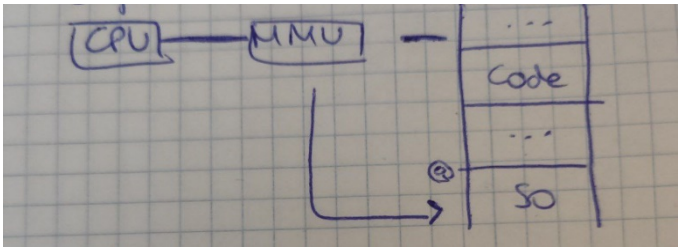
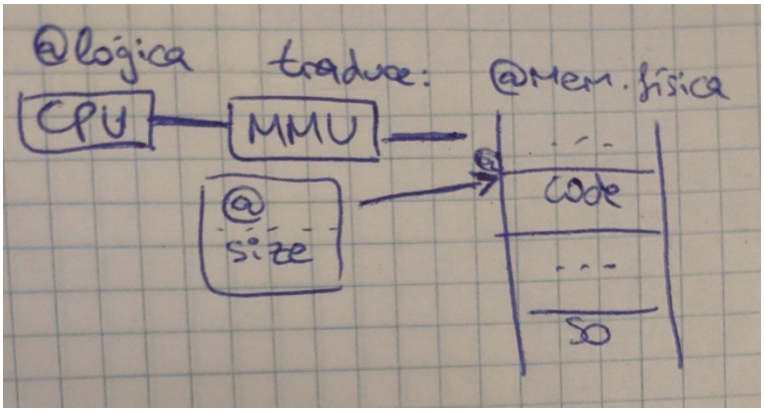
# TEMA 3 Memoria

Al principio, el espacio de direcciones del SO y del usuario era el mismo, y no existían los niveles de privilegio. Para solucionar esto, se crea la MMU.

El origen de la MMU era bloquear el acceso al código del SO, colocado en la parte inicial de la memoria, por lo que la MMU contenia la @alta del SO, y si intentaba acceder a una @física menor, saltaba excepción y mataba el proceso.

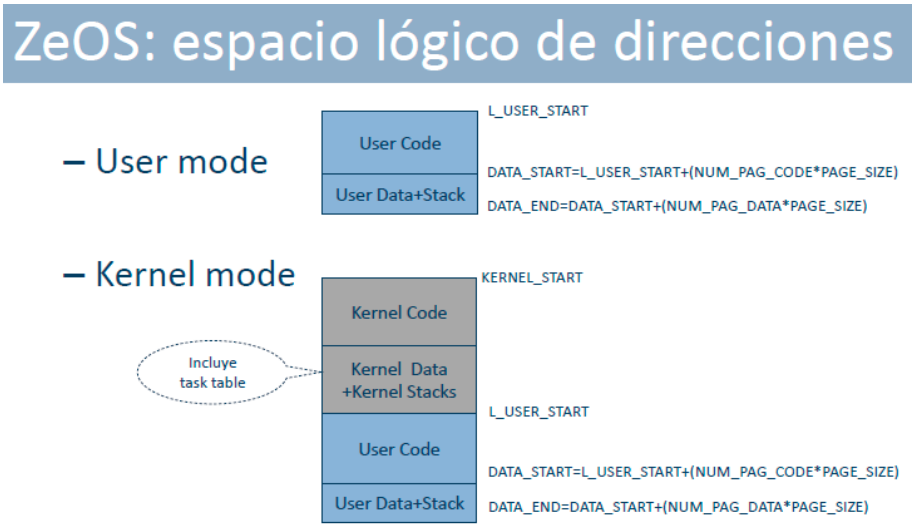



Con el tiempo, se estableció que, para ser compatibles con otros sistemas, y por seguridad (asignación de memoria aleatoria), los programas compilarían en la dirección 0 de su espacio de memoria. Así, sus direcciones de código son offsets, y lo único que se traduce es la ID de física a lógica y viceversa. Para estas traducciones se mejora la MMU, que a partir de ahora se encargará de traducir, sin dejar de proteger al SO.

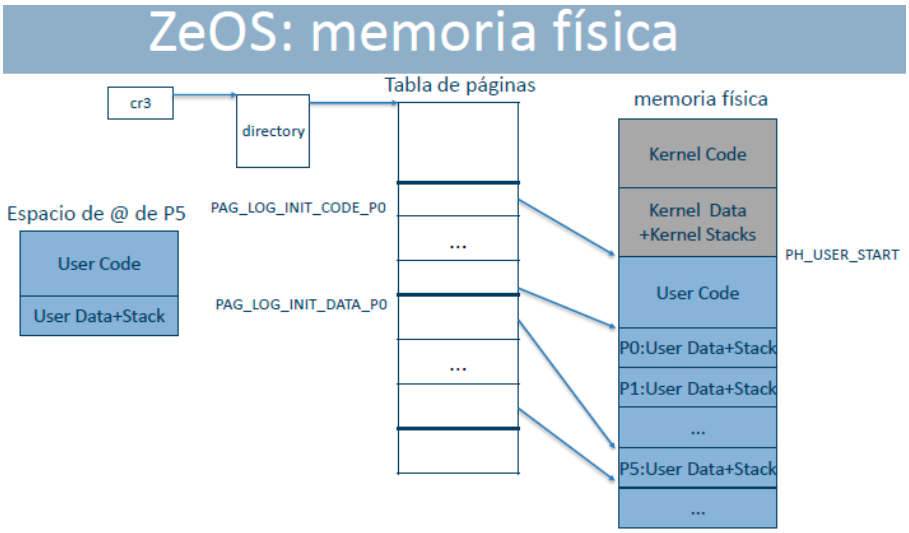



Ahora, la MMU contiene dos registros adicionales en la MMU: el registro base y el registro límite. Estos registros se utilizan para definir un rango de direcciones físicas que están asignadas a cada proceso en el sistema. El registro base contiene la dirección física del inicio de la memoria asignada a un proceso, mientras que el registro límite indica la longitud de esa memoria. Así, compara si la @ es menor que el size para saber si vamos a acceder a la memoria del proceso, y no a corromper otra parte del código de otro proceso o del SO. Estos regs se van cambiando según el proceso que se va ejecutando.

**Cada proceso tiene un bloque de memoria lógica, y todos comparten la memoria física.**

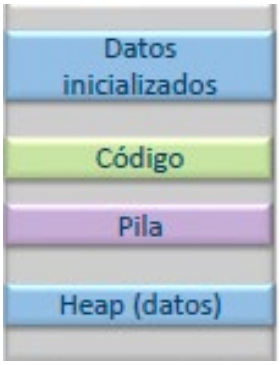



**Para un mismo proceso tenemos este espacio lógico, para user y sistema, con una pila user y otra de sistema para cada proceso.**




**La memoria lógica se divide en segmentos, y la memoria física en marcos de páginas, el tamaño de segmento ha de ser múltiplo del tamaño de página.**

La segmentación divide un programa en segmentos lógicos más pequeños y separados, como el código, los datos, la pila y el heap (memoria dinámica). Cada segmento asigna a un área separada en la memoria física, y se puede asignar permisos de acceso diferentes según la parte del programa. Estos segmentos se reparten por toda la memoria física según los huecos que hay libres.



## FRAGMENTACIÓN ##

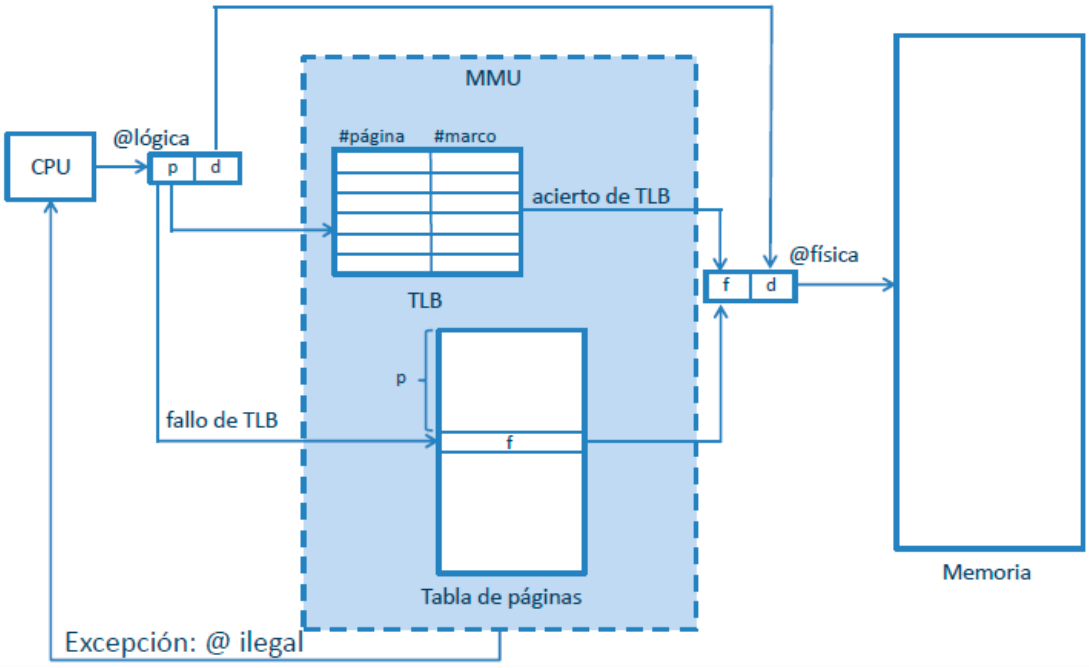
El código del proceso ha de estar totalmente consecutivo en memoria. Esto trae problemas de fragmentación, puesto que cada proceso tiene un tamaño e intentar encajarlos conlleva a que queden huecos libres entre los procesos.

Este problema se solucionó al principio con: reubicación dinámica --> cada cierto tiempo el SO va mirando el mapa de la ocupación de la memoria física, si ve que hay mucha fragmentación, mueve los procesos en la memoria para agruparlos y concentrar los huecos en uno solo.

Al principio, esto funcionaba bien porque los procesos son pequeños, pero con el tiempo los procesos son más grandes, y cuesta más la reubicación.

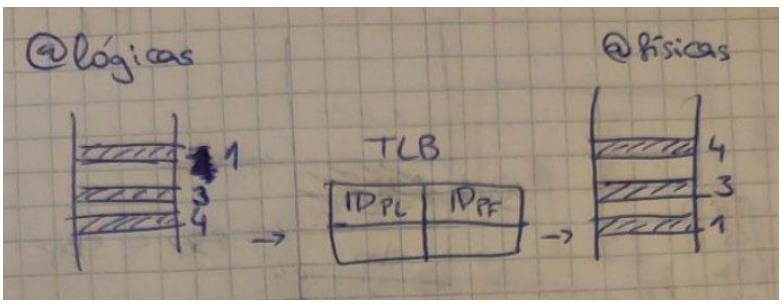
A raíz de esto aparece la **paginación**, y con ella la TLB (Translation Lookaside Buffer).

Una **TLB** es una única estructura hardware del procesador que se encuentra en la MMU junto a la tabla de páginas.




El SO ha de saber cómo trabajar con él. El TLB traduce las direcciones físicas y las direcciones virtuales. Las direcciones lógicas siempre serán consecutivas, pero el rango de espacio no ha de ser consecutivo necesariamente.

En memoria física hacemos lo mismo, se divide “virtualmente” en páginas físicas con direcciones consecutivas (pero no seguidas en el espacio necesariamente).

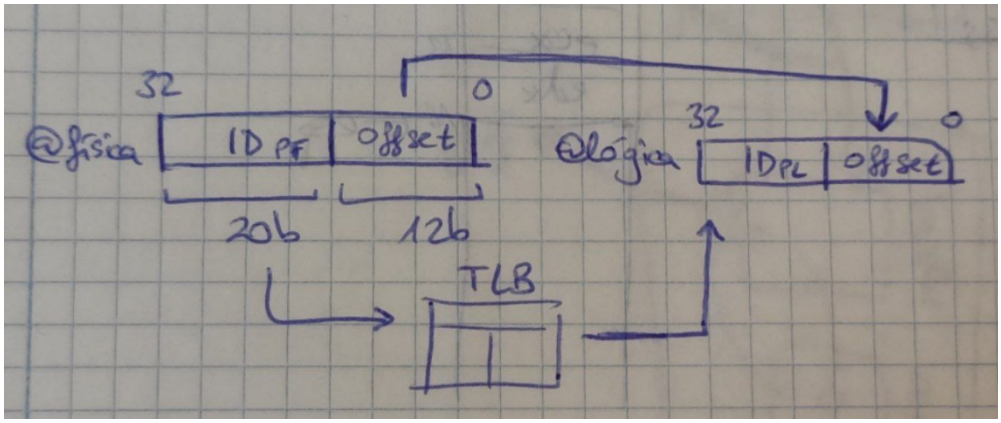


## TLB ##

La TLB traduce una @lógica a una @física muy rápidamente, es decir un ID de pág. lógica (20bits) a ID de pág. física (20bits).

Esta estructura suele tener 512 o 1024 entradas, y estas normalmente son de 4KB (2^12@, 12 bits).

De los 32 bits de una @, los 12 de menos peso serán igual en ambas @, el offset, y el resto (20bits) serán el ID de página lógica/física, y servirá para identificarlas.



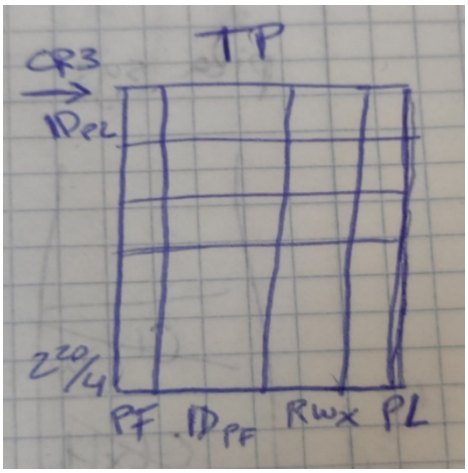

Al pasarle el ID a la TLB, habrá un **hit** si lo encuentra y lo traduce, si no, se produce un **miss, o fallo de TLB**, en este caso la TLB no es capaz de traducir y deberá acceder al registro CR3.

El **registro CR3** apunta a la primera @ de la tabla de páginas del proceso actual. En el cambio de proceso cambia de CR3, puesto que cada proceso tiene su tabla de páginas, en lo que también hay que reiniciar la TLB.

Al acceder a este registro, accede a la tabla de páginas y busca la ID que tiene que traducir.

## TABLA DE PÁGINAS ##

La **tabla de páginas** también está en la MMU y es un vector con 1024 entradas ($2^{20}$ / 4KB) **con tantas entradas como páginas lógicas**, donde almacena la traducción de estas páginas lógicas a físicas, por lo que **cada proceso tiene sus propias traducciones y su propio rango de memoria física**.



Cada entrada contiene una ID de página lógica, y su ID de página física asociada, y un bit de presencia PF, que indica si la entrada es válida.

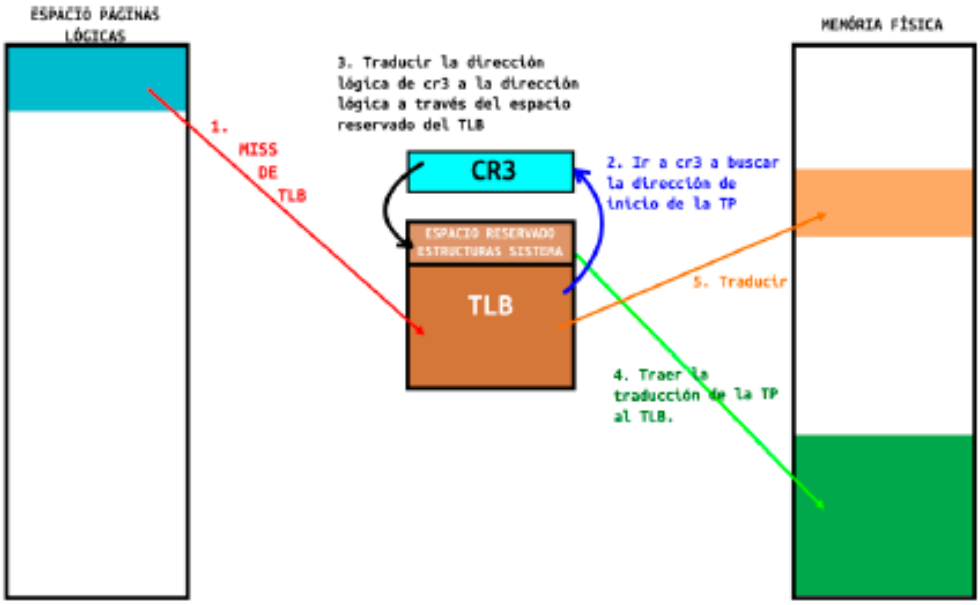
Una vez encontrada la entrada dentro de la TP del proceso, busca una entrada de la TLB que pueda sustituir (normalmente usa un algoritmo, ej. LRU, elimina la más antigua sin usar) y guarda la traducción.

## TRADUCCIÓN ##

**La paginación y la traducción se activan en tiempo de boot, a partir de entonces se trabaja con @lógicas**, y a partir de ahí ya no se desactiva la traducción, todo lo que se ejecute tiene que trabajar con @lógicas, el SO trabaja con @lógicas, por lo tanto, esta traducción de páginas del sistema, es decir, **la tabla de páginas está en la memoria física del SO, y cada una contiene la traducción lógica-física del proceso y del sistema.**

Por esto, para acceder a la TP, necesita la dirección física donde esta se encuentra, la @lógica está guardada en cr3, por lo que la CPU lee de este una @lógica que ha de traducir para acceder a la ubicación física, **por esto la TLB necesita pasar por sí misma para traducir la dirección contenida por cr3 y acceder a la @física de la TP**, por lo que sin la TLB no puede haber paginación. Esta traducción está en una parte reservada de la TLB para la memoria del sistema operativo.

La TP de cada proceso solo es accesible desde modo sistema puesto que están en memoria del SO, pero se puede consultar desde modo usuario puesto que siempre vamos a necesitar hacer traducciones, por tanto, dentro de la TP tendremos direcciones del sistema a la que solo puede acceder este, y direcciones del proceso, y se diferencian con los niveles de privilegio puestos a cada entrada.




Pero debemos asegurarnos de que en la TLB los accesos al espacio de memoria del SO estén restringidos, por lo que se asignan 2 bits para definir **el nivel de privilegios para cada entrada de la TP**, para el espacio del SO se asignan los privilegios más altos.

Para evitar que un proceso modifique las tablas para acceder a zonas de memoria que no le corresponden, del SO o de otros procesos, también se asignan unos bits para definir los **permisos para cada entrada de la TLB**, que también están en la TP y que se traen con la asociación a la TLB, y que son 3 bits: R/W/X. Si por ejemplo un proceso solo tiene permiso de R y W sobre una página, no podrá ejecutar código en ella.

##FALLO DE PÁGINAS##

El bit de presencia representa si existe una asociación entre página lógica y física. El SO llena toda la memoria lógica pero deja huecos, no tienen nada, no están asignadas. Solo se asignan páginas que contienen algo.


Cuando se queda sin espacio, **se crea asociación con la memoria física**, por lo que puede haber páginas lógicas sin asociación. El bit de presencia representa con un 0 que la página lógica que se quiere traducir no tiene asociada ninguna página física.

Cuando hay un acceso a la TLB y esta falla, lee el cr3, traduce la @ de la TP, se accede a esta. Cuando hay un fallo en la TLB, se activa el modo sistema, que lee en CR3 la @ de la TP del proceso actual, y accede a esta para traer la traducción, ya que el proceso no puede acceder a la memoria del SO donde se encuentra esta TP.

Mira el bit de presencia para la entrada de la página lógica correspondiente. Si está PF=1, es que está traducida, la coge y se la lleva a la TLB, donde al siguiente acceso será hit.
Si PF=0, entonces hay un **Page Fault**. Es imposible que haya una traducción física-lógica con P=0, pero **es posible que haya una traducción física - virtual en disco.**

A través del software se introducen mejoras en la gestión de la memoria. Una de ellas es la **memoria virtual**. La memoria física alberga datos que a los que no se acceden muy a menudo o no se utilizan. Lo que queremos es liberar un poco esta memoria física. Utilizamos el disco para llevarnos datos no muy accesibles, ya que son accesos lentos, extendiendo así la memoria física.

Existe un **Pager Daemon**, o paginador, que nunca se ejecuta en modo usuario (como todos los daemon). Son procesos con prioridad muy alta con nivel de privilegios 0. Este paginador lleva el tracking de la memoria física, se encarga de monitorizarla para hacer una gestión eficiente de esta. De vez en cuando, toma decisiones sobre liberación de memoria. El paginador mira el proceso actual, valora el proceso, si va a seguir adelante, etc. Luego mira la tabla de páginas y guarda todo el contenido de una página en el disco. Así, **va guardando un listado de páginas que va usando**

**ese proceso, en este caso, guarda esta página en el disco y la pone en el historial, y cambia el bit PF a 0 en la TP.**

**Es decir, el paginador traduce las páginas físicas a páginas virtuales, y la traducción se guarda en la tabla de páginas global a la que solo puede acceder el sistema.**

Cuando salta un page fault, puede ser que no exista ni haya existido una asociación entre ellas, pero también puede ser que la página esté en el disco en lugar de en la memoria física. La excepción le pasa la ID de la página que la causó a través de un parámetro, **CR2,** que guarda el ID lógico que ha hecho saltar el page fault (sólo las @lógicas generan page fault, pero es solo el ID porque el offset no hace falta), y el paginador mira en el historial si en algún momento fue movida al disco.

Si no la tiene, significa que hubo hueco siempre, entonces genera un **segmentation fault**. Si la tiene, la coge del disco y la vuelca en la página física, actualiza el PF en la tabla de páginas y actualiza la traducción, indicando que ahora sí tiene una página asignada, y devuelve la excepción que se había generado.

Ahora, que se ha modificado la tabla de páginas, pero no la TLB, se provocará un fallo de TLB porque no tiene la traducción, mirará la tabla y verá que hay PF=1, y actualiza la traducción, resolviendo el fallo.

El disco tiene una parte reservada para hacer esto, en Linux es una partición llamada swap.

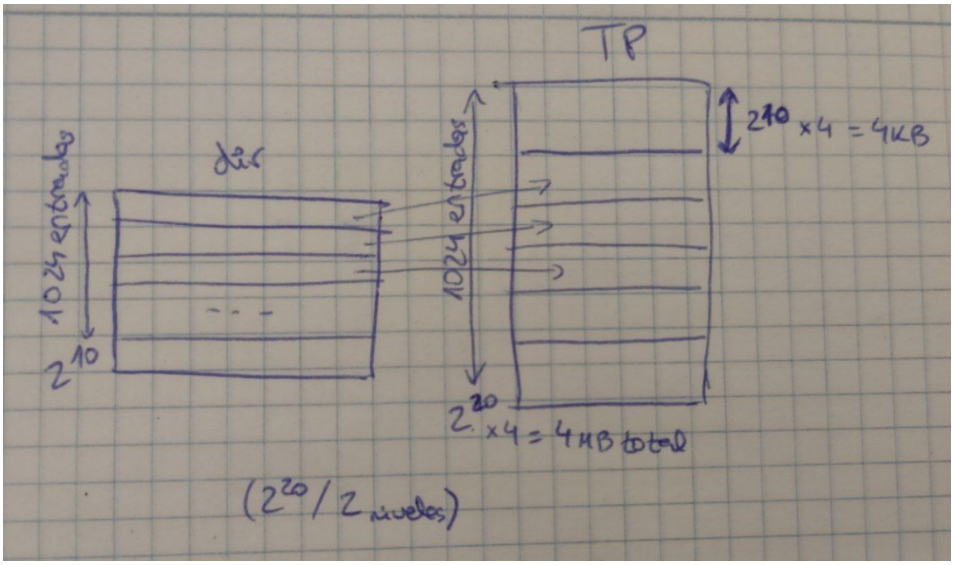
## PAGINACIÓN ##

Una tabla de páginas tiene $2^{20}$ entradas x 4B por entrada, eso son 4MB por tabla. Una de esas para cada proceso es muchísima memoria, además de que algunas páginas están vacías y no se traducen.

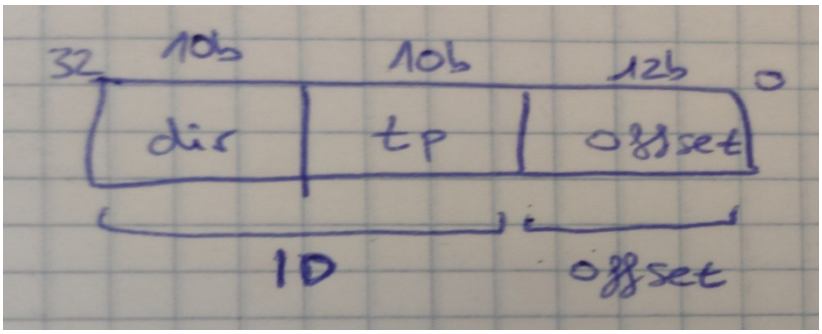
Por suerte, no nos hace falta mapear toda la memoria si no todas las páginas van a ir al disco.

La solución es la **tabla de páginas multinivel (2 niveles)**: Lo que hacemos es dividir la tabla de páginas (de tamaño $2^{20}$ * 4) en segmentos de $2^{10}$ entradas, 1024 entradas de 4KB cada una.

También tendremos un directorio con 1024 entradas, cada cual contendrá un puntero a un segmento de la tabla de páginas y un PF que indica si tiene traducción. Lo más inteligente sería agrupar entradas para que entren las máximas posibles en cada segmento.



Las direcciones se partirán:




**Ahora, el CR3 apuntará al directorio de páginas.**

Y el proceso de traducción será de la siguiente manera: