurso? Una *porción* de la memoria
- ¿Cuál es el contexto de una dirección física? La *computadora*
- ¿Y en una virtual? *computadora* + *esquema de paginación*

Direcciones y recursos

Hay cuatro casos posibles que pueden ocurrir al pensar de direcciones y recursos de esta manera:

- **Misma dirección, distinto recurso:**
  Mi casa significa distintas cosas para distintas personas
- **Distinta dirección, distinto recurso:**
  Este es el caso "normal"
- **Misma dirección, mismo recurso:**
  Ocurre con **recursos compartidos** *mi facultad* significa lo mismo para todos acá
- **Distinta dirección, mismo recurso:**
  Lo que yo le digo *"mi casa"* otros le dicen *"la casa del profe"*

Casos de uso

Casos de uso

Antes de entrar en detalle veamos un par de ejemplos de cosas que se pueden hacer con paginación.

Casos de uso

Antes de entrar en detalle veamos un par de ejemplos de cosas que se pueden hacer con paginación.

Varios de estos ejemplos se implementan manejando los *page faults* de forma especial. Es un buen momento de recordar que recibir una *excepción* del procesador no siempre significa *explotó todo*.

Carga bajo demanda

Para comenzar a ejecutar un programa no es necesario cargar **todo** su código. Si se marcan las páginas no cargadas como ausentes el procesador emitirá un *page fault* cuando se las intente usar.

Ni bien ocurra un *page fault* el sistema puede cargar la página asociada, marcarla como presente y seguir la ejecución.

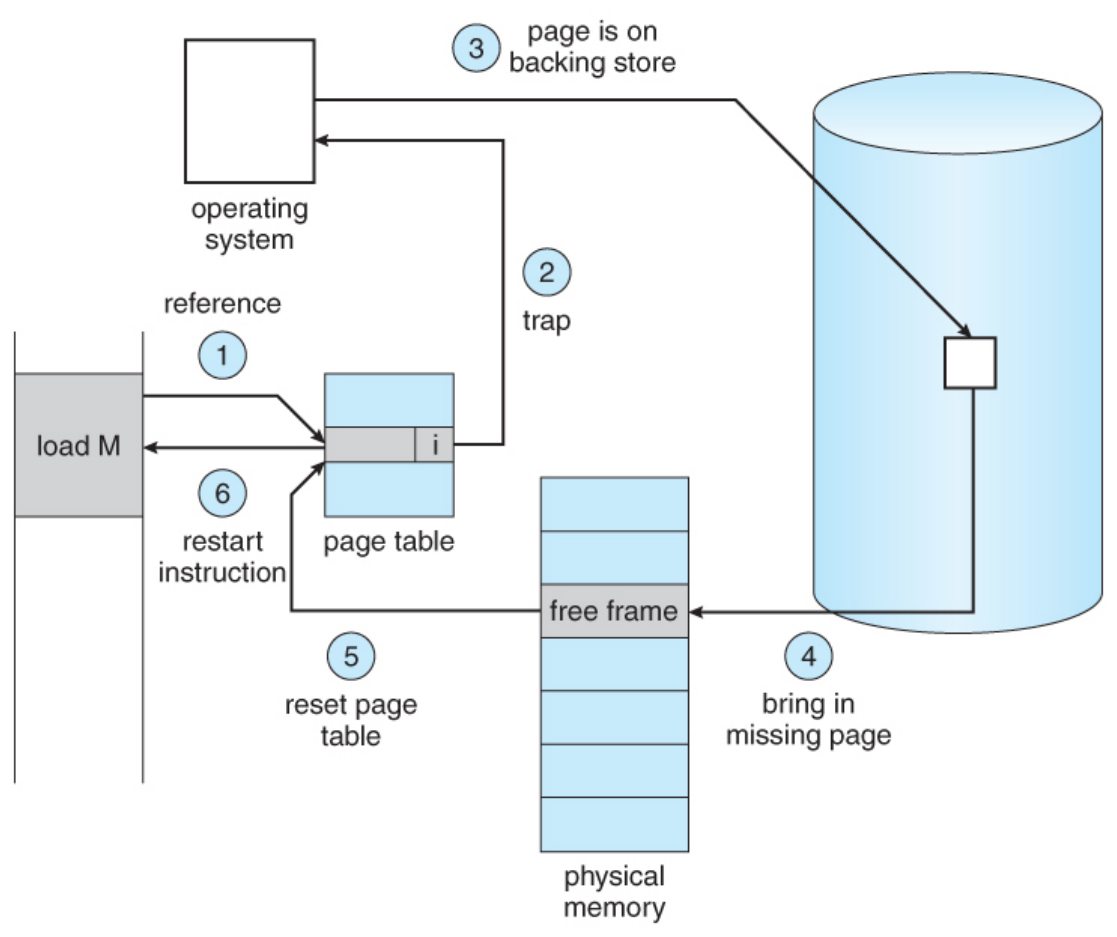
Carga bajo demanda

Para comenzar a ejecutar un programa no es necesario cargar **todo** su código. Si se marcan las páginas no cargadas como ausentes el procesador emitirá un *page fault* cuando se las intente usar.

Ni bien ocurra un *page fault* el sistema puede cargar la página asociada, marcarla como presente y seguir la ejecución.

En el sistema resultante un programa sólo ocupa la memoria que efectivamente usa.

Carga bajo demanda



Un **page fault** no siempre es malo.

Swapping

La memoria no es infinita. Si ésta se encuentra al límite el sistema operativo puede decidir mover páginas (o procesos) poco importantes al disco.

Ni bien esas páginas se necesiten nuevamente se cargan en una dirección física nueva pero preservando las direcciones virtuales originales.

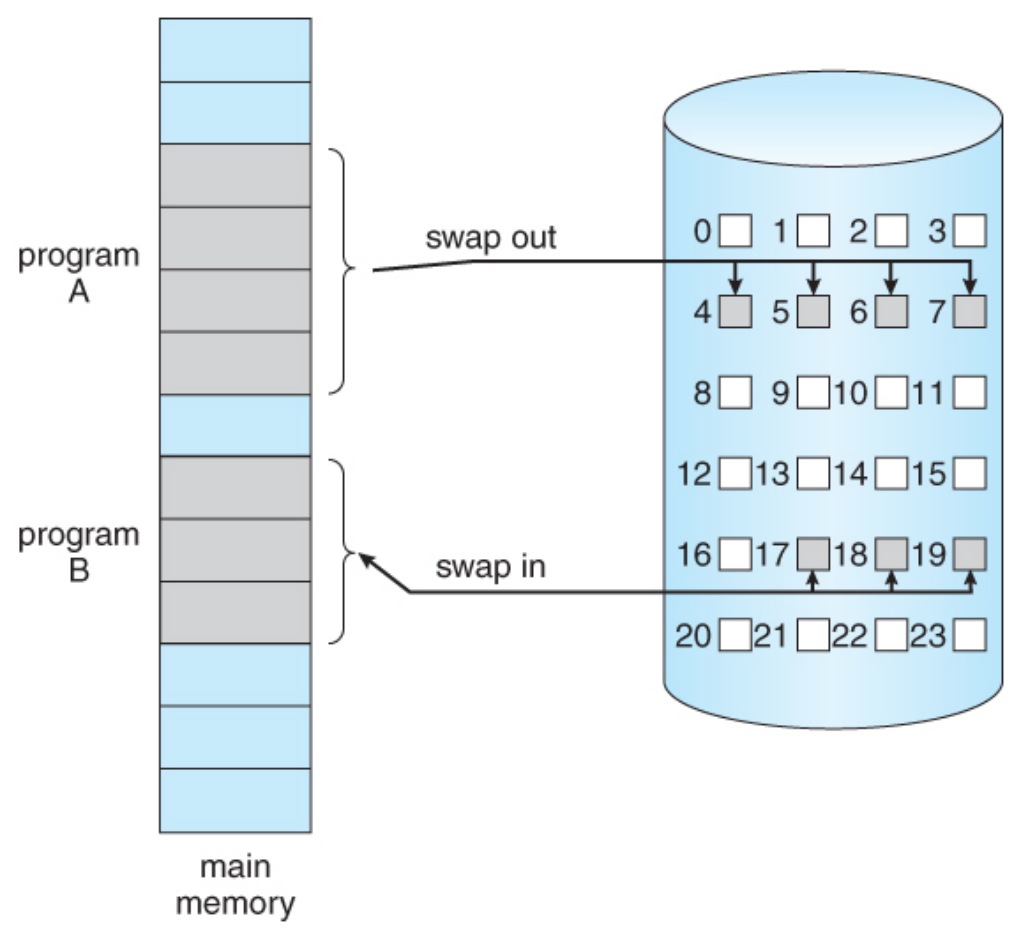
Swapping

La memoria no es infinita. Si ésta se encuentra al límite el sistema operativo puede decidir mover páginas (o procesos) poco importantes al disco.

Ni bien esas páginas se necesiten nuevamente se cargan en una dirección física nueva pero preservando las direcciones virtuales originales.

La cantidad de memoria efectiva del sistema resultante es tan grande como el almacenamiento secundario.

Swapping



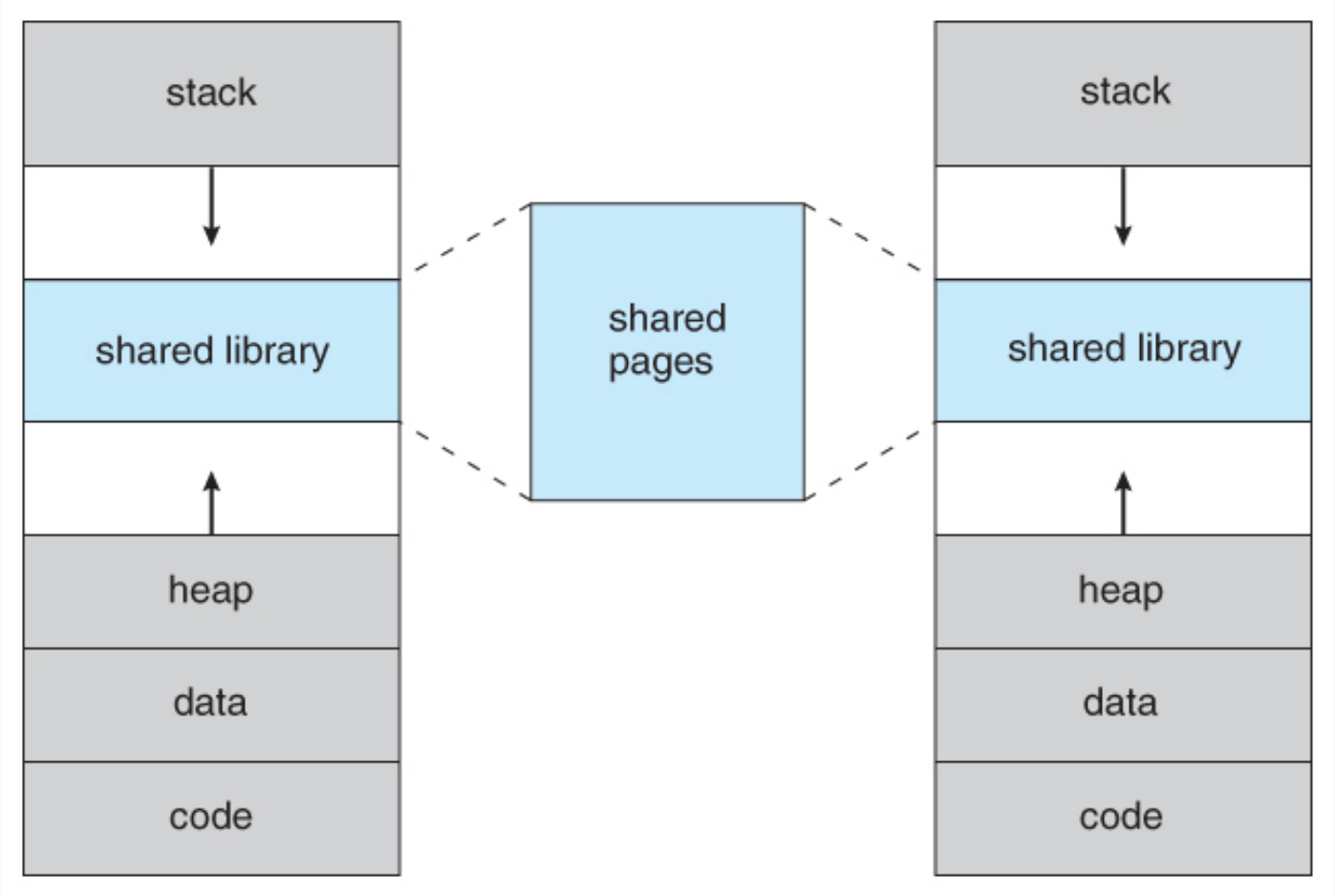
Intercambio de procesos en memoria principal.

Bibliotecas compartidas

Hablamos anteriormente de **bibliotecas compartidas** (`.dll` en Windows, `.so` en Linux y `.dylib` en MacOS). Con ellas podemos compartir código entre varios programas.

Si múltiples programas quieren usar la misma biblioteca no es necesario tener múltiples copias del código disponibles. El código de las bibliotecas en común puede ser *compartido* entre varios programas.

Bibliotecas compartidas



Un mismo rango físico mapeado en distintos rangos virtuales.

Copy on write

Hay un mecanismo de creación de procesos *clásico* llamado (`fork(3)`). Éste *bifurca* un proceso creando dos copias de él.

Una forma ingenua de implementar `fork(3)` es literalmente hacer una copia de todos los recursos en el momento.

La mayoría de los procesos no van a reescribir toda su memoria, evitar duplicar el consumo de memoria ahorraría muchos recursos.

Copy on write

Hay un mecanismo de creación de procesos *clásico* llamado (`fork(3)`). Éste *bifurca* un proceso creando dos copias de él.

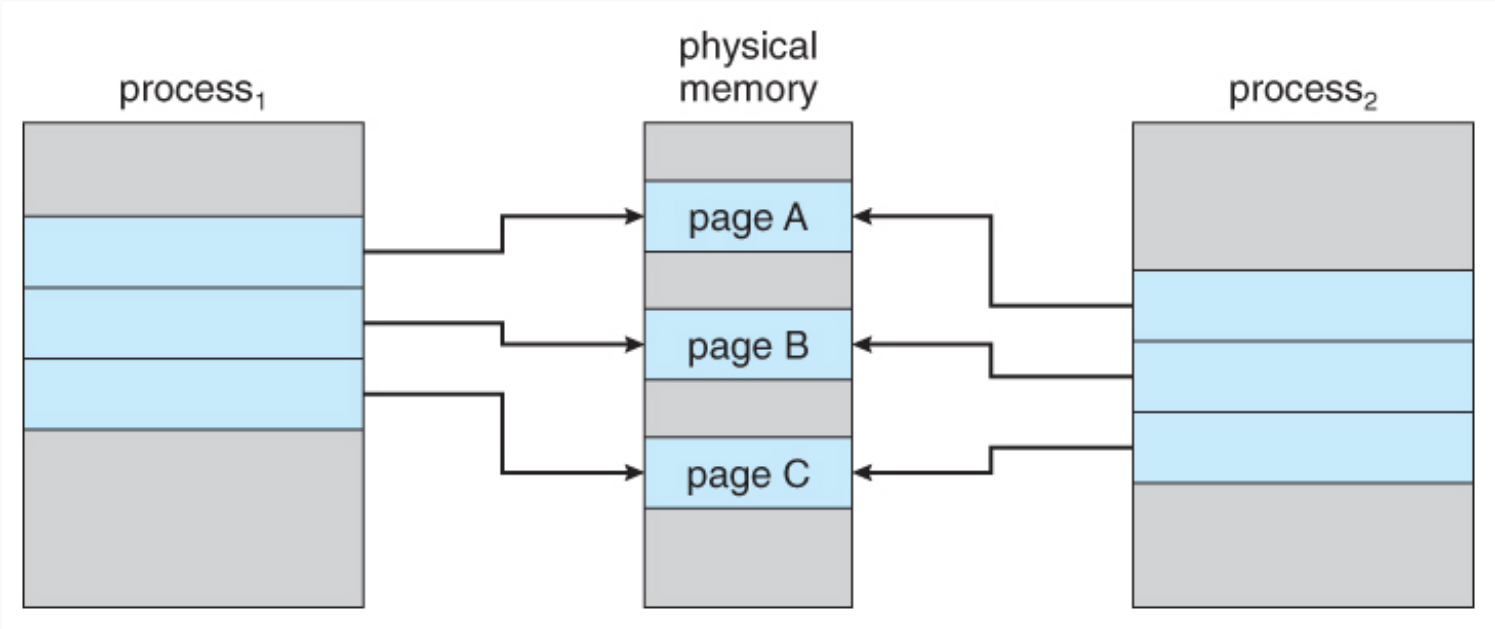
Una forma ingenua de implementar `fork(3)` es literalmente hacer una copia de todos los recursos en el momento.

La mayoría de los procesos no van a reescribir toda su memoria, evitar duplicar el consumo de memoria ahorraría muchos recursos.

Como solución podemos hacer que toda la memoria sea de sólo lectura. Cuando uno de los procesos quiera escribir nos va a llegar un `#PF`. Allí hacemos la copia de la página en cuestión

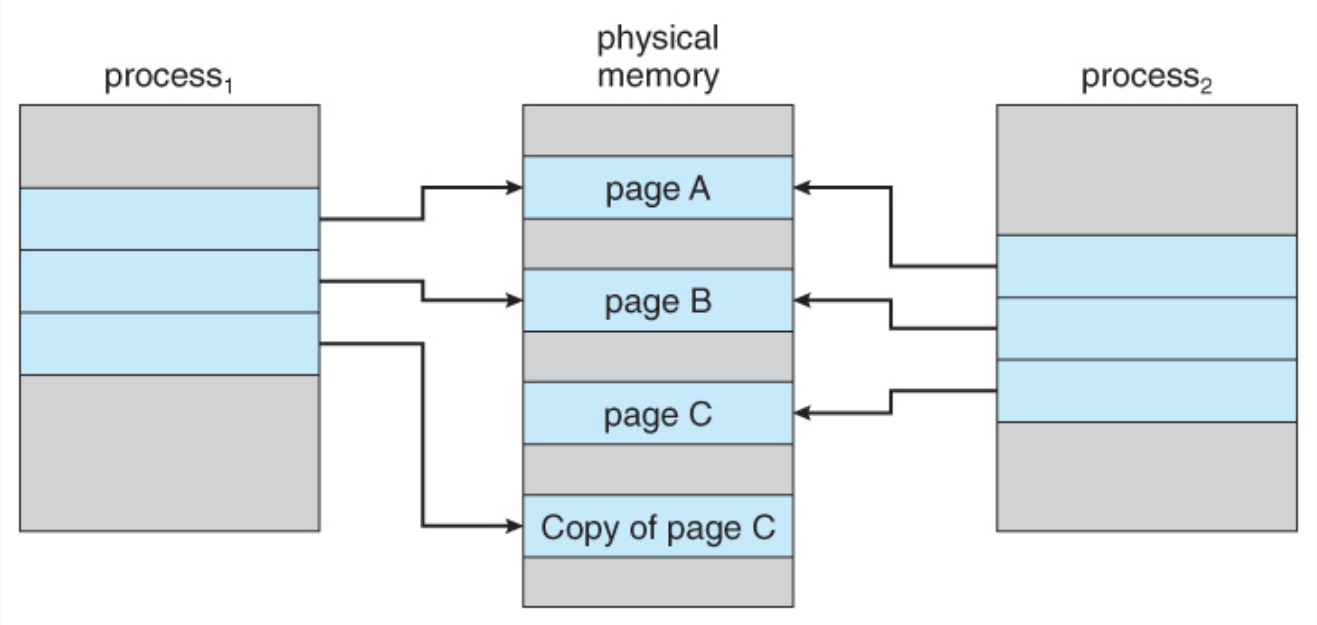
Toda la memoria que se puede compartir termina compartida.

Copy on write



Varios procesos resultantes de un **fork** pueden compartir páginas físicas siempre y cuando no escriban en ellas.

Copy on write



Cuando un proceso escribe en una página compartida es necesario copiarla y modificar las estructuras de paginación acorde a esto.

Mecanismos de paginación

Paginación en 32 bits

Nos interesa tener una noción de *dirección de memoria* que podamos manejar *por proceso*, evaluemos un par de opciones.

Paginación en 32 bits (de a byte)

¡Armemos una tablita!

Paginación en 32 bits (de a byte)

¡Armemos una tablita!

Por cada dirección virtual guardamos la dirección física que le corresponde. Son:

Paginación en 32 bits (de a byte)

¡Armemos una tablita!

Por cada dirección virtual guardamos la dirección física que le corresponde. Son:

- 2^{32} direcciones virtuales

Paginación en 32 bits (de a byte)

¡Armemos una tablita!

Por cada dirección virtual guardamos la dirección física que le corresponde. Son:

- 2^{32} direcciones virtuales
- 32 bits (4 bytes) que guardamos por c/u

Paginación en 32 bits (de a byte)

¡Armemos una tablita!

Por cada dirección virtual guardamos la dirección física que le corresponde. Son:

- 2^{32} direcciones virtuales
- 32 bits (4 bytes) que guardamos por c/u
- La tablita ocupa 4×2^{32} bytes (16GB)

Paginación en 32 bits (de a byte)

¡Armemos una tablita!

Por cada dirección virtual guardamos la dirección física que le corresponde. Son:

- 2^{32} direcciones virtuales
- 32 bits (4 bytes) que guardamos por c/u
- La tablita ocupa 4×2^{32} bytes (16GB)
- **Demasiado grande :(**

Paginación en 32 bits (de a página)

El problema es que estamos traduciendo de a byte, tomemos bloques un poco más grandes. Me dijeron que 4KB (4096 bytes, 2^{12} bytes) es un lindo número.

Paginación en 32 bits (de a página)

El problema es que estamos traduciendo de a byte, tomemos bloques un poco más grandes. Me dijeron que 4KB (4096 bytes, 2^{12} bytes) es un lindo número.

Por cada bloque de direcciones virtuales (página virtual) guardamos el bloque de direcciones físicas (página física) que le corresponde. Son:

Paginación en 32 bits (de a página)

El problema es que estamos traduciendo de a byte, tomemos bloques un poco más grandes. Me dijeron que 4KB (4096 bytes, 2^{12} bytes) es un lindo número.

Por cada bloque de direcciones virtuales (página virtual) guardamos el bloque de direcciones físicas (página física) que le corresponde. Son:

- 2^{32} direcciones virtuales

Paginación en 32 bits (de a página)

El problema es que estamos traduciendo de a byte, tomemos bloques un poco más grandes. Me dijeron que 4KB (4096 bytes, 2^{12} bytes) es un lindo número.

Por cada bloque de direcciones virtuales (página virtual) guardamos el bloque de direcciones físicas (página física) que le corresponde. Son:

- 2^{32} direcciones virtuales
- 2^{12} direcciones por página, osea 2^{20} páginas
- 20 bits (2,5 bytes) que guardamos por cada página

Paginación en 32 bits (de a página)

El problema es que estamos traduciendo de a byte, tomemos bloques un poco más grandes. Me dijeron que 4KB (4096 bytes, 2^{12} bytes) es un lindo número.

Por cada bloque de direcciones virtuales (página virtual) guardamos el bloque de direcciones físicas (página física) que le corresponde. Son:

- 2^{32} direcciones virtuales
- 2^{12} direcciones por página, osea 2^{20} páginas
- 20 bits (2,5 bytes) que guardamos por cada página
- La tablita ocupa $2,5 \times 2^{20}$ bytes (5MB)

Paginación en 32 bits (de a página)

El problema es que estamos traduciendo de a byte, tomemos bloques un poco más grandes. Me dijeron que 4KB (4096 bytes, 2^{12} bytes) es un lindo número.

Por cada bloque de direcciones virtuales (página virtual) guardamos el bloque de direcciones físicas (página física) que le corresponde. Son:

- 2^{32} direcciones virtuales
- 2^{12} direcciones por página, osea 2^{20} páginas
- 20 bits (2,5 bytes) que guardamos por cada página
- La tablita ocupa $2,5 \times 2^{20}$ bytes (5MB)
- **Bastante mejor, pero sigue siendo una tabla un poco grande**

Paginación en 32 bits (de a página)

Forzarnos a escribir todas las traducciones suena razonable hoy (5MB por proceso) pero miremos con más detenimiento:

Paginación en 32 bits (de a página)

Forzarnos a escribir todas las traducciones suena razonable hoy (5MB por proceso) pero miremos con más detenimiento:

- Intel agregó paginación en 1985 con el i386

Paginación en 32 bits (de a página)

Forzarnos a escribir todas las traducciones suena razonable hoy (5MB por proceso) pero miremos con más detenimiento:

- Intel agregó paginación en 1985 con el i386
- 5MB es un costo prohibitivo en esa época (para ser sinceros, también lo sería hoy)

Paginación en 32 bits (de a página)

Forzarnos a escribir todas las traducciones suena razonable hoy (5MB por proceso) pero miremos con más detenimiento:

- Intel agregó paginación en 1985 con el i386
- 5MB es un costo prohibitivo en esa época (para ser sinceros, también lo sería hoy)
- La mayoría de los procesos usan poca memoria, muchas traducciones son innecesarias

Paginación en 32 bits (de a página)

Forzarnos a escribir todas las traducciones suena razonable hoy (5MB por proceso) pero miremos con más detenimiento:

- Intel agregó paginación en 1985 con el i386
- 5MB es un costo prohibitivo en esa época (para ser sinceros, también lo sería hoy)
- La mayoría de los procesos usan poca memoria, muchas traducciones son innecesarias
- Estaría bueno poder marcar memoria como Read-Only/Read-Write ó System/User
  Solo traducir se queda corto

Paginación en 32 bits (de a página)

Forzarnos a escribir todas las traducciones suena razonable hoy (5MB por proceso) pero miremos con más detenimiento:

- Intel agregó paginación en 1985 con el i386
- 5MB es un costo prohibitivo en esa época (para ser sinceros, también lo sería hoy)
- La mayoría de los procesos usan poca memoria, muchas traducciones son innecesarias
- Estaría bueno poder marcar memoria como Read-Only/Read-Write ó System/User
  Solo traducir se queda corto

Queremos poder marcar rangos grandes de memoria como *no traducidos* y traducir en más detalle otros.

Paginación en 32 bits (de a página, dos niveles)

El problema es que queremos traducir todas las páginas: hagamos una traducción *gruesa* la cual mejoramos con otra traducción *fina*:

Paginación en 32 bits (de a página, dos niveles)

El problema es que queremos traducir todas las páginas: hagamos una traducción *gruesa* la cual mejoramos con otra traducción *fina*:

- Tenemos 4GB de memoria direccionable

Paginación en 32 bits (de a página, dos niveles)

El problema es que queremos traducir todas las páginas: hagamos una traducción *gruesa* la cual mejoramos con otra traducción *fina*:

- Tenemos 4GB de memoria direccionable
- Eso son 1024 bloques de 4MB

Paginación en 32 bits (de a página, dos niveles)

El problema es que queremos traducir todas las páginas: hagamos una traducción *gruesa* la cual mejoramos con otra traducción *fina*:

- Tenemos 4GB de memoria direccionable
- Eso son 1024 bloques de 4MB
- Cada bloque son 1024 bloques de 4KB

Paginación en 32 bits (de a página, dos niveles)

El problema es que queremos traducir todas las páginas: hagamos una traducción *gruesa* la cual mejoramos con otra traducción *fina*:

- Tenemos 4GB de memoria direccionable
- Eso son 1024 bloques de 4MB
- Cada bloque son 1024 bloques de 4KB

Suena bastante redondo ¿No?

Paginación en 32 bits (de a página, dos niveles)

Tomemos una tabla de 1024 entradas, cada una traduce los 4MB *que le tocan*:

Paginación en 32 bits (de a página, dos niveles)

Tomemos una tabla de 1024 entradas, cada una traduce los 4MB *que le tocan*:

- Si hacemos entradas de 4 bytes este *directorio de páginas* ocuparía 4KB (¡Una página!)

Paginación en 32 bits (de a página, dos niveles)

Tomemos una tabla de 1024 entradas, cada una traduce los 4MB *que le tocan*:

- Si hacemos entradas de 4 bytes este *directorio de páginas* ocuparía 4KB (¡Una página!)
- Si una entrada dice "sin traducción" entonces los 4MB correspondientes no tienen direcciónes físicas asociadas

Paginación en 32 bits (de a página, dos niveles)

Tomemos una tabla de 1024 entradas, cada una traduce los 4MB *que le tocan*:

- Si hacemos entradas de 4 bytes este *directorio de páginas* ocuparía 4KB (¡Una página!)
- Si una entrada dice "sin traducción" entonces los 4MB correspondientes no tienen direcciónes físicas asociadas
- Sino, la entrada indica dónde ir a buscar la tabla que hace la "traducción fina" (la *tabla de páginas*)

Paginación en 32 bits (de a página, dos niveles)

Tomemos una tabla de 1024 entradas, cada una traduce los 4MB *que le tocan*:

- Si hacemos entradas de 4 bytes este *directorio de páginas* ocuparía 4KB (¡Una página!)
- Si una entrada dice "sin traducción" entonces los 4MB correspondientes no tienen direcciónes físicas asociadas
- Sino, la entrada indica dónde ir a buscar la tabla que hace la "traducción fina" (la *tabla de páginas*)
- Si hacemos entradas de 4 bytes esta *tabla de páginas* ocuparía 4KB también

Paginación en 32 bits (de a página, dos niveles)

- **Dirección del directorio de páginas (32 bits):** 20 bits identifican la página del directorio, el resto son atributos.
- **Directorio de páginas (4KiB / una página física):** Array de 1024 entradas. Cada entrada traduce los 4MB que le corresponden.
- **Tablas de páginas (cada una 4KiB / una página física):** Array de 1024 entradas. Cada entrada traduce los 4KB que le corresponden (dentro del bloque de 4MB de su tabla).
- **Entradas del directorio de páginas (32 bits):** 20 bits identifican la tabla de páginas asociada, el resto son atributos.
- **Entradas de las tablas de páginas (32 bits):** 20 bits identifican la página fisica asociada, el resto son atributos.

Paginación en 32 bits (de a página, dos niveles)

- **Dirección del directorio de páginas (32 bits):** 20 bits identifican la página del directorio, el resto son atributos.
- **Directorio de páginas (4KiB / una página física):** Array de 1024 entradas. Cada entrada traduce los 4MB que le corresponden.
- **Tablas de páginas (cada una 4KiB / una página física):** Array de 1024 entradas. Cada entrada traduce los 4KB que le corresponden (dentro del bloque de 4MB de su tabla).
- **Entradas del directorio de páginas (32 bits):** 20 bits identifican la tabla de páginas asociada, el resto son atributos.
- **Entradas de las tablas de páginas (32 bits):** 20 bits identifican la página fisica asociada, el resto son atributos.

Llamamos a esto *esquema de paginación* ó *estructura de paginación*

Traduciendo direcciones

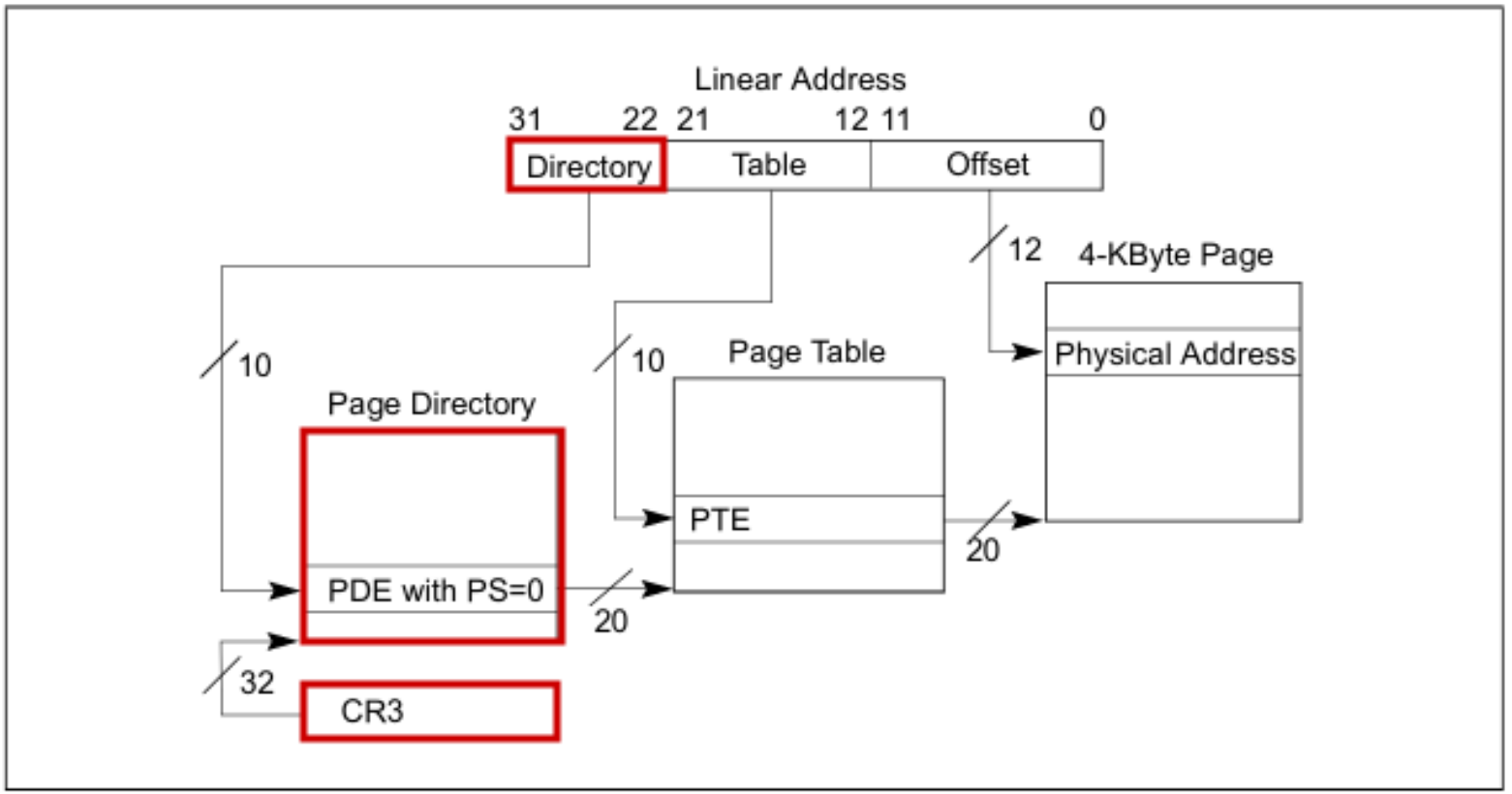


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Comenzamos por el **CR3**.

CR3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]																				Ignored							PCD	PWT	Ignored				**CR3**

La parte más importante del registro de control CR3 son los 20 bits más altos, ya que contienen el número de la página [1] donde se encuentra el directorio a utilizar para traducir las direcciones virtuales.

Dirección del directorio: `CR3 & 0xFFFFF000`

[1] Alcanzan 20 bits para un número de página porque los otros 12 son offsets dentro de la página ($\frac{2^{32}}{2^{12}} = 2^{32-12} = 2^{20}$)

Directorio de páginas

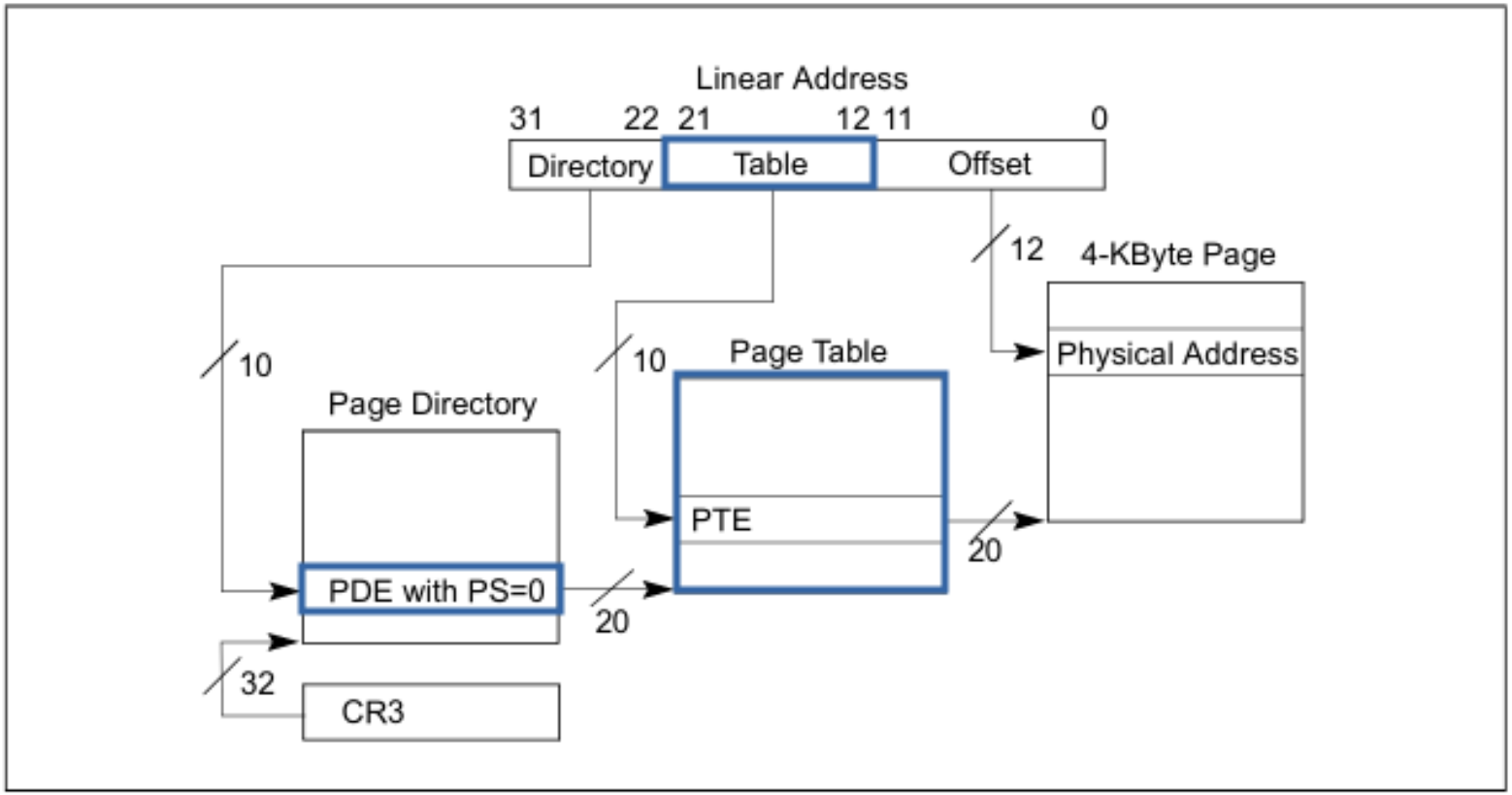


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Una vez identificado el directorio de páginas (**PD**, *page directory*) toca hacer foco en sus entradas (**PDE**, *page directory entry*).

Directorio de páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table

Tanto el directorio como la tabla de páginas tendrán 1024 entradas de 32 bits (4 bytes) por lo que ambos ocupan una página (4 KiB). Arriba vemos la estructura de una entrada del directorio. Los 20 bits más altos de la i-ésima entrada corresponden a el número de página de la i-ésima tabla de páginas.

Dirección de la *i*-ésima tabla: `pd[i] & 0xFFFFF000`

Tabla de páginas

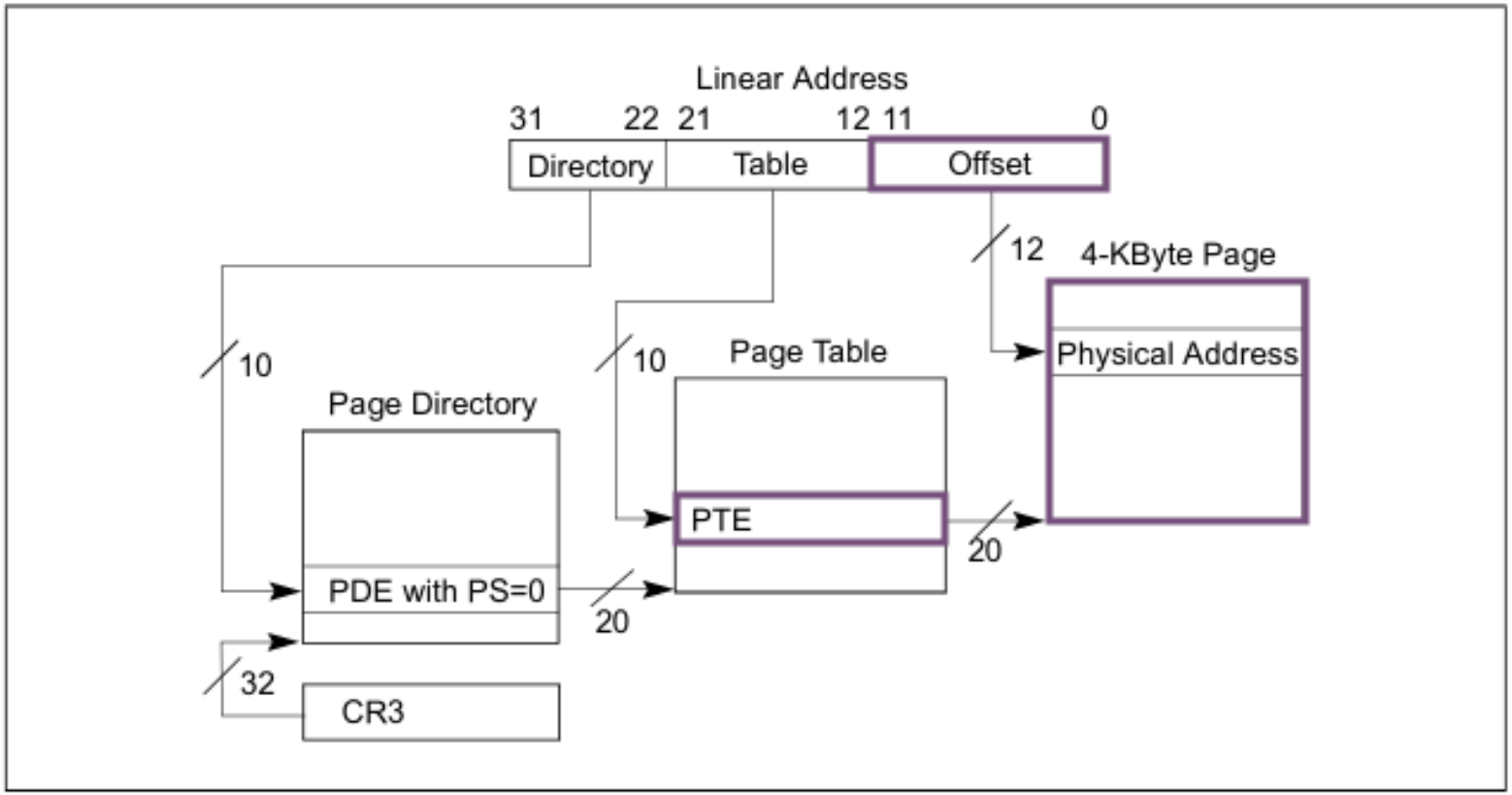


Figure 4-2. Linear-Address Translation to a 4-KByte Page using 32-Bit Paging

Finalmente toca ver la tabla de páginas (**PT**, *page table*) y sus entradas (**PTE**, *page table entry*).

Tabla de páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	**PTE: 4KB page**

Arriba vemos la estructura de una entrada de la tabla. Los 20 bits más altos de la i-ésima entrada corresponden a el número de página de la i-ésima página.

Dirección de la *i*-ésima página: `pt[i] & 0xFFFFF000`

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

```
pd := CR3 & 0xFFFFF000
```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

```
pd := CR3 & 0xFFFFF000
```

- Índice de PD (los 10 bits más altos de *virt*)

```
pd_index := (virt >> 22) & 0x3FF
```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

```
pd := CR3 & 0xFFFFF000
```

- Índice de PD (los 10 bits más altos de *virt*)

```
pd_index := (virt >> 22) & 0x3FF
```

- Dirección de PT (limpiamos la PDE):

```
pt := pd[pd_index] & 0xFFFFF000
```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

  ```
  pd := CR3 & 0xFFFFF000
  ```

- Índice de PD (los 10 bits más altos de *virt*)

  ```
  pd_index := (virt >> 22) & 0x3FF
  ```

- Dirección de PT (limpiamos la PDE):

  ```
  pt := pd[pd_index] & 0xFFFFF000
  ```

- Índice de PT (los 10 bits del medio de *virt*):

  ```
  pt_index := (virt >> 12) & 0x3FF
  ```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

  ```
  pd := CR3 & 0xFFFFF000
  ```

- Índice de PD (los 10 bits más altos de *virt*)

  ```
  pd_index := (virt >> 22) & 0x3FF
  ```

- Dirección de PT (limpiamos la PDE):

  ```
  pt := pd[pd_index] & 0xFFFFF000
  ```

- Índice de PT (los 10 bits del medio de *virt*):

  ```
  pt_index := (virt >> 12) & 0x3FF
  ```

- Dirección de la página (limpiamos la PTE):

  ```
  page_addr := pt[pt_index] & 0xFFFFF000
  ```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

  ```
  pd := CR3 & 0xFFFFF000
  ```

- Índice de PD (los 10 bits más altos de *virt*)

  ```
  pd_index := (virt >> 22) & 0x3FF
  ```

- Dirección de PT (limpiamos la PDE):

  ```
  pt := pd[pd_index] & 0xFFFFF000
  ```

- Índice de PT (los 10 bits del medio de *virt*):

  ```
  pt_index := (virt >> 12) & 0x3FF
  ```

- Dirección de la página (limpiamos la PTE):

  ```
  page_addr := pt[pt_index] & 0xFFFFF000
  ```

- Offset desde el inicio de la página (los 12 bits más bajos de *virt*):

  ```
  offset := virt & 0xFFF
  ```

Operaciones de la traducción

Queremos traducir la dirección

```
virt = dir(10 bits)|table(10 bits)|offset(12 bits)
```

- Dirección de PD (limpiamos CR3):

  ```
  pd := CR3 & 0xFFFFF000
  ```

- Índice de PD (los 10 bits más altos de *virt*)

  ```
  pd_index := (virt >> 22) & 0x3FF
  ```

- Dirección de PT (limpiamos la PDE):

  ```
  pt := pd[pd_index] & 0xFFFFF000
  ```

- Índice de PT (los 10 bits del medio de *virt*):

  ```
  pt_index := (virt >> 12) & 0x3FF
  ```

- Dirección de la página (limpiamos la PTE):

  ```
  page_addr := pt[pt_index] & 0xFFFFF000
  ```

- Offset desde el inicio de la página (los 12 bits más bajos de *virt*):

  ```
  offset := virt & 0xFFF
  ```

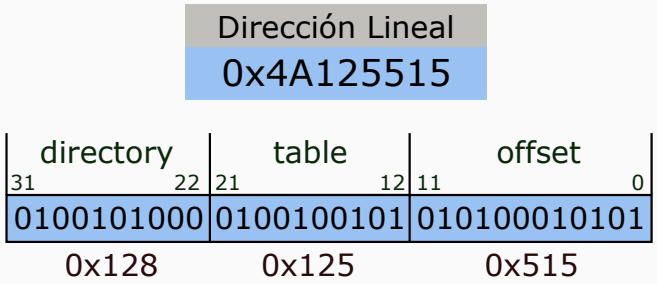
- Dirección física (sumamos la base de la página y el offset de *virt*):

  ```
  phys := page_addr | offset
  ```

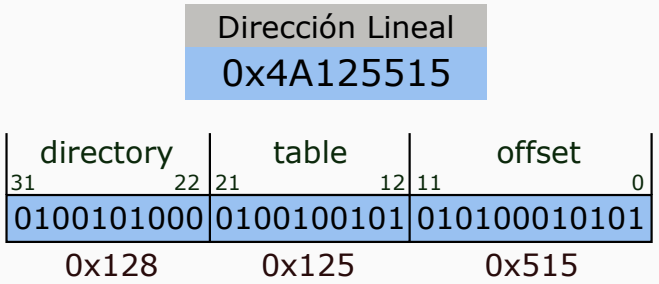
Ejemplo concreto

Dirección Lineal
0x4A125515

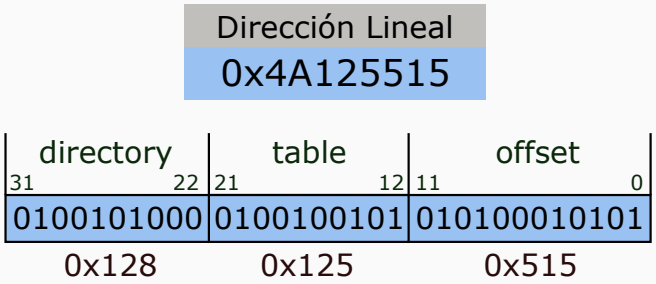
Ejemplo concreto

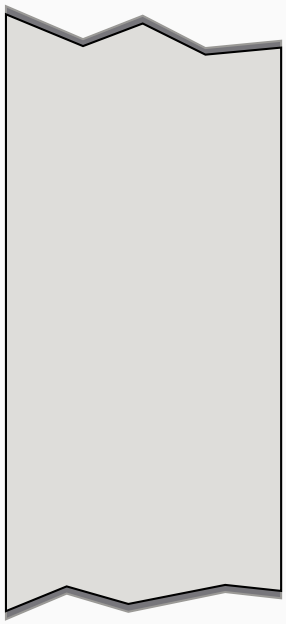


Ejemplo concreto

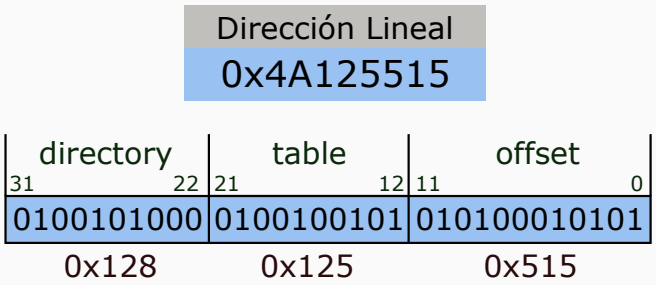


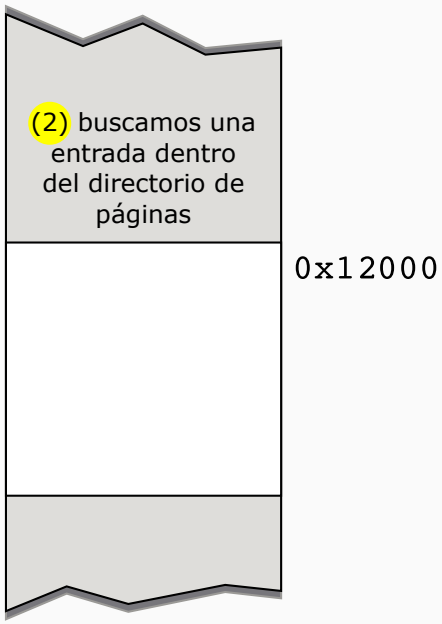
Ejemplo concreto





Ejemplo concreto



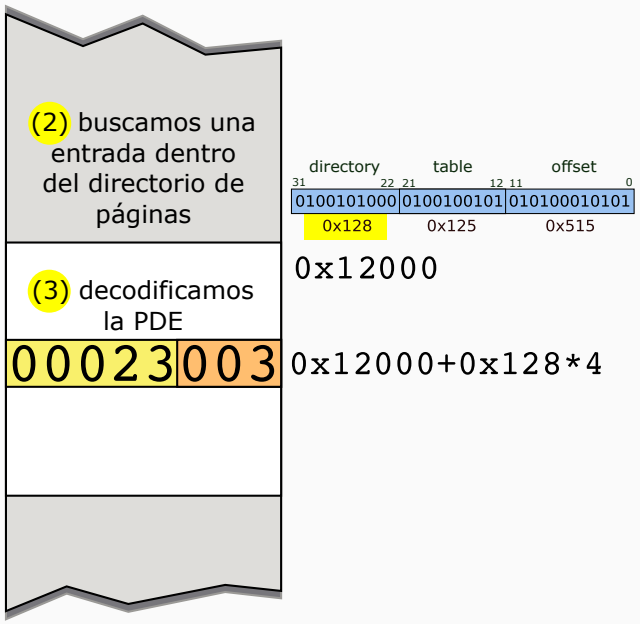


Page Directory

Ejemplo concreto

Dirección Lineal
0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515



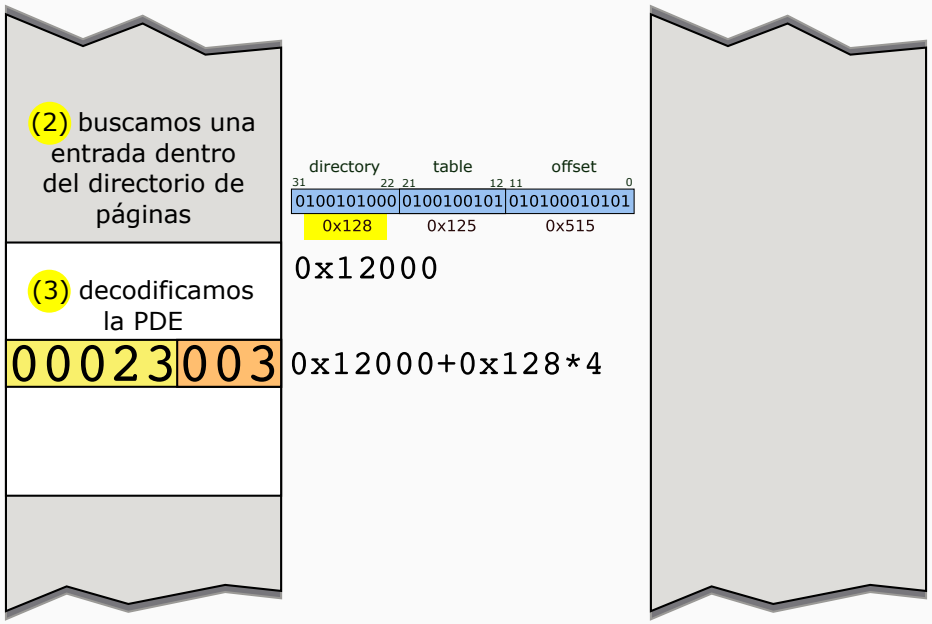
0x12000

0x12000+0x128*4

Ejemplo concreto

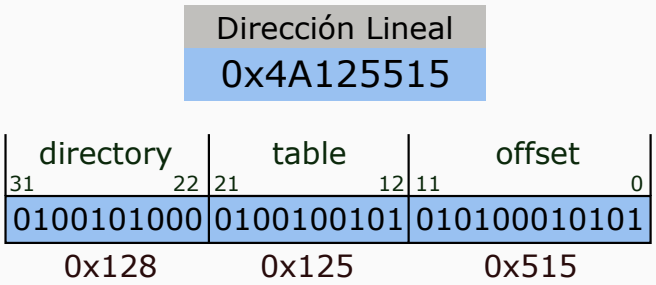
Dirección Lineal
0x4A125515

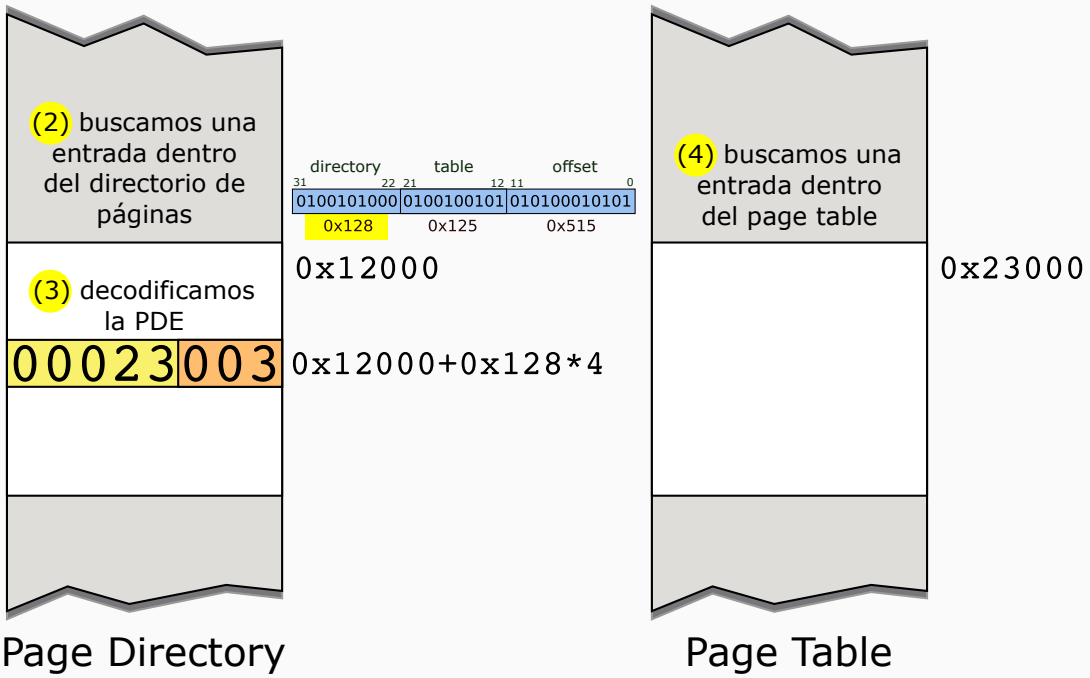
directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515



Page Directory

Ejemplo concreto

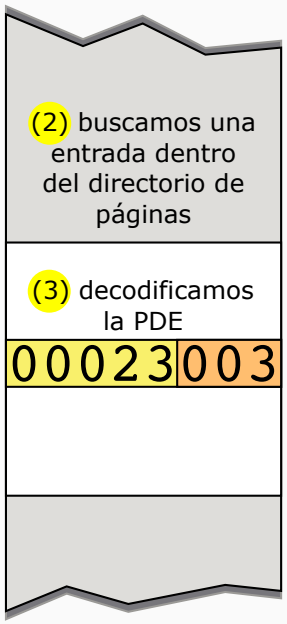




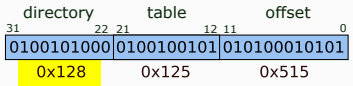
Ejemplo concreto

Dirección Lineal
0x4A125515

directory	table	offset
31 22	21 12	11 0
0100101000	0100100101	010100010101
0x128	0x125	0x515

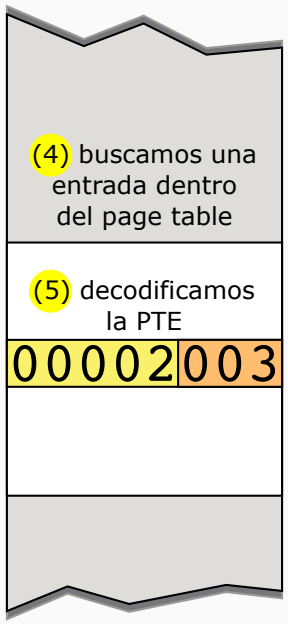


Page Directory

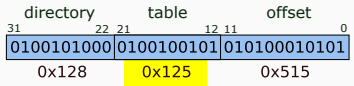


0x12000

0x12000+0x128*4



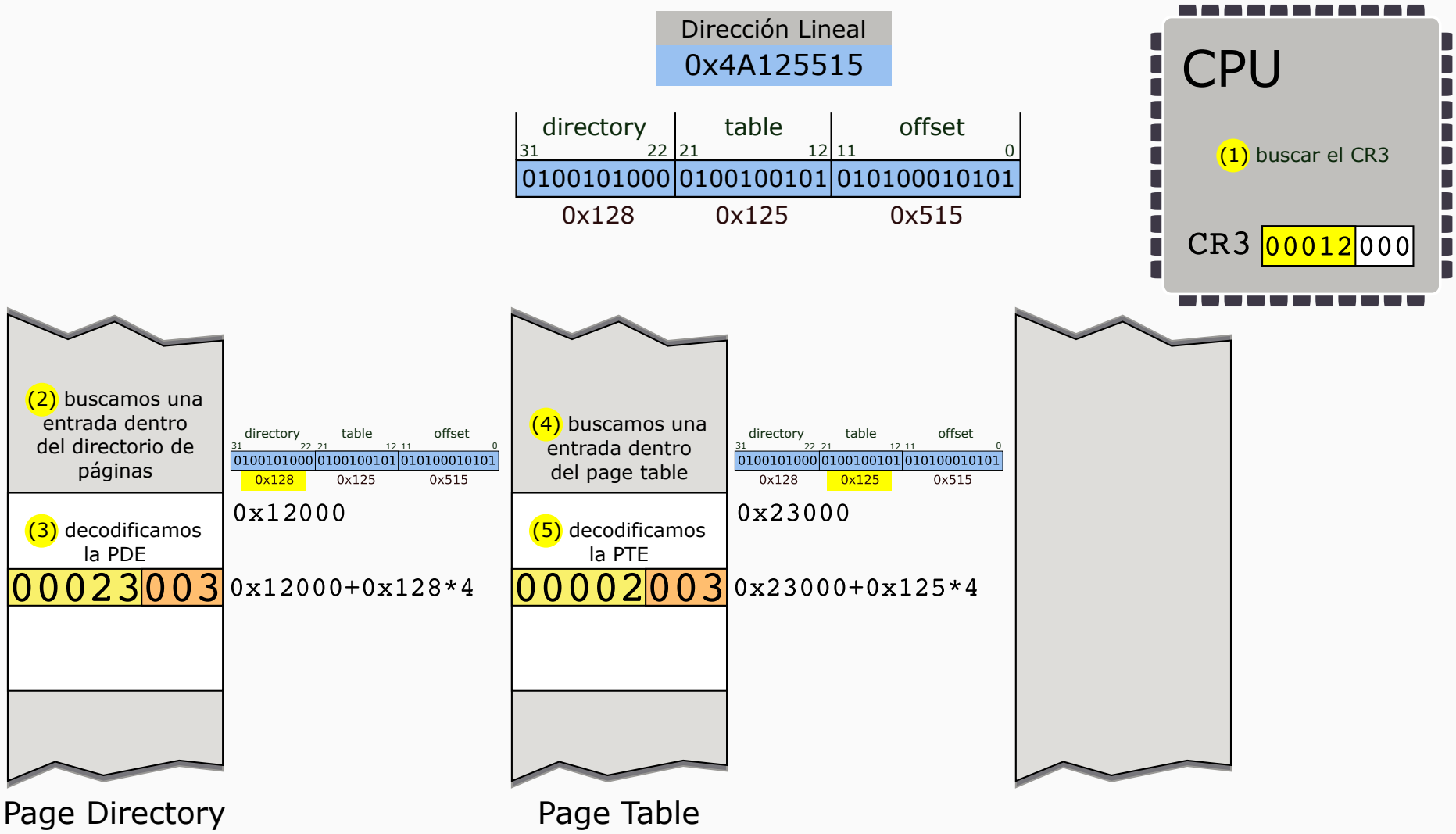
Page Table



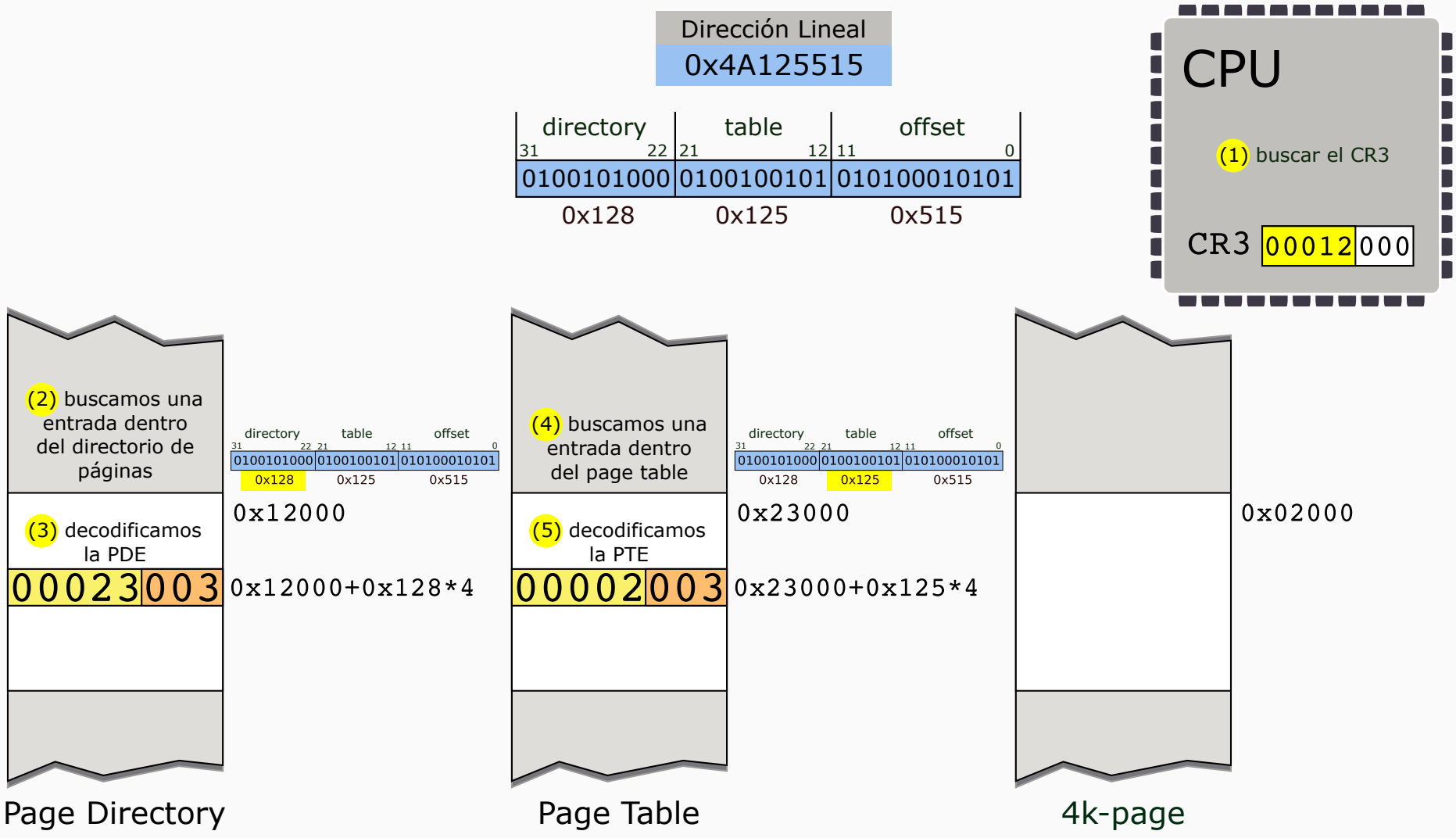
0x23000

0x23000+0x125*4

Ejemplo concreto

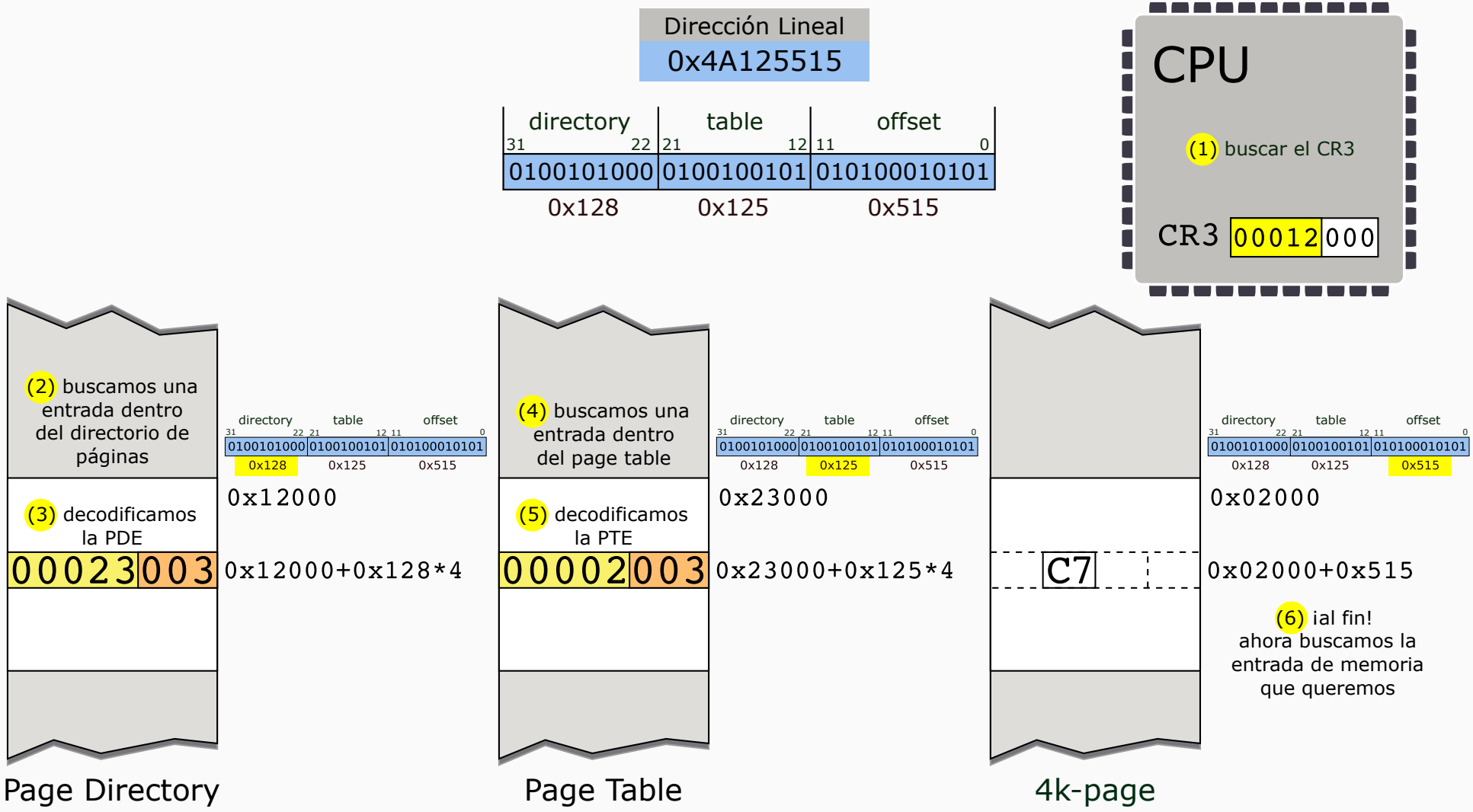


Ejemplo concreto



Ejemplo concreto

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA



TLB (translation lookaside buffer)

Para acelerar el proceso de traducción el procesador cuenta con una caché de traducciones tabla de traducciones comúnmente llamada *translation lookaside buffer*.

Cuando modifiquemos las estructuras de paginación es importante invalidar esta caché, de lo contrario podríamos observar traducciones cacheadas. Una forma de hacerlo es escribiendo en el registro `CR3`.

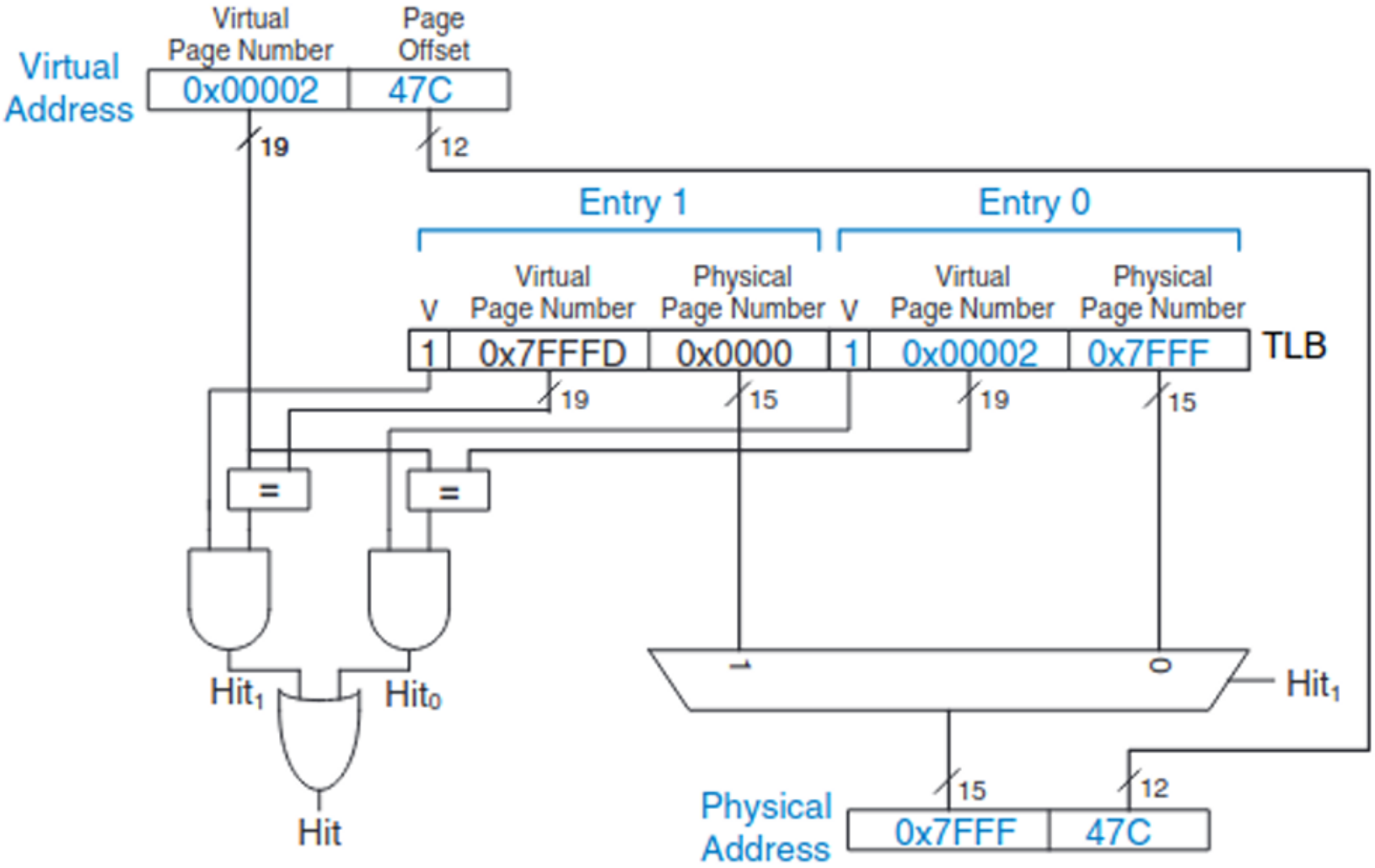
TLB (translation lookaside buffer)

Para acelerar el proceso de traducción el procesador cuenta con una caché de traducciones tabla de traducciones comúnmente llamada *translation lookaside buffer*.

Cuando modifiquemos las estructuras de paginación es importante invalidar esta caché, de lo contrario podríamos observar traducciones cacheadas. Una forma de hacerlo es escribiendo en el registro `CR3`.

Hay formas más sofisticadas (miren `invlpg`) pero `mov eax, cr3` seguido de `mov cr3, eax` es suficiente para nosotros. En el taller la rutina de invalidación ya está escrita y se llama `tlbflush`.

TLB (translation lookaside buffer)

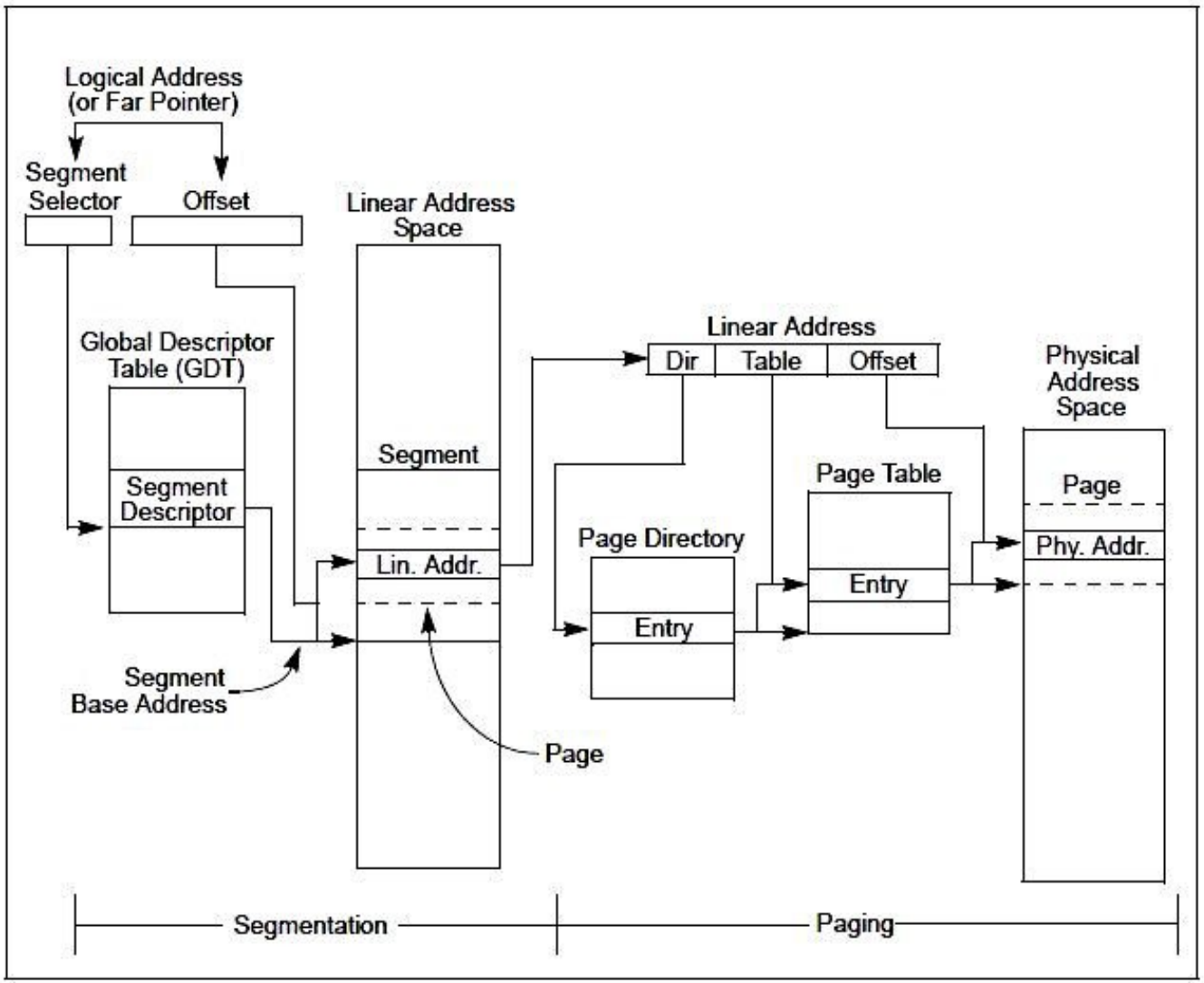


Segmentación + Paginación

Sólo para entender la interacción de las dos unidades, veamos como funciona la traducción al involucrar la unidad de segmentación.

El usar *segmentación flat* nos permite olvidarnos de la unidad de segmentación en el taller.

Segmentación + Paginación



Taller

Taller

En el taller van a tener que implementar algunas funciones relacionadas con la paginación y la unidad de manejo de memoria.

```
paddr_t mmu_next_free_kernel_page(void);
paddr_t mmu_next_free_user_page(void);
```

- Devuelve la dirección física de la próxima página de kernel/user disponible
- Las páginas de kernel y de usuario se encuentran en rangos de direcciones distintos, por lo que es necesario llevar un contador para cada uno

Taller

```
paddr_t mmu_init_kernel_dir(void);
```

- Genera el identity mapping de las primeras 4MB de memoria
- Recuerden setear correctamente los atributos de las páginas (W, S, P)
- ¿Cuántas páginas necesito para armar las tablas?

Taller

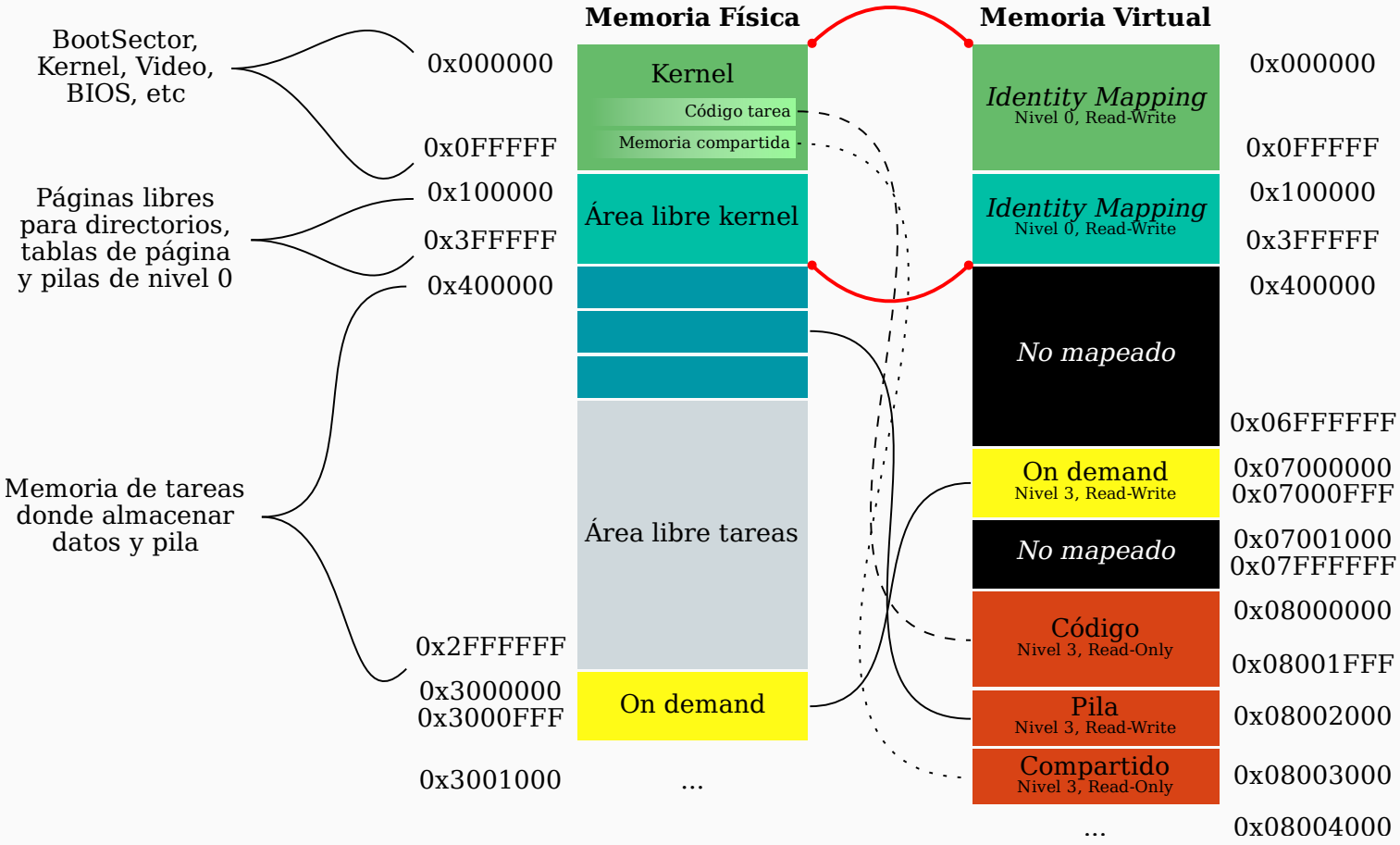
```
void mmu_map_page(uint32_t cr3, vaddr_t virt, paddr_t phy,
↪  uint32_t attrs);
```

- Genera un mapeo de una página virtual a una página física
- ¿En qué directorio? ¿Y si ya existe la page table?
- ¿Los atributos van para la PDE o la PTE?
- Recuerden llamar a `tlbflush`

Taller


```
paddr_t mmu_unmap_page(uint32_t cr3, vaddr_t virt);
```

- Desmapea una página virtual y devuelve la dirección de la página física desmapeada
- Pueden preguntar por Present y actuar en consecuencia
- Recuerden llamar a `tlbflush`


Memoria Física
Memoria Virtual
BootSector, Kernel, Video, BIOS, etc
Páginas libres para directorios, tablas de página y pilas de nivel 0
Memoria de tareas donde almacenar datos y pila
0x000000
0x0FFFFF
0x100000
0x3FFFFF
0x400000
0x2FFFFFF
0x3000000
0x3000FFF
0x3001000
Kernel
Código tarea
Memoria compartida
Área libre kernel
Área libre tareas
On demand
...
Identity Mapping
Nivel 0, Read-Write
Identity Mapping
Nivel 0, Read-Write
No mapeado
0x06FFFFFF
On demand
Nivel 3, Read-Write
0x07000000
0x07000FFF
No mapeado
0x07001000
0x07FFFFFF
Código
Nivel 3, Read-Only
0x08000000
0x08001FFF
Pila
Nivel 3, Read-Write
0x08002000
Compartido
Nivel 3, Read-Only
0x08003000
...
0x08004000

Taller


```
paddr_t mmu_init_task_dir(paddr_t phy_start);
```

- Genera el mapeo estático de una tarea del sistema
- Mapea 2 páginas de código, una de pila y una shared
- Recuerden mapear al kernel en los primeros 4MB

Cierre

Repaso de la introducción de hoy

En la introducción de hoy vimos:

Repaso de la introducción de hoy

En la introducción de hoy vimos:

- **Repaso sobre direcciones y memoria**

Repaso de la introducción de hoy

En la introducción de hoy vimos:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**

Repaso de la introducción de hoy

En la introducción de hoy vimos:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**
- **Ideas básicas de los mecanismos de paginación**

Repaso de la introducción de hoy

En la introducción de hoy vimos:

- **Repaso sobre direcciones y memoria**
- **Cosas que se pueden hacer con paginación**
- **Ideas básicas de los mecanismos de paginación**
- **Proceso de traducción y estructuras involucradas (`CR3`, `page directory`, `page table`)**

¿Consultas?

Anexo: Atributos del las estructuras

CR3

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page directory[1]																				Ignored							PCD	PWT	Ignored				**CR3**

- **20 bits más altos del `CR3`:** Número de la página física donde está el directorio actual
- **PCD (Page Cache Disable):** Deshabilita cachear entradas del *page directory*
- **PWT (Page Write-Through):** Deshabilita hacer write-back cuando el procesador modifica el *page directory*

Directorio de páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table

- **20 bits más altos del `PDE`:** Número de la página física donde está la tabla de páginas asociada
- **A (Accessed):** Indica si se accedió a memoria controlada por esta PDE. Lo escribe el procesador al traducir
- **PCD (Page Cache Disable):** Deshabilita cachear entradas de la *page table* asociada
- **PWT (Page Write-Through):** Deshabilita hacer write-back cuando el procesador modifica la *page table* asociada

Directorio de páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of page table																				Ignored				0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table

- **U/S (User/Supervisor):** Determina si un proceso en modo usuario puede acceder a la memoria controlada por esta PDE
- **R/W (Read/Write):** Determina si un proceso puede escribir a la memoria controlada por esta PDE
- **P (Present):** Es el bit 0 (siempre en uno), indica que ésta traducción es válida

Tabla de páginas

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Address of 4KB page frame																				Ignored			G	PAT	D	A	PCD	PWT	U/S	R/W	1	**PTE: 4KB page**

- **20 bits más altos del `PDE`:** Número de la página física a la que corresponde esta traducción
- **G (Global):** Marca la traducción como global. Las traducciones globales no se invalidan al cambiar el `CR3`.
- **PAT (Page Attribute Table):** Un feature del procesador que no vamos a usar. Permie un control más granular del mecanismo de caché.
- **D (Dirty):** Indica si se escribió a memoria controlada por esta PTE. Lo escribe el procesador al traducir
- **A (Accessed):** Indica si se accedió a memoria controlada por esta PTE. Lo escribe el procesador al traducir

Tabla de páginas

31 … 12	11 … 9	8	7	6	5	4	3	2	1	0	
Address of 4KB page frame	Ignored	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page

- **PCD (Page Cache Disable):** Deshabilita cachear los datos de página asociada
- **PWT (Page Write-Through):** Deshabilita hacer write-back al escribir en la página asociada
- **U/S (User/Supervisor):** Determina si un proceso en modo usuario puede acceder a la memoria controlada por esta PTE
- **R/W (Read/Write):** Determina si un proceso puede escribir a la memoria controlada por esta PTE
- **P (Present):** Es el bit 0 (siempre en uno), indica que ésta traducción es válida