

Universidad de Murcia
Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Estructura y Tecnología de Computadores

Tema 6: Jerarquía de memoria — Memoria virtual

Apuntes

CURSO 2010 / 11

VERSIÓN 2.0

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores



Índice general

6.1. Introducción	1
6.1.1. Conceptos generales	2
6.1.2. Consideraciones de diseño de un sistema de memoria virtual	4
6.2. La tabla de páginas	4
6.3. Tratamiento de los fallos de página	5
6.4. TLB (Translation–Lookaside Buffer)	7
6.5. Implementación de la protección con memoria virtual	10
6.6. Un marco común para las jerarquías de memoria	11
6.7. Jerarquía de memoria para la DECSTATION 3100	13
6.7.1. Memoria caché	13
6.7.2. Memoria virtual	17
B6. Boletines de prácticas	18
B6.1. Funcionamiento de la memoria virtual en Linux	18
B6.1.1. Objetivos	18
B6.1.2. Prerequisitos	18
B6.1.3. Plan de trabajo	18
B6.1.4. Órdenes para mostrar el uso de memoria	19
B6.1.5. Uso de la memoria en programas	21
B6.1.6. Ejercicios	23
E6. Ejercicios	25
E6.1. Jerarquía de memoria — Memoria virtual	25
E6.2. Solución a ejercicios seleccionados	35

6.1. Introducción

Como hemos visto en el tema anterior, la memoria caché es una pequeña memoria que, aprovechando el principio de localidad, nos permite acceder rápidamente a instrucciones y datos almacenados en una memoria más lenta (la memoria principal). De la misma forma, la memoria principal puede actuar como una “caché” para el almacenamiento secundario, constituido, normalmente, por uno o más discos duros. Esta técnica se llama *memoria virtual*. Hay dos razones principales por las que se utiliza memoria virtual:

1. **Eliminar el inconveniente de tener un espacio de memoria principal pequeño y limitado:** cuando no existía memoria virtual en las máquinas, si un programa era demasiado grande para la memoria principal existente, el programador tenía que resolver el problema. La solución habitual era dividir los programas en partes y después identificar los fragmentos que eran mutuamente excluyentes (*overlays*). Estas partes eran cargadas en memoria o salvadas en disco por el propio programa durante su ejecución, siendo responsabilidad del programador asegurar que el programa nunca accedía a una parte que no estaba ya en memoria y que las partes cargadas en un momento dado no excedían el tamaño de la memoria principal. Como podemos entender, esta manera de programar se convertía en una pesadilla para los programadores.

El *sistema de memoria virtual* soluciona el problema anterior de la siguiente forma. Por un lado, creando un espacio de direcciones virtuales (es decir, una *memoria virtual*) que el programa manejará “como si hubiera” una gran memoria principal. Por otro lado, gestionando automáticamente las transferencias

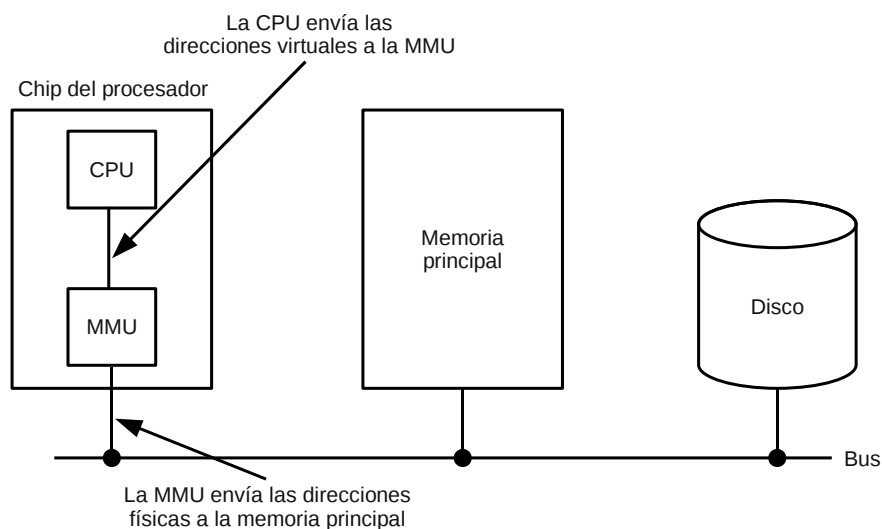


Figura 1: Procesador con soporte para memoria virtual.

de información entre los dos niveles de la jerarquía de memoria involucrados: la verdadera memoria principal de la máquina (que llamaremos *memoria física*) y la memoria secundaria. Ya que la memoria virtual (la usada por el programa) tendrá, generalmente, un tamaño mayor que la memoria física, en un momento dado sólo habrá una parte de la memoria virtual en la memoria principal de la máquina, sin que el programa se entere de ello. Donde en todo momento estará presente todo el contenido de la memoria virtual que maneja un programa (instrucciones y datos) será en la memoria secundaria.

2. **Permitir la compartición eficiente y sin peligros de memoria entre múltiples programas:** cuando varios programas se están ejecutando al mismo tiempo en una máquina, la memoria requerida por todos esos programas puede ser mayor que el espacio de memoria principal libre. Sin embargo, como hemos dicho anteriormente, sólo una parte de esa memoria se usa activamente en un momento dado, por lo que la memoria principal sólo tiene que contener las partes en uso de los programas en ejecución. Esto permite ejecutar varios programas a la vez, lo que a su vez permite compartir más eficazmente el procesador y la memoria principal. Evidentemente, para posibilitar que múltiples programas compartan la misma memoria, hay que proteger a unos de otros, asegurando que un programa sólo pueda leer y escribir la memoria que tiene asignada.

6.1.1. Conceptos generales

Los conceptos de funcionamiento de la memoria virtual y la memoria caché son iguales. Sin embargo, las diferentes raíces históricas han llevado al uso de diferentes terminologías. Así, un bloque de memoria virtual se llama “página”, un acierto de memoria virtual se llama “acierto de página” y un fallo se llama “fallo de página”.

Como se muestra en la figura 1, con memoria virtual, la CPU produce *direcciones virtuales* y la unidad de gestión de memoria o **MMU** (*Memory Management Unit*) las traduce a *direcciones físicas*, que serán usadas para acceder realmente a la memoria principal (o a la memoria caché si hubiera). Es decir, la clave de la memoria virtual está en la *traducción de direcciones* (*address translation*).

La traducción de direcciones supone que toda dirección virtual generada por la CPU debe convertirse en una dirección física. Sin embargo, para que esta traducción sea eficiente en cuanto al tiempo y al uso de recursos, el sistema de memoria virtual no puede guardar todas las asociaciones dirección virtual–dirección física posibles. En su lugar, el sistema divide la memoria virtual (es decir, el conjunto de todas las direcciones virtuales posibles) en *páginas virtuales* y la memoria física (es decir, el conjunto de todas las direcciones

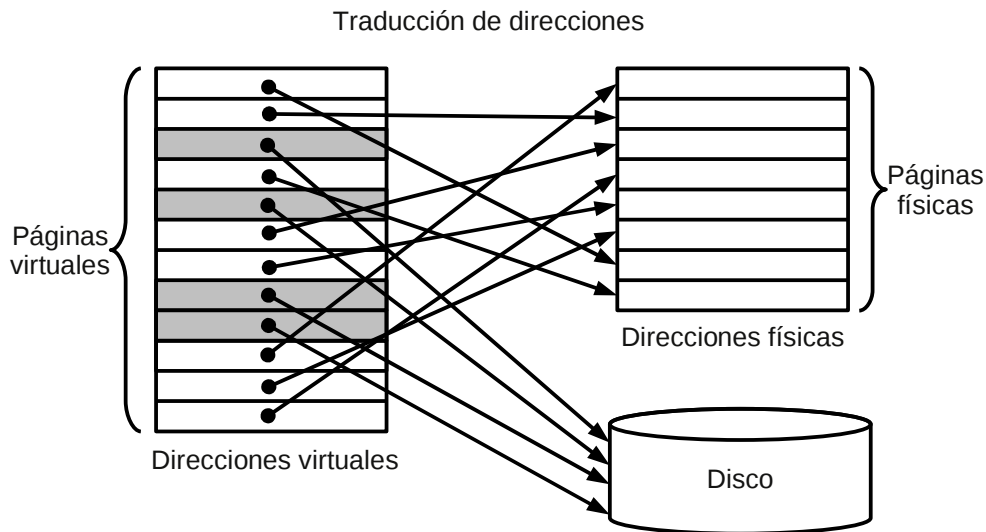


Figura 2: Traducción de direcciones: conjunto de direcciones virtuales → conjunto de direcciones físicas.

físicas posibles) en *páginas físicas* y guarda todas las asociaciones página virtual–página física posibles. Estas asociaciones se muestran gráficamente en la figura 2.

Como la memoria virtual suele ser más grande que la memoria física, habrá más páginas virtuales que físicas, por lo que habrá páginas virtuales asignadas a páginas físicas y páginas virtuales guardadas en disco (ver figura 2). Una página virtual en disco se cargará en memoria y se asignará a una página física cuando el programa en ejecución (o, lo que es lo mismo, la CPU) acceda a alguna de sus direcciones virtuales, lo que dará lugar a un *fallo de página*.

La memoria virtual también simplifica la carga en memoria de un programa a ejecutar al proporcionar *reubicación*. En un sistema sin memoria virtual, un programa ocupa generalmente una única zona de memoria contigua por lo que el sistema operativo debe buscar un hueco del tamaño adecuado antes de poder cargar y ejecutar el programa. Si la memoria libre está muy fragmentada (hay muchos huecos libres pequeños) puede ocurrir que, aún habiendo memoria libre suficiente, un programa no se pueda cargar al no encontrar un hueco suficientemente grande.

En un sistema con memoria virtual, sin embargo, un programa se divide en páginas virtuales. Ya que una página virtual se puede asignar a cualquier página física, el sistema operativo no necesitará encontrar un único hueco de memoria libre sino que simplemente necesitará encontrar suficientes páginas físicas libres, no necesariamente consecutivas, para cargar el programa. Además, algunas páginas virtuales de un programa podrán ser expulsadas a disco para posteriormente ser cargadas (posiblemente, en páginas físicas distintas) si son necesarias de nuevo. Por lo tanto, gracias a la reubicación proporcionada por la memoria virtual, un programa no sólo se puede cargar en cualquier parte de la memoria física sino que, además, puede cambiar de posición durante su ejecución.

Para traducir una dirección virtual a física, la MMU divide la dirección virtual en un *número de página virtual* y en un *desplazamiento dentro de la página*, y sustituye (traduce) el número de página virtual por el correspondiente *número de página física*, como podemos ver en la figura 3. El número de página física constituye la parte alta de la dirección física, mientras que el desplazamiento dentro de la página, que no cambia, constituye la parte baja.

El número de bits del desplazamiento determina el tamaño de la página. Normalmente, el número de páginas virtuales suele ser mayor que el de páginas físicas de cara a tener la visión de un tamaño prácticamente ilimitado de memoria virtual. En este ejemplo, el tamaño de la página es $2^{12} = 4$ KB. El número de páginas físicas permitidas en memoria es 2^{18} , ya que el campo “número de página física” es de 18 bits. Esto significa que la memoria principal de este ejemplo puede tener, como máximo, $2^{18} \times 2^{12} = 2^{30} = 1$ GB, mientras que el

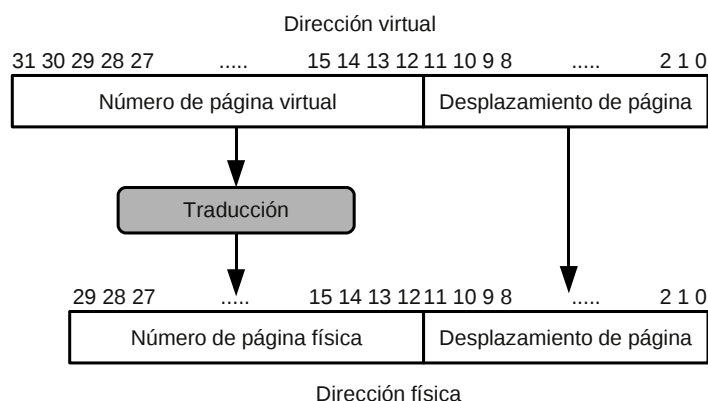


Figura 3: Traducción de una dirección virtual a una dirección física.

espacio de direcciones virtuales es de $2^{20} \times 2^{12} = 2^{32} = 4$ GB, ya que hay un total de 2^{20} página virtuales (el campo “número de página virtual” es de 20 bits). También podemos ver que la dirección virtual es de 32 bits y que la dirección física es de 30 bits. Evidentemente, todas estas cantidades están interrelacionadas, como veremos después.

6.1.2. Consideraciones de diseño de un sistema de memoria virtual

La mayor parte de las decisiones que se deben tomar en el diseño de un sistema de memoria virtual están relacionadas con el gran coste de los fallos de página. Este gran coste es debido principalmente a la gran diferencia de velocidad entre la memoria principal y la secundaria, ya que esta última es más de 10.000 veces más lenta. Algunas decisiones que suelen tomarse son:

- Las páginas deben ser lo bastante grandes para amortizar el tiempo de acceso a disco. Tamaños de 16 KB a 64 KB son normales hoy en día.
- Interesa organizar las páginas en memoria principal de manera que se reduzca al máximo la tasa de fallos. La principal técnica que se usa es permitir un esquema totalmente asociativo en la colocación de páginas virtuales en memoria principal, es decir, permitir que cualquier página virtual se pueda asignar a cualquier página física.
- Los fallos de página se pueden gestionar por software, porque la sobrecarga de usar software en lugar de hardware es poca comparada con el tiempo de acceso a disco. Además, el software puede usar algoritmos más inteligentes para decidir cómo colocar las páginas, porque hasta reducciones pequeñas en la tasa de fallos compensan el coste de estos algoritmos.
- Usar una política de escritura directa para gestionar escrituras en memoria virtual no es apropiada porque cada escritura sería muy lenta. En su lugar, los sistemas de memoria virtual usan postescritura.

6.2. La tabla de páginas

Al usar un esquema totalmente asociativo para la colocación de las páginas en memoria principal, se hace más difícil localizar una página concreta en memoria, ya que puede estar en cualquier lugar. En el sistema de memoria virtual, las páginas se localizan usando una tabla que indexa la memoria. Esta estructura se llama **tabla de páginas** y se encuentra en la memoria principal, con lo que la MMU tendrá que realizar un acceso a memoria principal cada vez que quiera realizar una traducción de direcciones. Una tabla de páginas se indexa con el número de página virtual y contiene el correspondiente número de página física.

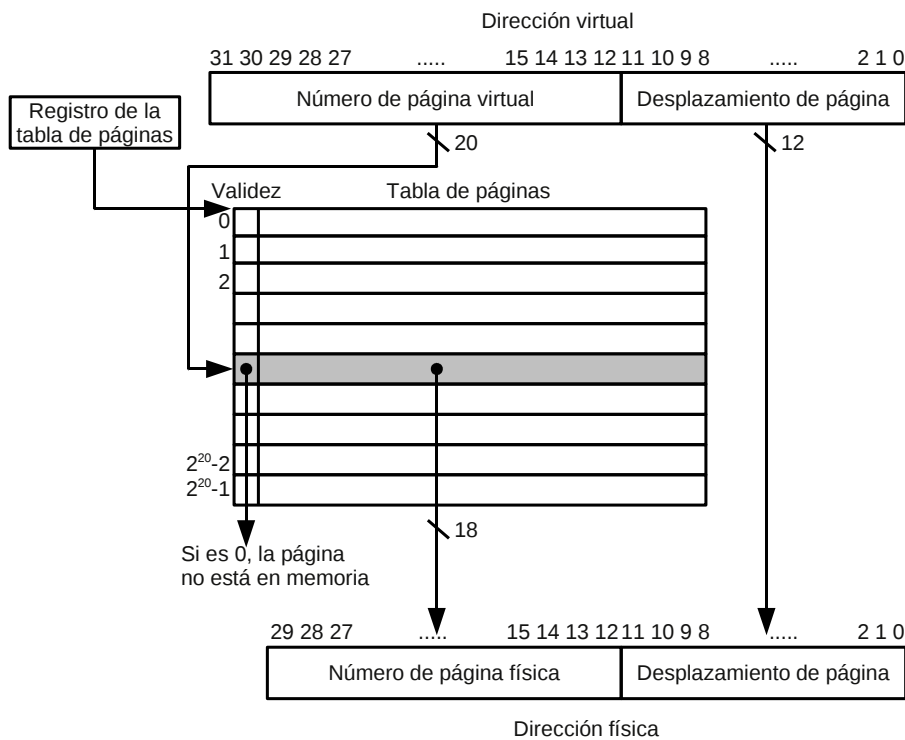


Figura 4: Uso de la tabla de páginas para traducir una dirección virtual a una física.

Cada programa tiene su propia tabla de páginas, que traduce el espacio de direcciones virtuales que el programa maneja a direcciones físicas de la memoria principal asignada a ese programa en un momento dado. Para indicar la ubicación de la tabla de páginas en la memoria, la circuitería incluye un registro que apunta al principio de ésta. Este registro se llama **registro de la tabla de páginas**.

En la figura 4, se puede ver el uso del registro de la tabla de páginas para acceder al inicio de ésta y, tras ello, usar el número de página virtual para acceder a la entrada necesitada, donde se encuentra la información del número de página física correspondiente. Con este número de página física y el desplazamiento dentro de la página (que es el mismo en la página virtual que en la física), se forma la dirección física completa necesaria para el acceso efectivo a la memoria principal. Además, en cada entrada de la tabla de páginas, se usa un bit de validez. Si este bit está apagado indica que la página no está en memoria principal, por lo que un intento de acceder a alguna palabra de esta página producirá un fallo de página.

Debido a que la tabla de páginas contiene una entrada por cada página virtual posible, no se necesitan etiquetas (una entrada no puede pertenecer a dos páginas virtuales distintas), como ocurría en una caché totalmente asociativa.

En el ejemplo de la figura 4, cada entrada sólo necesita tener 19 bits de ancho (18 bits para el número de página física y 1 bit de validez), pero normalmente este tamaño se redondea por exceso a 32 bits para que sea más fácil a la hora de indexar. Los bits extra se suelen usar para guardar información adicional que se necesita para cada página como bits de protección (por ejemplo, para indicar que una página que contiene código es de sólo lectura), direcciones de disco, etc.

6.3. Tratamiento de los fallos de página

Cuando se produce un fallo de página, se produce una **excepción por fallo de página** que provoca que el programa que ha causado el fallo interrumpa su ejecución y el control pase al sistema operativo, que ha de encontrar la página solicitada en la memoria secundaria (es decir, en el disco) y decidir dónde colocarla en la memoria principal.

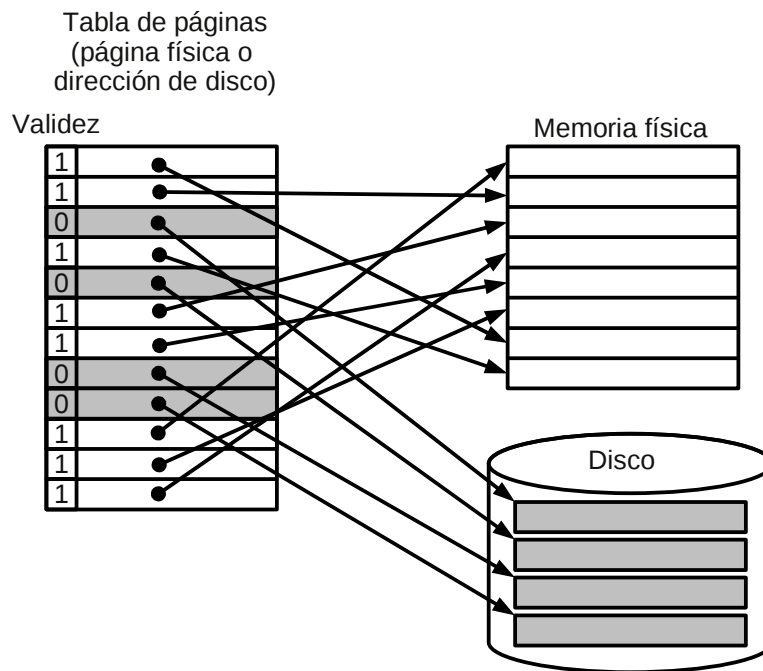


Figura 5: Tabla de páginas unificada: página virtual → página física o dirección en disco.

La dirección virtual, por sí misma, no contiene información de dónde se encuentra la página en la memoria secundaria. Por ello, cuando el sistema operativo inicia la ejecución de un programa, por un lado, crea en memoria secundaria el espacio necesario para almacenar todas las páginas del programa y, por otro lado, crea también una estructura de datos para almacenar el lugar (la dirección) en memoria secundaria donde se guarda cada página virtual. Esta estructura puede ser parte de la tabla de páginas o puede ser una estructura de datos auxiliar indexada de la misma forma que la tabla de páginas. La figura 5 muestra la organización cuando una sola tabla contiene tanto el número de página física como la dirección en memoria secundaria.

Cuando se produce un fallo de página, si todas las páginas físicas de memoria principal están en uso, el sistema operativo ha de escoger una para reemplazar. Ya que se quiere minimizar el número de fallos de página, la mayoría de los sistemas operativos escogen una página que creen que no se necesitará en un futuro breve, usando normalmente el esquema *LRU*.

Un esquema LRU puro es demasiado caro ya que requiere actualizar la estructura de datos usada para el LRU en cada acceso a memoria. En su lugar, la mayoría de los sistemas operativos realizan una aproximación al esquema LRU guardando las páginas que se han referenciado recientemente. Para ayudar al sistema operativo a construir este esquema *pseudo-LRU*, algunas máquinas proporcionan por cada página virtual un *bit de uso* (también llamado *bit de referencia*) que se activa cuando se accede a la página. El sistema operativo apaga periódicamente los bits de referencia de tal manera que, al final de un periodo, las páginas que tengan el bit de uso a 1 serán las que se hayan referenciado en dicho periodo, mientras que las que tengan el bit de uso a 0 serán las que hace más tiempo que no se referencian. A la hora de seleccionar una página para reemplazar, el sistema operativo seleccionará una entre las que tengan el bit de uso desactivado¹.

En un sistema de memoria con memoria virtual, las escrituras en la memoria secundaria tardan millones de ciclos, por lo que una política de escritura directa es completamente impracticable (incluso usando un buffer de escrituras). Debido a esto, los sistemas de memoria virtual usan un esquema de *postescritura*: realizan las escrituras individuales en las páginas de la memoria principal y copian las páginas a memoria secundaria cuando éstas se van reemplazando.

¹Este esquema pseudo-LRU se puede perfeccionar más si en lugar de tener en cuenta el último periodo se tiene en cuenta, para cada página, el valor de su bit de uso en los últimos “n” periodos.

Tabla 1: Valores típicos para un TLB.

Característica	Valor
Tamaño de TLB	32 – 4096 entradas
Tamaño de bloque	1 – 2 entradas de la tabla de páginas
Tiempo de acierto	0,5 – 1 ciclos
Penalización por fallo	10 – 30 ciclos
Tasa de fallos	0,01 % – 1 %

La política de postescritura hace necesario añadir un *bit de modificación* (*dirty bit*) a cada página para conocer si una página ha sido modificada desde que se leyó de la memoria secundaria o no. Este bit se activa cuando se escribe en alguna palabra de la página. Si el sistema operativo decide reemplazar una página con el bit de modificación activo, la página tendrá que ser salvada en memoria secundaria antes de que su lugar sea ocupado por una nueva página.

El bit de modificación puede ser usado por el sistema operativo, junto con el bit de uso, a la hora de seleccionar una página. Entre dos páginas no referencias recientemente, una modificada y otra no, es siempre mejor reemplazar la que no se ha modificado ya que reemplazar la página modificada supone, como hemos dicho, escribirla antes en disco. La página no modificada, sin embargo, puede ser reemplazada sin más por la nueva página, lo que permite resolver más rápidamente el fallo de página.

6.4. TLB (Translation–Lookaside Buffer)

Como las tablas de páginas están en memoria principal, cada petición de lectura/escritura a memoria por parte de un programa supone, al menos, dos accesos a memoria: un primer acceso, que realiza la MMU a la tabla de páginas del programa, para traducir la dirección virtual a dirección física y un segundo acceso para, usando la dirección física obtenida, acceder físicamente a la palabra solicitada.

Para evitar acceder dos veces a la memoria principal en cada petición de lectura/escritura de una palabra, las máquinas actuales incorporan en el chip del procesador, junto a la MMU, una memoria caché especial que guarda las traducciones *número de página virtual* → *número de página física* más frecuentemente usadas. Esta caché especial de la tabla de páginas es conocida como *buffer de traducción adelantada de direcciones* o *TLB* (*Translation–Lookaside Buffer*).

Un TLB es una caché que contiene sólo información de la tablas de páginas, es decir, el TLB es una caché que trata la tabla de páginas como una mini-memoria principal de bloques monopalabra donde cada posición de la tabla de páginas es como una palabra de memoria, con lo que el índice de la tabla de páginas (número de página virtual) viene a ser como la dirección de la palabra y el contenido de la posición de la tabla (básicamente, el número de página física) viene a ser el contenido de la palabra. Esta organización se puede ver en la figura 6. En este caso, el TLB es una memoria caché 2–asociativa de 4 conjuntos, por lo que los 2 bits menos significativos del número de página virtual se utilizan como índice y los restantes 18 bits se utilizan como etiqueta. En la tabla 1 podemos ver algunos valores típicos para un TLB.

El principio básico por el que funciona un TLB es la localidad de las referencias a la tabla de páginas. Esta localidad se cumple ya que, cuando un programa accede a una palabra que pertenece a una página virtual **X**, es muy probable que próximamente acceda a palabras de direcciones cercanas y que, por tanto, pertenezcan a la misma página **X**. Esto haría que se accediera varias veces a la misma entrada de la tabla de páginas para traducir el número de página virtual **X** al número de página física correspondiente.

En un sistema de memoria virtual con TLB, en cada referencia a memoria se sigue el proceso recogido en el siguiente algoritmo:

1. La MMU descompone la dirección virtual en: número de página virtual + desplazamiento.

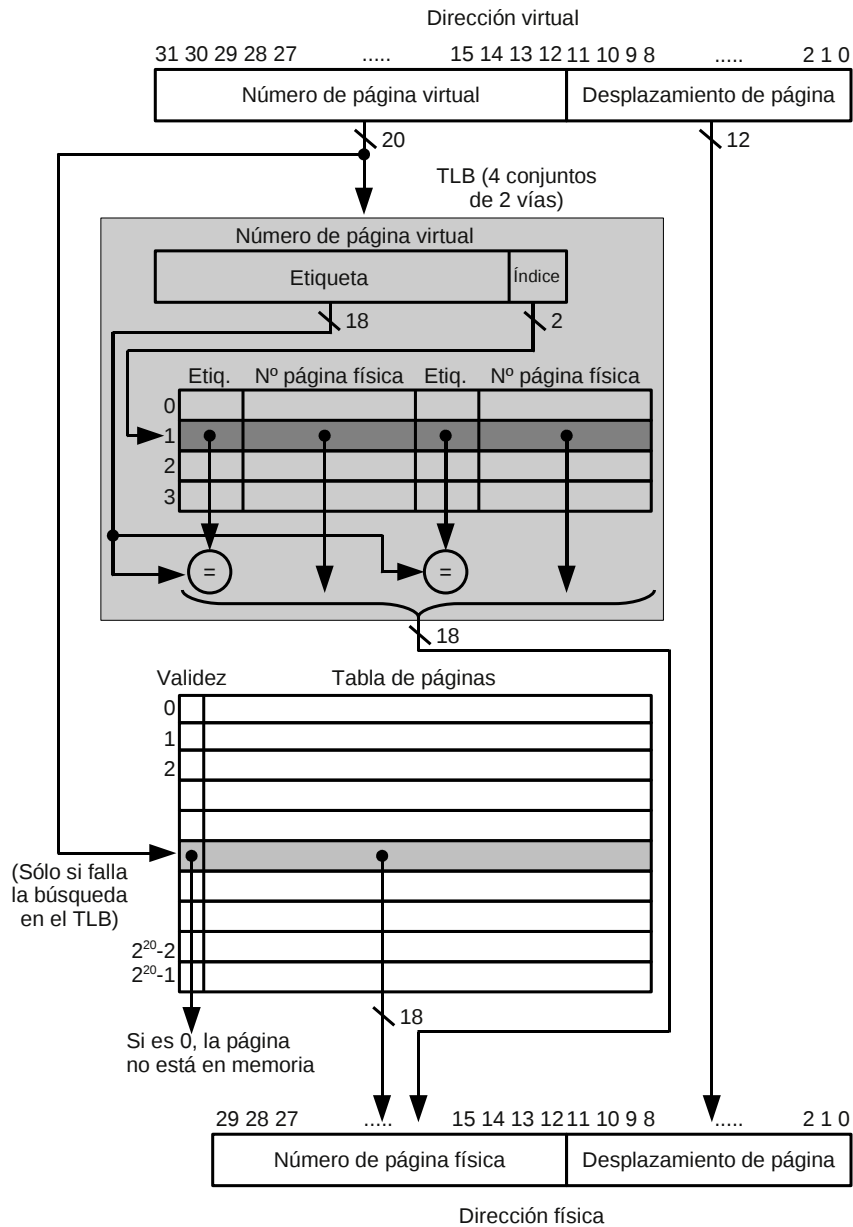


Figura 6: Uso del TLB como una caché de la tabla de páginas.

Tabla 2: Características de los TLBs del Pentium Pro y del PowerPC 604.

Característica	Intel Pentium Pro	IBM PowerPC 604
Direcciones virtuales	32 bits	52 bits
Direcciones físicas	32 bits	32 bits
Tamaño de página	4KB, 4MB	4KB, 256MB
Organización del TLB	Un TLB para instrucciones y otro para datos. Ambos 4-asociativos. Pseudo-LRU. TLB instrucciones: 32 entradas. TLB datos: 64 entradas. Fallos tratados en hardware.	Un TLB para instrucciones y otro para datos. Ambos 2-asociativos. LRU. TLB instrucciones: 128 entradas. TLB datos: 128 entradas. Fallos tratados en hardware.

2. La MMU busca el número de página virtual en el TLB:
 - Si hay *acierto de TLB*, se salta al **paso 6**.
 - Si hay *fallo de TLB*, se sigue por el **paso 3**.
3. Como hay un *fallo de TLB*, entonces la MMU no tendrá más remedio que ir a la tabla de páginas y comprobar el bit de validez de la entrada correspondiente:
 - Si está encendido, hay *acierto de página* y se salta al **paso 5**.
 - Si está apagado, hay *fallo de página* y se sigue por el **paso 4**.
4. Como hay *fallo de página*, el programa que ha provocado el fallo interrumpe su ejecución y el control pasa al sistema operativo, que ha de encontrar la página en la memoria secundaria y decidir dónde colocar la página solicitada en la memoria principal. Además, si es necesario, la página que se sustituye en memoria principal es escrita en memoria secundaria. Tras ello, en la entrada de la tabla de páginas de esa página virtual, se pondrá el número de página física donde ha sido colocada y se encenderá el bit de validez.
5. La MMU copia el contenido de la entrada de la tabla de páginas que corresponde a la página virtual buscada (bit de uso, bit de modificación y número de página física) al TLB en la posición que le corresponda. Además, actualiza la etiqueta de esa posición del TLB para identificar a esta nueva entrada.
6. La MMU forma la dirección física (el número de página virtual se traduce a número de página física) usando la información contenida en el TLB. Además, actualiza los bits de control de esa página (se activa el bit de uso y, si la referencia es una escritura, se activa también el bit de modificación).

Los diseñadores han usado diferentes valores de asociatividad en el TLB. Con un esquema totalmente asociativo, escoger la entrada a reemplazar en el TLB es caro ya que se espera que los fallos de TLB sean más frecuentes que los fallos de páginas (por tener el TLB menos entradas que la tabla de páginas). Como consecuencia, muchos sistemas utilizan TLBs asociativos de unas pocas vías o proporcionan algún soporte para escoger aleatoriamente la entrada del TLB a reemplazar en vez de utilizar un esquema LRU. Por ejemplo, la tabla 2 muestra las características de los TLBs de dos procesadores bastante diferentes: el Pentium Pro y el PowerPC 604. Como podemos ver, ambos permiten páginas de dos tamaños distintos: las de tamaño normal y otras mucho más grandes, que se pueden usar para almacenar código y datos que siempre están en memoria (por ejemplo, del sistema operativo) y así reducir el número de páginas necesarias.

En cuanto a la escritura, los TLB suelen emplear una estrategia de postescritura, ya que se espera que la tasa de fallos del TLB sea pequeña (aunque mayor que la de la tabla de páginas). Esto conlleva que en el paso 6 del algoritmo anterior ocurra que:

- La actualización de los bits de control correspondientes a la página utilizada se realice solamente en el TLB y no en la tabla de páginas.
- Cuando una entrada en el TLB va a ser sustituida, toda su información se debe copiar en la tabla de páginas.

6.5. Implementación de la protección con memoria virtual

Una de las funciones más importantes de la memoria virtual es permitir que varios programas compartan una única memoria principal, a la vez que se proporciona protección entre estos programas y el sistema operativo. El mecanismo de protección tiene que asegurar que, aunque muchos programas estén compartiendo la misma memoria principal, un programa no pueda leer/escribir en la memoria perteneciente a otro programa o al sistema operativo, tanto intencionadamente como por error.

Cada programa tiene su propio espacio de direcciones virtuales con su propia tabla de páginas que hace corresponder cada página virtual con una determinada página física. Por lo tanto, si el sistema operativo mantiene las tablas de páginas de los programas de tal manera que las páginas virtuales de dos programas distintos nunca compartan páginas físicas, un programa nunca podrá acceder a los datos de otro. Evidentemente, es necesario que un programa de usuario no tenga capacidad para modificar por sí mismo los valores de su tabla de páginas, siendo el sistema operativo el encargado de hacerlo.

Cuando el sistema operativo decide cambiar del programa en ejecución P_1 al programa P_2 (esto se llama *cambio de contexto* o *cambio de proceso*) tiene que asegurarse que P_2 no pueda tener acceso a la tabla de páginas de P_1 . Si no hay TLB, es suficiente con cambiar el valor del registro que apunta al inicio de la tabla de páginas del programa en ejecución para que ahora apunte a la tabla de P_2 (y no a la de P_1). Con un TLB, se tiene que vaciar el TLB de las entradas que pertenezcan a P_1 , tanto para proteger los datos de P_1 como para dar espacio a las entradas de P_2 .

Para garantizar esta protección del sistema de memoria virtual por parte del sistema operativo, es necesario que el hardware de la máquina proporcione al menos estas tres características:

1. Posibilidad de que la CPU tenga dos modos de funcionamiento: un *modo usuario*, que indique que el programa en funcionamiento es un programa de usuario (con permisos de escritura/lectura limitados a su espacio de direcciones virtuales), y un *modo supervisor* (también llamado *modo núcleo*), que indique que el código en ejecución es el del sistema operativo.
2. Información sobre el estado en el que se encuentra la CPU, que pueda ser leída por los programas de usuario pero no modificada, y que el sistema operativo puede modificar usando instrucciones especiales que sólo se pueden ejecutar en modo supervisor. Esta información será:
 - El bit que indica si el modo de la CPU es usuario o supervisor.
 - El registro que apunta al inicio de la tabla de páginas.
 - El contenido de la tabla de páginas.
 - El contenido del TLB.
3. Mecanismos que permitan a la CPU pasar del modo usuario al modo supervisor y viceversa. Lo primero ocurre cuando se produce una interrupción, cuando un programa de usuario provoca una excepción o cuando un programa de usuario ejecuta una llamada al sistema, que se basa en una instrucción especial, (*syscall*) que transfiere el control al código del sistema operativo. Lo segundo, es decir, el retorno al modo usuario, se realiza mediante una instrucción de *retorno de excepción (RFE)*, que restaura el estado del programa que estaba en ejecución en el momento de pasar del modo usuario al modo supervisor.

6.6. Un marco común para las jerarquías de memoria

La memoria virtual y los sistemas de caché funcionan juntos como una jerarquía, de manera que un dato no puede estar en la caché si la página a la que pertenece no está presente en la memoria principal. El sistema operativo tiene un papel importante en el mantenimiento de esta jerarquía, ya que, cuando decide enviar una página a memoria secundaria, elimina los contenidos de la página que estén en la caché. Al mismo tiempo, el sistema operativo modifica las tablas de páginas y el TLB, de manera que un intento de acceder a cualquier dato de la página que se ha copiado a la memoria secundaria generará un fallo de página. Las posibles combinaciones de sucesos en el TLB, memoria virtual y caché serían, ordenados de mayor a menor rapidez de acceso, los mostrados en la tabla 3 (excluidos posibles reemplazos de bloques o páginas).

A continuación mostraremos el algoritmo completo de acceso a un dato por parte del procesador en una máquina que cuenta con memoria virtual (MMU con TLB en el chip del procesador y tabla de páginas en la memoria principal) y una memoria caché en el chip del procesador (figura 7). Los pasos del 0 al 6 son los vistos anteriormente en el algoritmo de traducción de direcciones virtuales a físicas (apartado 6.4, *Translation–Lookaside Buffer*).

0. El procesador quiere leer un dato que está en una dirección concreta dentro de su espacio virtual de memoria. Para ello, envía la dirección virtual a la MMU para que la traduzca en la correspondiente dirección física.
1. La MMU descompone la dirección virtual en: número de página virtual + desplazamiento.
2. La MMU busca el número de página virtual en el TLB:
 - Si hay *acierto de TLB*, saltar al **paso 6**.
 - Si hay *fallo de TLB*, seguir por el **paso 3**.
3. Como hay un *fallo de TLB*, la MMU no tendrá más remedio que ir a la tabla de páginas y comprobar el bit de validez de la entrada correspondiente:
 - Si está encendido: hay *acierto de página*; saltar al **paso 5**.
 - Si está apagado: hay *fallo de página*; seguir por el **paso 4**.
4. Como hay *fallo de página*:
 - a) El programa que ha provocado el fallo interrumpe su ejecución, el control pasa al sistema operativo.
 - b) El S.O. ha de encontrar la página en la memoria secundaria.
 - c) El S.O. decide dónde colocar la página solicitada en la memoria principal (algoritmo de sustitución).
 - d) Además, si es necesario, el S.O. escribe en memoria secundaria la página que se sustituye en memoria principal.
 - e) El S.O. escribe la página solicitada en memoria principal.
 - f) El S.O. actualiza la tabla de páginas: en la entrada de la tabla de páginas de esa página virtual se pondrá el número de página física donde ha sido colocada y se encenderá el bit de validez.
5. La MMU:
 - a) Copia el contenido de la entrada de la tabla de páginas que corresponde a la página virtual buscada (bit de uso, bit de modificación y número de página física) al TLB en la posición que le corresponda.

Tabla 3: Posibles combinaciones de sucesos en el TLB, memoria virtual y caché ordenados de mayor a menor rapidez de acceso.

TLB	TP	Caché	¿Posible?	Nº Mínimo accesos MP	Nº Mínimo accesos MS
A	A	A	Lo más deseado. No se comprobaría la tabla de páginas porque hay acierto de TLB. La palabra solicitada sí está en caché.	0	0
A	A	F	Posible. No se comprobaría la tabla de páginas porque hay acierto de TLB. La palabra solicitada no está en caché; habrá que acceder a memoria principal, donde se encuentra la página a la que pertenece.	1	0
F	A	A	Posible. Hay que acceder a la tabla de páginas para obtener la dirección física. La palabra solicitada sí está en caché.	1	0
F	A	F	Posible. Hay que acceder a la tabla de páginas para obtener la dirección física. La palabra solicitada no está en caché; habrá que acceder a memoria principal, donde se encuentra la página a la que pertenece, para obtener la palabra.	2	0
F	F	F	Lo menos deseado. Hay que acceder a la tabla de páginas para obtener la dirección física. La página a la que pertenece la palabra solicitada no está en memoria principal; necesario acceso a memoria secundaria para traerla. Tras ello, como la palabra solicitada no está en caché, habrá que acceder a memoria principal, donde ya se encuentra la página a la que pertenece, para obtener la palabra.	2	1
A	F	X	Imposible. No se puede tener una traducción en el TLB de una página que no está en memoria principal.		
X	F	A	Imposible. No se puede tener un dato en la caché cuya página no está en memoria principal.		

- b) Actualiza la etiqueta de esa posición del TLB para identificar a esta nueva entrada.
- c) Si es necesario, copia a la tabla de páginas los datos de la entrada sustituida en el TLB.

6. La MMU:

- a) Forma la dirección física (el número de página virtual se traduce al correspondiente número de página física) usando la información contenida en el TLB.
- b) Actualiza los bits de control de esa página (se activa el bit de uso y, si la referencia es una escritura, se activa también el bit de modificación).

7. La dirección física llega al hardware de la caché.

8. El hardware de la caché descompone la dirección física en: etiqueta + índice de caché + desplazamiento de palabra + desplazamiento de byte y, a continuación, busca el dato solicitado (los detalles de esta búsqueda dependen de la estructura de la caché):

- Si hay *acierto de caché*, saltar al **paso 10**.
- Si hay *fallo de caché*, seguir por el **paso 9**.

9. Hay que acceder a memoria principal a por el dato

- a) La dirección física llega a la memoria principal.
- b) Se copia el bloque completo al que pertenece el dato solicitado desde memoria principal a la posición apropiada de la caché.
- c) Si es necesario, el bloque sustituido en caché se escribe en memoria principal.

10. El hardware de la caché:

- a) Proporciona al procesador el dato solicitado.
- b) Actualiza los bits de control apropiados de ese bloque en caché.

En resumen, a partir de una dirección virtual generada por la CPU, hay que construir la dirección física con lo que, en primer lugar, se accederá al TLB. Si hay acierto de TLB, éste proporcionará la dirección física. Si hay fallo de TLB, la MMU tendrá que acceder a la tabla de página, indexando esta tabla con el número de página virtual. Si se produce un fallo de página, hasta que éste no sea resuelto por el sistema operativo, no se podrá realizar la traducción de dirección virtual a física. Finalmente, con la dirección física, se realizará el acceso a caché, que puede producir, a su vez, un acceso a la memoria principal si se produce un fallo de caché. En la figura 8 se muestra el caso más deseable, es decir, acierto de TLB y acierto de caché, con lo que se puede acceder rápidamente a la palabra deseada.

6.7. Jerarquía de memoria para la DECSTATION 3100

La DECStation 3100 era una estación de trabajo que utilizaba un procesador MIPS R2000 con una estructura sencilla de memoria caché y memoria virtual. A continuación, describimos cada tipo de memoria.

6.7.1. Memoria caché

Cuenta con dos cachés diferentes, una para instrucciones y otra para datos. Cada caché es de 64 KB (16 Kpalabras de 32 bits cada una) con bloques monopalabra (figura 9).

Las peticiones de lectura a caché son triviales. Como hay cachés separadas para instrucciones y datos, se necesitan señales de control separadas para leer y escribir en cada caché. Los pasos para una petición de lectura a cualquiera de las cachés son los siguientes:

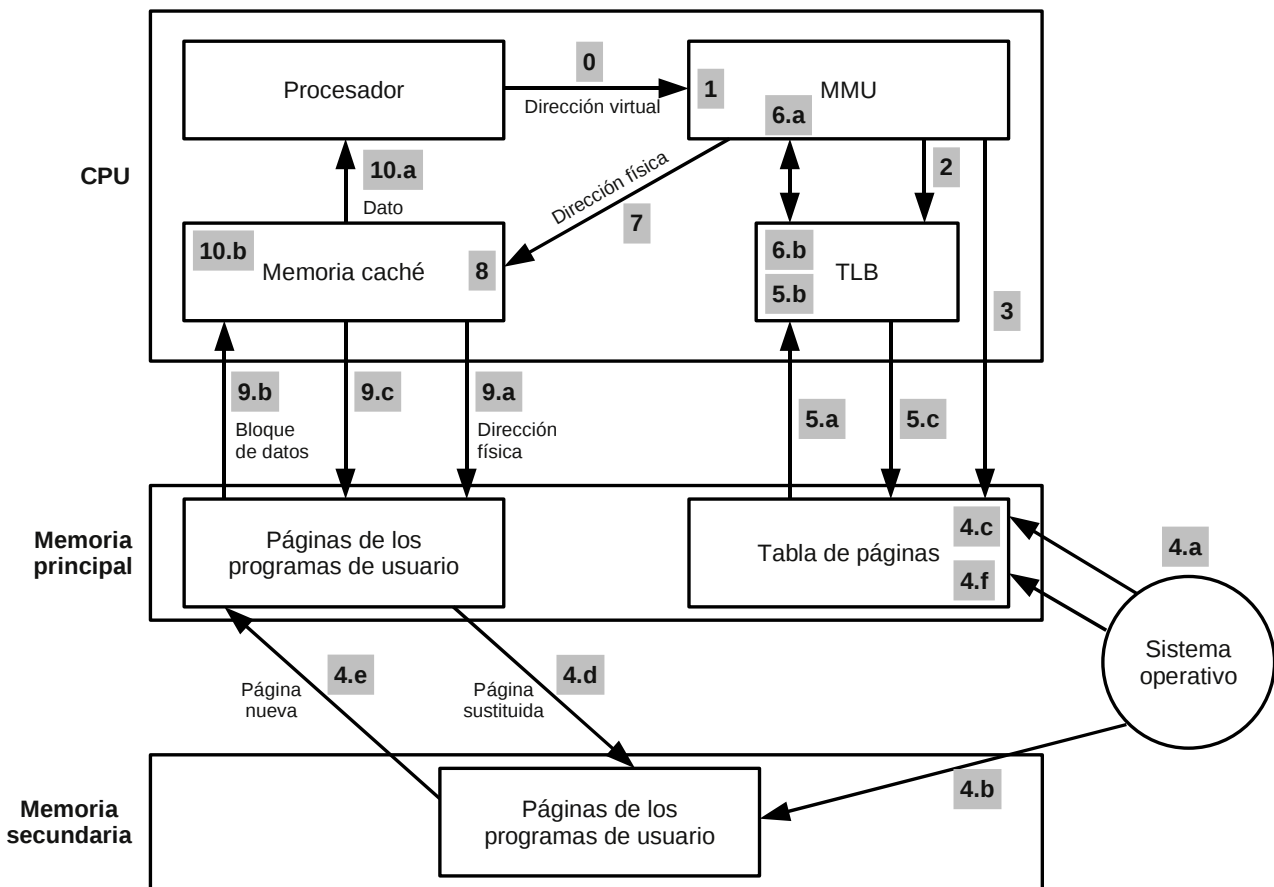


Figura 7: Esquema de acceso a un dato a través de la jerarquía completa de memoria.

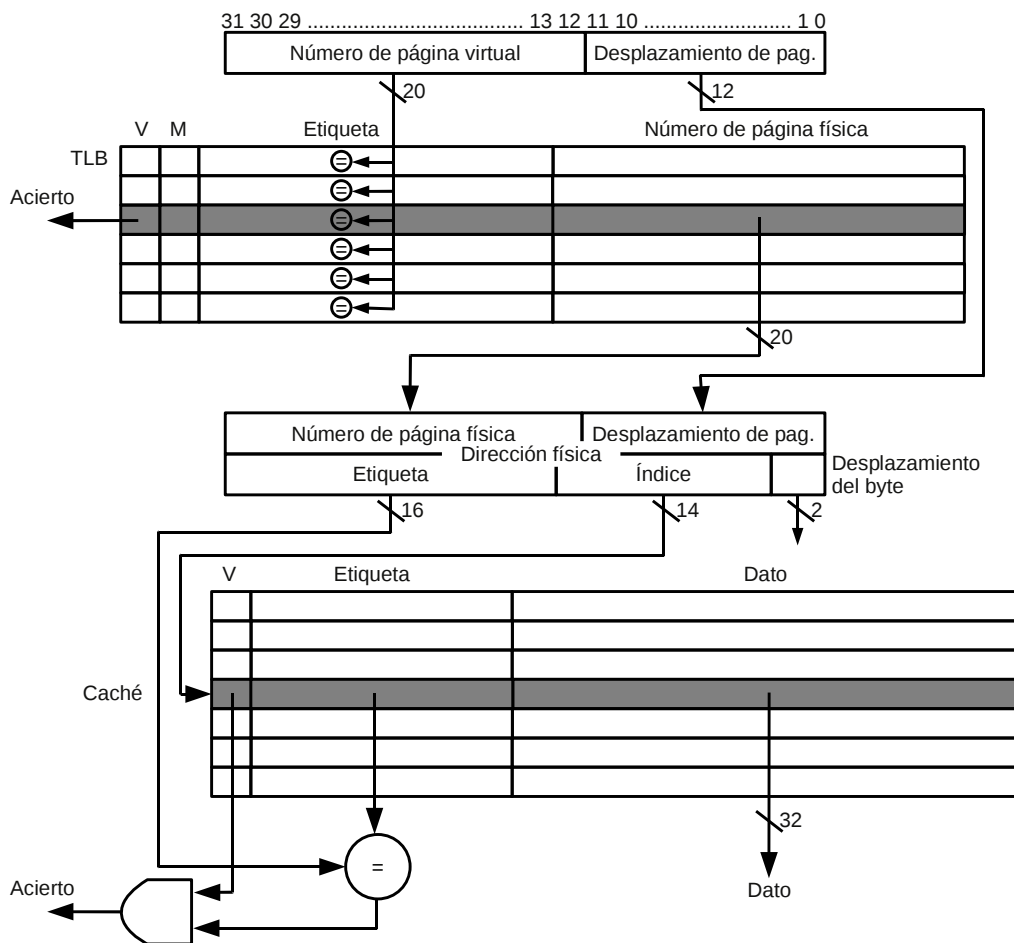


Figura 8: Uso del TLB y la caché para la lectura de un dato solicitado por la CPU.

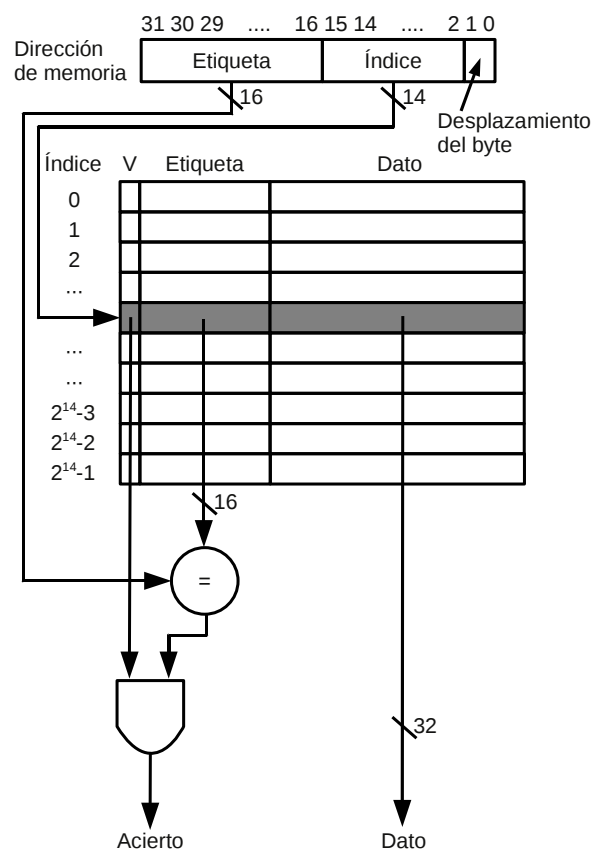


Figura 9: Estructura de una de las dos cachés de la DECStation 3100.

1. Enviar la dirección a la caché apropiada. La dirección proviene tanto del PC (para leer instrucciones) como de la ALU (para accesos a datos).
2. Indexar la caché con los bits del 15 al 2 de la dirección.
3. Comparar los bits del 31 al 16 con la etiqueta.
4. Si el acceso a caché es un acierto, la palabra pedida está accesible en las líneas de datos. Si el acceso a la caché es un fallo, se envía la dirección a la memoria principal. Cuando la memoria envía los datos, éstos se escriben en la caché.

Las escrituras funcionan usando una política de escritura directa utilizando un buffer de escrituras que almacena los datos mientras están esperando para ser escritos en memoria. Cuando hay un fallo de escritura, al tener bloques monopalabra, no hay razón para leer nada de memoria principal. El esquema para procesar escrituras sería:

1. Enviar la dirección a la caché apropiada.
2. Indexar la caché con los bits del 15 al 2 de la dirección.
3. Escribir los bits 31 al 16 en la etiqueta, escribir la palabra de datos en el campo de datos y actualizar el bit de validez.
4. Escribir también la palabra en memoria (en el buffer de escrituras realmente) usando la dirección entera.

6.7.2. Memoria virtual

El sistema de memoria virtual de esta máquina tiene un espacio de direcciones virtuales de 32 bits; como usa páginas de 4KB, el campo de la dirección que indica el desplazamiento dentro de la página es de 12 bits y, por tanto, el campo que indica el número de página virtual es de 20 bits. La dirección física es del mismo tamaño que la dirección virtual.

El TLB contiene 64 entradas, es totalmente asociativo y es compartido entre las referencias a instrucciones y a datos. Cada entrada consta de 64 bits y contiene 20 bits de etiqueta (que, al ser totalmente asociativo, coincide con el número de página virtual de esa entrada en el TLB), el correspondiente número de página física, un bit de validez y otros bits de control.

Boletines de prácticas

Normas sobre la entrega de prácticas

Será necesario seguir las siguientes reglas para entregar las prácticas de todos los boletines, además de cualquier otra regla específica que se indique en un boletín particular:

- Las prácticas se entregarán únicamente mediante la opción de contenidos del alumno en SUMA.
- Se entregará un único archivo comprimido en formato *.tar.gz* o *.zip* que contendrá la memoria en formato PDF, los circuitos y programas que se hayan generado (código fuente) y cualquier otro fichero que se considere oportuno. El nombre del archivo será *prácticas-DNI-BOLETIN.FORMATO* (por ejemplo: *prácticas-12345678-B2.3.tar.gz*).
- La memoria incluirá, al menos, la siguiente información en un solo documento PDF:
 - Nombre y DNI del autor o autores de la práctica.
 - Descripción de los ficheros y directorios contenidos en el archivo entregado.
 - Contestación a las preguntas planteadas en los boletines. La respuesta a cada pregunta debe ser independiente, y debe estar claramente identificada.
 - Explicación de las pruebas realizadas para comprobar la corrección de la práctica entregada e instrucciones para su reproducción. Cuando sea posible, se incluirán los ficheros utilizados en dichas pruebas.
 - Explicación del trabajo realizado y cualquier aclaración que el alumno considere pertinente.
 - Lista de bibliografía y otras fuentes de información consultadas.

No se corregirá ninguna práctica que no se ciña estrictamente a los formatos especificados anteriormente.

B6.1. Funcionamiento de la memoria virtual en Linux

B6.1.1. Objetivos

El objetivo de esta sesión es que el alumno vea, de una forma práctica, el funcionamiento de la memoria virtual en Linux. Para ello, se describirá el funcionamiento de varias órdenes que proporcionan información sobre el uso de la memoria, como `top`, `free`, `ps` y `vmstat`, y se mostrará, mediante pequeños programas en C, cómo los programas hacen uso de la memoria virtual.

B6.1.2. Prerequisitos

- Lectura de los apuntes de teoría.

B6.1.3. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura por parte del alumno de las secciones **B6.1.4** y **B6.1.5**.
2. Realización, de forma individual, de los ejercicios propuestos en el boletín (con supervisión del profesor).

```

USER      PID  %CPU  %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0   0.0   4076   636 ?        Ss   Feb26   0:10 /sbin/init
root         2   0.0   0.0     0     0 ?        S<   Feb26   0:00 [kthreadd]
root         3   0.0   0.0     0     0 ?        S<   Feb26   0:00 [migration/0]
root         4   0.0   0.0     0     0 ?        S<   Feb26   0:02 [ksoftirqd/0]
root         5   0.0   0.0     0     0 ?        S<   Feb26   0:00 [watchdog/0]
root         6   0.0   0.0     0     0 ?        S<   Feb26   0:00 [migration/1]
root         7   0.0   0.0     0     0 ?        S<   Feb26   0:09 [ksoftirqd/1]
root         8   0.0   0.0     0     0 ?        S<   Feb26   0:00 [watchdog/1]
root         9   0.0   0.0     0     0 ?        S<   Feb26   0:01 [migration/2]
root        10   0.0   0.0     0     0 ?        S<   Feb26   0:04 [ksoftirqd/2]
root        11   0.0   0.0     0     0 ?        S<   Feb26   0:00 [watchdog/2]
root        12   0.0   0.0     0     0 ?        S<   Feb26   0:01 [migration/3]
root        13   0.0   0.0     0     0 ?        S<   Feb26   0:04 [ksoftirqd/3]
root        14   0.0   0.0     0     0 ?        S<   Feb26   0:00 [watchdog/3]
root        15   0.0   0.0     0     0 ?        S<   Feb26   0:00 [migration/4]
.....
gdm        19225  0.0   0.0  120024  1512 ?        S    Apr13   0:00 /usr/libexec/gvfsd
gdm        19226  0.0   0.1  518664  6792 ?        Sl   Apr13   0:01 metacity
gdm        19227  0.0   0.1  278964  6468 ?        S    Apr13   0:05 gnome-power-manager
gdm        19228  0.0   0.2  384084  10560 ?       S    Apr13   0:17 /usr/libexec/gdm-simple-greeter
root       22878  0.0   0.0   84216   864 ?        S    Mar12   2:29 /usr/sbin/kerneloops --nodaemon
root       23476  0.0   0.0   84220  1108 ?        S    Mar15   2:18 /usr/sbin/kerneloops --nodaemon
root       25002  0.0   0.0   75524  1692 ?        Ss   Apr13   0:05 sendmail: accepting connections
gdm        27157  0.0   0.0   94684  2112 ?        S    Apr13   0:00 /usr/libexec/pulse/gconf-helper
root       27198  0.0   0.0   84216   868 ?        S    Mar09   2:39 /usr/sbin/kerneloops --nodaemon
root       27332  0.0   0.0   3900   408 tty2    Ss+  Apr13   0:00 /sbin/mingetty tty2
root       29813  0.0   0.0   84216   864 ?        S    Mar08   2:51 /usr/sbin/kerneloops --nodaemon

```

Figura B6.1: Ejemplo de salida obtenida al ejecutar `ps aux`.

B6.1.4. Órdenes para mostrar el uso de memoria

En Linux, existen varias órdenes que nos permiten obtener información sobre el uso de la memoria por parte de los procesos¹. Entre las órdenes más comunes y útiles están las que se describen a continuación.

`ps` muestra los procesos en ejecución, uno por línea. Sin opciones, esta orden nos da los procesos asociados a nuestra sesión indicando, para cada uno, su PID, terminal asociada, cantidad de CPU consumida hasta ese momento y orden correspondiente. Como vemos, no se muestra ninguna información sobre el uso de memoria; para ello, debemos ejecutar `ps` con la opción `aux` que, además, nos muestra todos los procesos en ejecución de todos los usuarios. La salida de esta orden es similar a la que aparece en la figura B6.1.

Hay tres columnas que nos informan sobre el uso de la memoria: `%MEM`, `VSZ` y `RSS`. Para cada proceso, `%MEM` muestra el porcentaje de memoria RAM ocupada por el proceso, `VSZ` indica la cantidad de memoria total (virtual o no), en KB, que el proceso usa y `RSS` es la cantidad de memoria RAM, en KB, ocupada por el proceso (por lo tanto, si dividimos `RSS` por la cantidad de memoria RAM del sistema y multiplicamos por 100, obtendremos el valor mostrado en la columna `%MEM`). Observa que hay «procesos» que, aparentemente, no consumen memoria (son aquellos cuya orden aparece entre corchetes). Estos procesos se tratan de manera especial ya que son en realidad «hilos» del sistema operativo y la memoria que usan es la memoria RAM ocupada por el núcleo del propio sistema operativo.

Es importante comprender y distinguir los valores mostrados en las columnas `VSZ` y `RSS`. Recordemos que, gracias a la memoria virtual, un proceso no tiene por qué estar completamente en memoria; hay código y datos que deben estar en la memoria principal para que el proceso se pueda ejecutar, mientras que otras partes del mismo pueden estar almacenadas en el disco duro. Por tanto, `VSZ` muestra la cantidad total de memoria que un proceso ocupa, tanto en disco como en memoria principal, mientras que `RSS` muestra sólo la cantidad de memoria principal usada por el proceso.

¹Como ya se explicó en Fundamentos de Computadores, un proceso es un programa en “ejecución”; un proceso es, por tanto, un elemento activo que se crea al ejecutar el programa correspondiente, evoluciona durante la ejecución del programa y se destruye cuando finaliza la ejecución; un programa, en cambio, es algo estático, almacenado en disco en uno o varios ficheros.

```

top - 13:27:04 up 54 days, 15:09, 13 users,  load average: 1.35, 0.44, 0.15
Tasks: 328 total,  1 running, 325 sleeping,   0 stopped,   2 zombie
Cpu(s):  0.7%us,  0.1%sy,  0.0%ni, 99.2%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4017624k total, 3700604k used,   317020k free,   348192k buffers
Swap:  6078456k total,  43580k used,  6034876k free,  1780916k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 4937 piernas   20   0  339m 293m 5660 S   3.6   7.5   5:24.32 nxagent
12061 piernas   20   0 1156m 217m  26m S   1.2   5.5   4:00.88 firefox
 5138 piernas   20   0  592m  25m  17m S   0.8   0.6   0:20.77 kwin
 5142 piernas   20   0  919m  57m  29m S   0.4   1.5   4:26.80 plasma-desktop
 5145 piernas   20   0  9064 1184  720 S   0.4   0.0   2:09.56 ksysguardd
 5625 piernas   20   0  459m  29m  15m S   0.4   0.7   0:43.34 konsole
15323 piernas   20   0 14992 1324  856 R   0.4   0.0   0:00.02 top
     1 root      20   0  4076  636  528 S   0.0   0.0   0:10.94 init
     2 root      15  -5     0     0     0 S   0.0   0.0   0:00.00 kthreadd
.....

```

Figura B6.2: Ejemplo de salida obtenida al ejecutar `top`.

```

procs -----memory-----  ---swap---  -----io-----  --system--  -----cpu-----
 r  b  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id  wa  st
 0  0  43580 285392 348288 1780916  0  0  2  12  0  0  1  0  98  0  0
 0  0  43580 285260 348288 1780916  0  0  0  0 305 685  0  0 100  0  0
 1  1  43580 285136 348292 1780912  0  0  0  40 344 743  0  0  99  0  0
 0  0  43580 285012 348292 1780916  0  0  0  4 575 955  2  0  98  0  0
 0  0  43580 284888 348292 1780916  0  0  0  0 350 741  0  0 100  0  0
 0  0  43580 284764 348292 1780916  0  0  0 24 301 681  0  0 100  0  0
 0  0  43580 284516 348292 1780916  0  0  0  0 349 733  0  0  99  0  0
^C

```

Figura B6.3: Ejemplo de salida obtenida al ejecutar `vmstat 1`.

Una orden que muestra información similar a la mostrada por `ps` es `top` (ver figura B6.2). De hecho, las columnas `%MEM`, `VSZ` y `RSS` que muestra una orden `ps aux`, aparecen en `top` como `%MEM`, `VIRT` y `RES`, respectivamente. Una ventaja de `top` respecto a `ps` es que `top` actualiza periódicamente la información que aparece por pantalla (mientras no salgamos del programa pulsando «q»). Otra ventaja es que permite ordenar los procesos según un determinado criterio. Por ejemplo, si pulsamos «M», `top` ordenará los procesos según la columna `%MEM`, es decir, según el porcentaje de memoria RAM ocupada, mientras que si pulsamos «P» los ordenará por consumo de CPU (columna `%CPU`). Una desventaja de `top`, sin embargo, es que no muestra información para todos los procesos en ejecución en el sistema (ya que, por lo general, hay más procesos que líneas se pueden ver en pantalla).

La orden `top` nos puede ser muy útil para saber si hay algún proceso que puede estar afectando al rendimiento del sistema, haciendo que éste vaya anormalmente lento. Los posibles culpables serán aquellos procesos que consuman mucha CPU o mucha memoria.

Otra orden interesante es `vmstat`. Esta orden nos proporciona información estadística diversa y, entre otras cosas, muestra, bajo la columna `swap`, el número de páginas que se han leído de disco (columna `si`) y el números de página que se han expulsado a disco (columna `so`). Esta orden se puede ejecutar con el parámetro `1` (uno) que muestra, cada segundo y de forma ininterrumpida (hasta que se pulsa «Ctrl+C»), la información estadística recogida en el último segundo. En la figura B6.3 podemos ver un ejemplo de ejecución de esta orden con dicho parámetro.

La última orden que vamos a describir para obtener información sobre el uso de la memoria es `free`. Un ejemplo de salida de esta orden se puede ver en la figura B6.4. Como nos muestra la figura, hay dos líneas claramente diferenciadas. Por un lado, la línea `Mem` nos da la cantidad total de memoria RAM disponible en el

	total	used	free	shared	buffers	cached
Mem:	4017624	3664892	352732	0	349384	1797716
-/+ buffers/cache:		1517792	2499832			
Swap:	6078456	43580	6034876			

Figura B6.4: Ejemplo de salida obtenida al ejecutar `free`.

sistema, cuánta de esa memoria está en uso y cuánta está libre. Por su parte, la línea `Swap` nos muestra cuánta memoria de intercambio² hay disponible en el sistema, cuánta está en uso y cuánta está libre.

Generalmente, tras varias horas o días en funcionamiento, la cantidad de memoria RAM libre en un sistema será muy pequeña. Esto, al contrario de lo que se podría pensar, no significa que el sistema se haya quedado sin memoria. En la mayoría de los casos, significará que gran parte de la memoria se está utilizando para almacenar datos que realmente ningún proceso necesita, pero que se mantienen en memoria por si muy pronto los necesitara algún proceso (al no tener que leer de disco, el acceso a dichos datos será mucho más rápido). Esta memoria se puede considerar a todos los efectos como memoria libre y aparece bajo las columnas `buffers` y `cached`.

En el caso de la memoria de intercambio, al contrario de lo que pasa con la memoria principal, lo normal es que la cantidad de memoria usada sea 0 o muy pequeña. Si la cantidad de memoria de intercambio usada es mucha, puede significar que un proceso tiene un tamaño excesivamente grande o que el número de procesos en ejecución es demasiado elevado, lo que puede degradar el rendimiento del sistema. En estos casos, podemos hacer uso de las órdenes anteriores para intentar averiguar la causa del problema y así solucionarlo.

Finalmente, recordemos que, una vez que hemos identificado al proceso que está consumiendo una cantidad anormalmente grande de memoria (esto también es válido para los procesos que están consumiendo una cantidad anormalmente grande de CPU), podemos finalizar su ejecución con las órdenes `kill -9 <PID>` y `killall -9 <orden>` (siendo `PID` el identificador del proceso y `orden` el nombre del programa que originó el proceso).

B6.1.5. Uso de la memoria en programas

En los sistemas operativos de propósito general actuales, la forma en la que un programa usa su espacio de memoria virtual suele ser bastante compleja ya que, al uso de la memoria que hace el propio programa, hay que sumar el uso que hacen las bibliotecas estáticas y dinámicas que enlazan con el programa. Aunque el estudio de la organización del espacio de memoria virtual de un programa queda fuera de esta asignatura, sí vamos a comentar brevemente el uso de la memoria dinámica, ya que tiene un impacto más inmediato sobre la memoria virtual.

Generalmente, cuando un programa debe procesar una gran cantidad de datos, utiliza memoria dinámica para adaptar su consumo de memoria al tamaño de los datos de entrada. En C, las funciones que se usan para reservar este tipo de memoria son `malloc`, `calloc` y `realloc`, siendo `free` la función empleada para liberar la memoria reservada por cualquiera de las otras tres funciones. El programa de la figura B6.5 muestra el uso de `malloc` y `free` para reservar y liberar, respectivamente, tantos megabytes de memoria como se pasan como parámetro al programa.

Lo siguiente muestra cómo se puede compilar y ejecutar el programa (el programa finaliza cuando se pulsa «Ctrl+C», que muestra por pantalla la cadena `^C`):

```
$ gcc -Wall -o memalloc memalloc.c
$ ./memalloc 4096
^C
$
```

²Recordemos que la memoria de intercambio es el espacio en disco usado por el sistema de memoria virtual.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <string.h>
5
6 #define MEGA 1048576
7
8 int main(int argc, char *argv[])
9 {
10     long megas;
11     char * endptr, * buffer;
12
13     if (argc != 2)
14     {
15         fprintf(stderr, "Uso: %s <cantidad de MB>\n", argv[0]);
16         return 1;
17     }
18
19     megas = strtol(argv[1], &endptr, 10);
20     if ((endptr != NULL && *endptr != '\0') || megas < 0)
21     {
22         fprintf(stderr, "Cantidad de memoria inválida: %s\n", argv[1]);
23         return 2;
24     }
25
26     buffer = (char *) malloc(megas * MEGA);
27     if (buffer == NULL)
28     {
29         fprintf(stderr, "Fallo al reservar memoria\n");
30         return 3;
31     }
32
33     printf("Reserva realizada. Pulse Ctrl+C para finalizar.\n");
34
35     pause();
36
37     free(buffer);
38
39     return 0;
40 }
```

Figura B6.5: memalloc.c

En este ejemplo, el programa se ejecuta con el parámetro 4096 que hace que el proceso correspondiente reserve y libere 4096 MB (o 4 GB) de memoria, que es la cantidad de memoria RAM total en el sistema usado (evidentemente, a la hora de ejecutar el programa, debemos adaptar este valor a la cantidad de memoria RAM total en nuestro sistema).

Lo curioso es que, si ejecutamos la orden `free` antes y después de ejecutar el programa, veremos que el consumo de memoria apenas varía y que la cantidades de memoria de intercambio usada y libre son la misma. Este resultado, un tanto sorprendente, se produce porque el sistema distingue entre «reservar» memoria y «usar» la memoria reservada. Dicho de otra manera, si un programa reserva una gran cantidad de memoria pero no la usa, el sistema no le asignará realmente memoria física.

B6.1.6. Ejercicios

Para cada ejercicio, el portafolios deberá incluir una explicación de cómo se ha resuelto y, en caso de que se realice o modifique algún programa, el código realizado y al menos un ejemplo de su ejecución.

1. Ejecuta varios programa que consuman gran cantidad de memoria (por ejemplo, Firefox, Evolution, OpenOffice, etc.) y muestra, de forma ordenada y de mayor a menor, el consumo de memoria RAM de cada uno de ellos. ¿Cuál es el programa que consume una mayor cantidad de memoria virtual?
2. Modifica el programa de la figura B6.5 para que la reserva de memoria se haga con `calloc` (`calloc`, al igual que `malloc` reserva memoria dinámica pero, además, inicializa dicha memoria a 0; puedes consultar la página de manual de esta función ejecutando `man calloc` para ver cómo se invoca). Ejecuta el programa resultante para que reserve tanta memoria como memoria RAM haya en tu sistema. ¿Se modifica la cantidad de memoria de intercambio usada? ¿Por qué?
3. El programa de la figura B6.6 no sólo reserva la cantidad (en MB) de memoria que se le pasa como parámetro, sino que, además, inicializa los bytes de la memoria reservada con el código ASCII de la letra «x». Ejecuta este programa pasándole como parámetro la cantidad de memoria RAM total en tu sistema. ¿Se modifica la cantidad de memoria de intercambio usada? ¿Por qué?

En teoría, el programa más grande que se puede ejecutar en un sistema con memoria virtual es aquel que usa tanta memoria como memoria RAM y memoria de intercambio juntas haya en el sistema. Ejecuta de nuevo el programa pasándole ahora, como parámetro, la suma de las cantidades de memoria de ambos tipos que haya en tu sistema. ¿Qué ocurre ahora? ¿Y si como parámetro pasamos la suma menos, digamos, 200MB?

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <string.h>
5
6 #define MEGA 1048576
7
8 int main(int argc, char *argv[])
9 {
10     long megas, i;
11     char * endptr, * buffer;
12
13     if (argc != 2)
14     {
15         fprintf(stderr, "Uso: %s <cantidad de MB>\n", argv[0]);
16         return 1;
17     }
18
19     megas = strtol(argv[1], &endptr, 10);
20     if ((endptr != NULL && *endptr != '\0') || megas < 0)
21     {
22         fprintf(stderr, "Cantidad de memoria inválida: %s\n", argv[1]);
23         return 2;
24     }
25
26     for (i = 0; i < megas; i++)
27     {
28         buffer = (char *) malloc(MEGA);
29         if (buffer == NULL)
30         {
31             fprintf(stderr, "Fallo al reservar memoria. Se han podido reservar %ld MB de memoria.\n",
32                 i);
33             return 3;
34         }
35         memset(buffer, 'x', MEGA);
36
37         printf("Reserva realizada. Pulse Ctrl+C para finalizar.\n");
38
39         pause();
40
41         return 0;
42     }
```

Figura B6.6: usamem.c

Ejercicios

E6.1. Jerarquía de memoria — Memoria virtual

1. La máquina M posee un sistema de memoria virtual con las siguientes características:

- Tabla de páginas de 512 entradas.
- Memoria física de 64 KB.
- Tamaño de página virtual de 256 bytes.
- TLB asociativo con 16 conjuntos de 2 vías (2 bloques por conjunto). El TLB sigue una estrategia de reemplazo aleatoria y una política de postescritura.

Se pide:

- a) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios (justificando su inclusión). Para cada entrada de la tabla de páginas, su bit de validez es 1 si el número de página virtual $\text{DIV } 8$ es 0. Además, el número de página física almacenado es $8 - \text{número de página virtual}$.
- b) Dibujar un esquema detallado del TLB y calcular su tamaño incluyendo todos los bits de control necesarios (justificando su inclusión).
- c) Mostrar de forma exacta cómo evoluciona el contenido del TLB y la tabla de páginas después de cada una de las siguientes referencias a bytes (direcciones virtuales) generadas por el procesador: L25, E904 y E75 (L = LECTURA, E = ESCRITURA), determinando en cada referencia a memoria si se produce un acierto/fallo de TLB y/o un acierto/fallo de página y obtener la dirección física resultante.
- d) La máquina M, posee una memoria caché de datos de correspondencia directa con bloques de 4 palabras y 64 bytes de datos. La caché sigue una política de postescritura. Dibuja un esquema detallado de la memoria caché y calcular su tamaño incluyendo los bits de control necesarios (justificando su inclusión). Las palabras son de 16 bits.
- e) Mostrar de forma exacta cómo evoluciona el contenido del TLB, la tabla de páginas y la memoria caché después de cada una de las siguientes referencias a bytes (direcciones virtuales) generadas por el procesador: L25, E904, E75 (L = LECTURA, E = ESCRITURA), determinando en cada referencia a memoria si se produce un acierto/fallo de TLB y/o un acierto/fallo de página y/o un acierto/fallo de caché.

2. La máquina M posee un sistema de memoria virtual con las siguientes características:

- Tabla de páginas de 512 entradas.
- Memoria física de 1024 bytes.
- Tamaño de página virtual de 32 bytes.
- TLB asociativo por conjuntos de 2 vías con una estrategia de reemplazo aleatoria y una política de postescritura.
- El TLB posee 4 conjuntos.

Se pide:

- a) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios (justificando la inclusión de cada uno de ellos).
- b) Dibujar un esquema detallado del TLB y calcular su tamaño incluyendo todos los bits de control necesarios.
- c) En este caso particular, la tabla de páginas parte de un estado inicial en el que para cada entrada de la tabla de páginas, su bit de validez es 1 si el número de página virtual DIV 8 es 0. Además, el número de página física almacenado es $8 - \text{número de página virtual}$. Mostrar de forma exacta el contenido de la tabla de páginas después de cada una de las siguientes referencias a direcciones virtuales generadas por el procesador (referencias a byte): L69, E71, L200 (L = LECTURA, E = ESCRITURA). Determinar, en cada caso, la dirección física resultante. En este apartado no debe tenerse en cuenta el TLB.
- d) Mostrar de forma exacta el contenido del TLB para la misma secuencia de referencias del apartado anterior. Determinar para cada una de ellas si se produce un acierto/fallo de TLB o un acierto/fallo de página. Suponed que el TLB está inicialmente vacío.

Nota: suponed que los bits de control de la tabla de páginas toman inicialmente el valor cero.

3. La máquina M posee un sistema de memoria virtual con las siguientes características:

- Tabla de páginas de 1024 entradas.
- Memoria física de 1024 bytes.
- Tamaño de página virtual de 8 bytes.
- El proceso P genera la siguiente secuencia de referencias a direcciones virtuales (referencias a byte): E15, L2, E20, L26, L38, E39, L18, L47.
- Al proceso P se le asignan como máximo cuatro páginas físicas (1, 3, 5 y 7) que se encuentran inicialmente libres.

Se pide

- a) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios y justificando la inclusión de cada uno de ellos.
- b) Para cada referencia generada por el proceso P, determinar si ocasiona un acierto o un fallo, calcular la dirección física resultante y mostrar cómo quedaría la tabla de páginas en cada caso.
- c) Calcular la tasa de fallos de página. Si la lectura/escritura de una página de/en disco consume 5ms, determinar la penalización por fallos total debida a la ejecución del proceso P.

Nota: la tabla de páginas está inicialmente vacía y todos sus bits de control toman el valor cero.

4. La máquina M posee un sistema de memoria virtual con las siguientes características:

- Tabla de páginas de 8192 entradas.
- Memoria física de 256 bytes.
- Tamaño de página virtual de 8 bytes.
- TLB asociativo por conjuntos de 2 vías con una estrategia de reemplazo LRU y una política de postescritura.
- El TLB posee 2 conjuntos.

```

##### SEGMENTO DE DATOS #####
        .data
        ...
resultados: .space    1000
        ...
##### SEGMENTO DE TEXTO #####
        .text
        ...
        li        $s0, 4
        li        $s1, 64
        la        $s2, resultados
ini_1:    xor        $t0, $t0, $t0
ini_2:    add        $t1, $s1, $t0
        add        $t1, $t1, $s2
        sb        $s0, 0($t1)
        addi       $t0, $t0, 1
        slti       $at, $t0, 2
        bne        $at, $zero, ini_2
        addi       $s1, $s1, -16
        addi       $s0, $s0, -1
        bne        $s0, $zero, ini_1
        ...

```

Figura E6.1: Programa MIPS para el ejercicio 4.

Se pide:

- Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios (justificando su inclusión).
- Dibujar un esquema detallado del TLB y calcular su tamaño incluyendo todos los bits de control necesarios (justificando su inclusión).
- Mostrar de forma exacta cómo evoluciona el contenido de la tabla de páginas durante la ejecución del siguiente programa (ver figura E6.1), determinando en cada referencia a memoria si se produce un acierto/fallo de página y obtener la dirección física resultante. En este apartado no debe tenerse en cuenta el TLB.
- Mostrar de forma exacta la evolución del contenido del TLB para el mismo programa del apartado anterior, determinando en cada referencia a memoria si se produce un acierto/fallo de TLB y/o un acierto/fallo de página y obtener la dirección física resultante.

Notas:

- Suponed que la memoria física y el TLB están inicialmente vacíos.
- Asumid como únicos accesos a memoria las referencias a resultados.
- Asumid que resultados comienza en la dirección virtual 0.

5. La máquina M posee un sistema de memoria virtual con las siguientes características:

- Tabla de páginas de 2048 entradas.
- Memoria física de 1MB.

- Tamaño de página virtual de 2KB.

En dicha máquina se ejecuta el programa en ensamblador de la figura E6.2. Se pide:

- a) Explicar brevemente lo que hace el programa.
- b) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema detallado de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios y justificando la inclusión de cada uno de ellos.
- c) Se sabe que los tres arrays se guardan consecutivamente en memoria a partir de la dirección virtual 8192. Determina la tasa de fallos de página del programa teniendo en cuenta, únicamente, los accesos a los tres arrays, suponiendo que el programa dispone sólo de las páginas físicas 15 y 36 (inicialmente vacías) y utilizando el algoritmo de reemplazo LRU.
- d) Modifica el programa, sin añadir nuevas variables, para que la tasa de fallos se reduzca considerablemente.

6. La máquina M posee un sistema de memoria con las siguientes características:

- Tabla de páginas de 256 entradas.
- Memoria física de 256 bytes.
- Tamaño página virtual de 16 bytes.
- TLB asociativo por conjuntos de 2 vías con una estrategia de reemplazo aleatoria y una política de postescritura.
- El TLB posee 2 conjuntos
- Memoria caché de correspondencia directa combinada para datos e instrucciones con capacidad para 16 palabras (palabras de 32 bits), bloques de 2 palabras y política de postescritura.

Sobre dicha máquina, se ejecuta el programa que aparece en la figura E6.3. Se pide:

- a) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios (justificando su inclusión).
- b) Dibujar un esquema detallado del TLB y de la caché. Calcular sus tamaños incluyendo todos los bits de control necesarios (justificando su inclusión).
- c) Hacer un seguimiento del estado del TLB, tabla de páginas y caché en la ejecución del programa anterior. Tener en cuenta los siguientes datos:
 - Dirección de comienzo del “almacen” $\rightarrow (0)_{10}$
 - Dirección de comienzo instrucciones $\rightarrow (80)_{10}$
 - Los bits de validez de la tabla de páginas están puestos a 1 en las posiciones 0, y 5, siendo los números de página física que ocupan esas posiciones $(12)_{10}$ y $(0)_{10}$ respectivamente.
 - Sabemos que durante la ejecución de este código, en memoria principal está libre la página física $(2)_{10}$. Inicialmente caché y TLB están vacíos.
- d) Calcular la tasa de fallos de instrucciones y datos de la caché en la ejecución del programa anterior. Suponiendo que el procesador está dedicado en exclusiva a la ejecución de nuestra aplicación y que son 20 los ciclos necesarios para acceder a memoria principal y 2000 los necesarios para acceder a memoria secundaria, ¿cuántos ciclos estará parado el procesador sin realizar ninguna tarea debido a los distintos fallos de la caché, el TLB y la tabla de páginas? (En los 2000 ciclos de acceso a memoria secundaria se incluye el tiempo necesario para llevar la página desde memoria secundaria a memoria principal y la posterior actualización de la tabla de páginas y del TLB).

```

##### SEGMENTO DE DATOS #####
        .data
        ...
A:      .space    2048
B:      .space    2048
C:      .space    2048
        ...
##### SEGMENTO DE TEXTO #####
        .text
        ...
PRINCIPAL:  li      $s2, 512
           xor     $s0, $s0, $s0
           xor     $s1, $s1, $s1
           la     $s3, A
           la     $s4, B
BUCLE1:    sw     $s0, 0($s3)
           sw     $s1, 0($s4)
           addi   $s3, $s3, 4
           addi   $s4, $s4, 4
           addi   $s0, $s0, 1
           sll   $s1, $s0, 1
           addi   $s2, $s2, -1
           bne   $s2, $zero, BUCLE1
           li    $s2, 512
           la    $s3, A
           la    $s4, B
           la    $s1, C
BUCLE2:    lw     $s0, 0($s3)
           lw     $t0, 0($s4)
           addi   $s0, $s0, $t0
           sw     $s0, 0($s1)
           addi   $s3, $s3, 4
           addi   $s4, $s4, 4
           addi   $s1, $s1, 4
           addi   $s2, $s2, -1
           bne   $s2, $zero, BUCLE2
        ...

```

Figura E6.2: Programa MIPS para el ejercicio 5.

```

        .data
        ...
almacen: .space 8
        ...
##### SEGMENTO DE TEXTO #####
        .text
        ...
        li    $s0, 2
        li    $s1, 4
        la    $s2, almacen
bucle:  lb     $t0, 0($s2)
        addi  $s2, $s2, 4
        addi  $s0, $s0, -1
        bne  $s0, $zero, bucle
        ...

```

Figura E6.3: Programa MIPS para el ejercicio 6

7. La máquina M posee un sistema de memoria con las siguientes características:

- Tabla de páginas de 512 entradas.
- Memoria física de 256 bytes.
- Tamaño de página virtual/página física de 8 bytes.
- TLB asociativo de 4 conjuntos de 2 vías (cada uno) con una estrategia de reemplazo LRU y una política de postescritura.
- Memoria caché de correspondencia directa combinada para datos e instrucciones con capacidad para 8 palabras (palabras de 32 bits), bloques de 2 palabras y política de postescritura.

Se pide:

- a) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño, incluyendo los bits de control necesarios (justificando su inclusión).
- b) Dibujar un esquema detallado del TLB y de la caché. Calcular sus tamaños incluyendo todos los bits de control necesarios (justificando su inclusión).
- c) Hacer un seguimiento del estado del TLB, la tabla de páginas y la caché en la ejecución del programa mostrado en la figura E6.4, teniendo en cuenta que:
 - La dirección virtual de comienzo del segmento de datos es $0)_{10}$.
 - La dirección virtual de comienzo del segmento de código es $96)_{10}$.
 - Los bits de validez de la tabla de páginas están puestos a 1 en las posiciones 0 y 12, siendo los números de página física que ocupan esas posiciones $16)_{10}$ y $20)_{10}$ respectivamente. La página física 21 y siguientes están todas libres.
 - Inicialmente, la caché y el TLB están vacíos.
- d) Suponiendo que el procesador está dedicado en exclusiva a la ejecución de nuestra aplicación y que son 20 los ciclos necesarios para acceder a memoria principal y 2000 los necesarios para acceder a memoria secundaria, ¿cuántos ciclos estará parado el procesador sin realizar ninguna tarea debido a los distintos fallos de la caché, el TLB y la tabla de páginas? (En los 2000 ciclos de acceso a memoria secundaria se incluye el tiempo necesario para llevar la página desde memoria secundaria a memoria principal y la posterior actualización de la tabla de páginas y del TLB).

```

##### SEGMENTO DE DATOS #####
.data
almacen: .space 96
...
##### SEGMENTO DE TEXTO #####
.text
li      $s0, 2
li      $s1, 0
la      $s2, almacen
lw      $t0, 0($s2)
add     $s1, $s1, $t0

```

Figura E6.4: Programa MIPS para el ejercicio 7

8. La máquina M posee un sistema de memoria virtual con las siguientes características:

- Tabla de páginas de 256 entradas.
- Memoria física de 256 bytes.
- Tamaño de página de 16 bytes.
- TLB asociativo con 4 conjuntos de 2 vías (2 bloques por conjunto). El TLB sigue una estrategia de reemplazo LRU y una política de escritura de postescritura.

Se pide:

- a) Especificar detalladamente el formato de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño incluyendo los bits de control necesarios (justificando su inclusión). Se parte de una situación inicial en la que las páginas virtuales 1, 3, 5 y 7 se encuentran cargadas en memoria física en las páginas físicas 0, 1, 2 y 3, respectivamente.
- b) Dibujar un esquema detallado del TLB y calcular su tamaño incluyendo todos los bits de control necesarios (justificando su inclusión). El TLB se encuentra inicialmente vacío.
- c) Si durante la ejecución de un programa el procesador genera una lectura a la dirección virtual: 52_{10} , y una escritura a la dirección 190_{10} , indicar cuáles son sus correspondientes NPV (números de página virtual), si habrá acierto o fallo en la tabla de páginas, cuáles son sus NPF (números de página física) y la dirección física resultante. Suponer en este apartado que nuestra máquina no dispone de TLB. (Se sabe que la página física 4 se encuentra libre).
- d) Repetir el apartado anterior, pero ahora teniendo en cuenta que nuestra máquina dispone de TLB.

9. Se desea diseñar una memoria virtual para un procesador que cumpla los siguientes requisitos:

- Direcciones físicas de 24 bits y memoria direccionable por bytes.
- La consulta de la tabla de páginas debe realizarse en un único acceso a memoria. Además, la tabla de páginas no puede ocupar más del 6,25% (la dieciseisava parte) de la capacidad máxima de la memoria física. Cada entrada de la tabla de páginas tiene un tamaño de 2 bytes.
- Para los fallos de página se empleará una política de reemplazo pseudo-LRU (un sólo bit de uso) y una estrategia de post-escritura.

Se pide:

<pre> .data E1: .word 1,2,3,4,5,6,7,8 .word 8,7,6,5,4,3,2,1 E2: .word 0,1,0,1,0,0,1,1 .word 0,1,1,0,1,0,0,1 S: .word 0,0,0,0,0,0,0,0 .word 0,0,0,0,0,0,0,0 .text main: la \$s0,E1 la \$s1,E2 la \$s2,S li \$s3,0 li \$t1,16 li \$t0,0 </pre>	<pre> li \$t7,3 loop: beq \$t0,\$t1,fin sll \$t2,\$t0,2 add \$t3,\$s0,\$t2 lw \$t4,0(\$t3) add \$t3,\$s1,\$t2 lw \$t5,0(\$t3) add \$t6,\$t4,\$t5 add \$t3,\$s2,\$t2 sw \$t6,0(\$t3) addi \$t0,\$t0,1 j loop fin: ... </pre>
--	---

Figura E6.5: Programa MIPS para el ejercicio 10

- a) Según las especificaciones anteriores, diseñar el formato óptimo de las direcciones virtuales y físicas y de la tabla de páginas si se desea implantar una memoria virtual con la mayor capacidad posible. Especificar detalladamente el formato de la dirección física y de la dirección virtual, dibujar un esquema de la tabla de páginas detallando los bits de control necesarios (justificando su inclusión).
- b) Se desea implantar también un TLB 2-asociativo de dos conjuntos (4 bloques en total) con estrategia de reemplazo LRU y una política de post-escritura. Dibujar un esquema detallado del TLB y calcular el tamaño exacto incluyendo los bits de control necesarios (justificando su inclusión).
- c) Mostrar de forma exacta cómo evoluciona el contenido del TLB y de la tabla de páginas después de cada una de las siguientes referencias virtuales generadas por el procesador expresadas en **hexadecimal**: lectura 810, lectura 1010, escritura 800 y escritura 2000. Obtener también la dirección física resultante. (Nota: para los fallos de página, suponga que las primeras direcciones bajas de memoria están libres.)
10. Supongamos una máquina M con un sistema de memoria virtual que tiene direcciones virtuales de 14 bits, 32 páginas físicas y un tamaño de página de 32 bytes. Dicho sistema de memoria virtual posee un TLB asociativo de 2 vías con reemplazo LRU y postescritura capaz de almacenar 8 entradas de la tabla de páginas.
- Sobre dicha máquina se va a ejecutar el código en MIPS que se muestra en la figura E6.5, donde el segmento de datos está situado al comienzo de la memoria virtual (dirección 0x0). Sabiendo que las palabras son de 4 bytes, se pide:
- a) Especificar el formato de la dirección virtual y de la dirección física. Dibujar un esquema detallado de la tabla de páginas y del TLB, incluyendo todos los bits de control necesarios, justificando su inclusión. Hallar el tamaño de ambas estructuras en bits (sin redondear las entradas a múltiplo de byte).
- b) Mostrar la evolución del contenido de la tabla de páginas y del TLB en la primera iteración del bucle considerando únicamente las referencias a datos generadas por el procesador. Mostrar también las direcciones físicas resultantes sabiendo que el S.O. ha reservado las páginas físicas 1, 8 y 27 para almacenar los datos utilizados por el programa. Asumir que en un principio tanto la memoria principal (y, por tanto, la tabla de páginas) como el TLB están vacíos. Indicar si se produce un fallo o un acierto tanto en la tabla de páginas como en el TLB.

- c) Describir lo que hace el programa y mostrar el contenido final del TLB y de la tabla de páginas una vez finalizada la ejecución del programa. Calcular además el número de accesos a disco duro que realizará el S.O. durante la ejecución de este programa.
11. Dado un sistema de memoria virtual con una tabla de páginas de 1024 entradas, una memoria física de tamaño 1 KB, un tamaño de página de 32 bytes y un TLB asociativo de 2 vías y un total de 4 bloques con una estrategia de reemplazo LRU y política de postescritura. Se pide:
- Especificar el formato y tamaño de la dirección física y de la dirección virtual. Dibujar un esquema de la tabla de páginas y calcular su tamaño exacto (sin redondear entradas a múltiplo de bytes) incluyendo los bits de control necesarios (justificando su inclusión). Se parte de una situación inicial en la que las páginas virtuales 1 y 3 se encuentran cargadas en las páginas físicas 5 y 8, respectivamente. Además, los bits de uso y modificación de las páginas 1 y 3 están inicialmente a 0.
 - Dibujar un esquema detallado del TLB y calcular su tamaño exacto incluyendo todos los bits de control necesarios (justificando su inclusión). El TLB se encuentra inicialmente vacío.
 - Mostrar la evolución del contenido del TLB y la tabla de páginas después de cada una de las siguientes referencias (direcciones virtuales) generadas por el procesador: Lectura $24)_{16}$ y Escritura $44)_{16}$. Determinar en cada acceso si se produce un acierto/fallo de TLB y/o un acierto/fallo de página y obtener la dirección física resultante. Suponer que la asignación de páginas físicas se hará de forma ordenada y comenzando por la número 0 (la primera disponible).
 - El TLB es común para datos e instrucciones y tiene una tasa de fallos de 5 % y una penalización en caso de fallo de 100 ciclos (acceso a la tabla de páginas en MP) y por otro lado tenemos una caché compartida para datos e instrucciones con una tasa de fallos de 10 % y una penalización por fallo de 120 ciclos. Si el CPI ideal es de 2 ciclos y sabemos que hay un 20 % de instrucciones que son de carga o almacenamiento, calcula el CPI real de la máquina teniendo en cuenta las detenciones producidas por la caché y por el TLB y suponiendo que no se producen fallos de página.
12. La máquina `eme.inf.um.es` posee un sistema de memoria virtual con las siguientes características:
- Tabla de páginas de 2048 entradas.
 - Memoria física de 1 MB.
 - Tamaño de página de 1 KB.
 - TLB asociativo con 4 conjuntos de 2 vías, usando una estrategia de reemplazo LRU y una política de escritura de postescritura.

Se parte de una situación inicial en la que:

- Las páginas virtuales 1, 3, 5 y 7 del proceso P se encuentran cargadas en las páginas físicas 0, 1, 2 y 3, respectivamente.
- Todos los bits de control de la tabla de páginas del proceso P están a cero, excepto los bits de validez de las páginas 1, 3, 5 y 7.
- Todas las páginas físicas están ocupadas por otros procesos distintos de P, excepto las páginas físicas 0, 1, 2, 3 y 35.
- El TLB está completamente vacío.

Se pide:

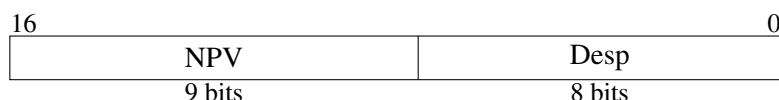
- Especifique detalladamente el formato de la dirección física y de la dirección virtual.

- b) Dibuje un esquema de la tabla de páginas y calcule su tamaño incluyendo los bits de control necesarios, justificando la inclusión de cada uno de ellos.
- c) Dibuje un esquema detallado del TLB y calcule su tamaño incluyendo todos los bits de control necesarios, justificando la inclusión de cada uno de ellos.
- d) El proceso P genera la siguiente secuencia de accesos a memoria:
- Lectura a la dirección 0x124.
 - Lectura de la dirección 0x422.
 - Escritura a la dirección 0x124.
 - Escritura a la dirección 0x1418.
 - Lectura a la dirección 0x1428.
 - Escritura a la dirección 0x1d18.
 - Lectura a la dirección 0x1620.
- Suponga, para este apartado, que la máquina no tiene TLB. Indique para cada acceso:
- El número de página virtual (NPV).
 - Desplazamiento dentro de la página.
 - Si se produce un acierto o fallo de página.
 - Número de página física (NPF).
 - Dirección física resultante.
 - Estado exacto de la entrada correspondiente de la tabla de páginas después del acceso.
- e) Repita el apartado anterior teniendo en cuenta el TLB de la máquina. En este caso, para cada acceso debe indicar:
- El número de página virtual (NPV).
 - Desplazamiento dentro de la página.
 - Si se produce un acierto o fallo de TLB.
 - Si se produce un acierto o fallo de página.
 - Número de página física (NPF).
 - Dirección física resultante.
 - Estado exacto de la entrada correspondiente del TLB después del acceso.
 - Estado exacto de la entrada correspondiente de la tabla de páginas después del acceso.
- f) Calcule el número total de accesos a memoria (incluyendo los propios accesos para leer cada dato) y a disco que se producen en el apartado anterior. Suponga que `eme.inf.um.es` no dispone de caché.

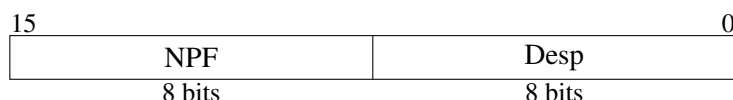
E6.2. Solución a ejercicios seleccionados

1. a) Si la tabla de páginas tiene 512 entradas, eso es porque hay 512 páginas virtuales, luego el campo NPV de la dirección virtual necesitará $\log_2 512 = 9$ bits.

Como las páginas tienen un tamaño de 256 bytes, eso significa que el campo de desplazamiento de la dirección virtual necesita $\log_2 256 = 8$ bits. La dirección virtual, por lo tanto, tiene un tamaño total de $9 + 8 = 17$ bits y queda formada de la siguiente manera:



En el caso de la dirección física, sabemos que la memoria principal tiene un tamaño total de 64 KB, por lo que la dirección física tiene un tamaño total de $\log_2 64 \times 2^{10} = 16$ bits. De esos, 8 bits constituyen el desplazamiento y el resto (otros 8 bits) el NPF. La dirección física queda, pues, de la siguiente manera:



Cada entrada de la tabla de páginas necesitará un bit de modificación (M) para permitir la política de postescritura y un bit de uso (U) para permitir la política de reemplazo LRU, además del bit de validez (V). La tabla de páginas queda así:

	V	M	U	NPF (8 bits)
0	1	0	0	8
1	1	0	0	7
2	1	0	0	6
3	1	0	0	5
4	1	0	0	4
5	1	0	0	3
6	1	0	0	2
7	1	0	0	1
8	0	–	–	–
9	0	–	–	–
10	0	–	–	–
...
511	0	–	–	–

Cada entrada de la tabla de páginas tiene (al menos) 11 bits, por lo que en total se necesitarán $512 \times 11 = 5632$ bits. Si se redondea el tamaño de cada entrada a 16 bits (que es el tamaño de palabra según otro apartado del problema) se necesitarán en total $512 \times 2 = 1024$ bytes = 1 KB.

- b) Debido a que el TLB tiene 16 conjuntos, los 4 bits de menor peso del NPV se utilizarán para elegir la posición en la caché, mientras que los 5 restantes formarán la etiqueta.

Cada conjunto tiene 2 vías. Cada vía necesitará un bit de modificación (M) para permitir la política de postescritura y un bit de validez (V) para saber cuándo la vía contiene información útil. No es necesario un bit de uso (U) porque el TLB sigue una política de reemplazo aleatorio. Además de esos bits, cada vía contendrá también los bits M y U de la entrada correspondiente de la tabla de páginas (que mostraremos como M' y U', respectivamente) y el campo NPF con el correspondiente número de página física. El TLB queda, pues, de la siguiente forma:

Vía 0					
V	M	Etiqueta (5 bits)	M'	U'	NPF (8 bits)
0	0	-	-	-	-
1	0	-	-	-	-
2	0	-	-	-	-
...
15	0	-	-	-	-

Vía 1					
V	M	Etiqueta (5 bits)	M'	U'	NPF (8 bits)
0	-	-	-	-	-
0	-	-	-	-	-
0	-	-	-	-	-
...
0	-	-	-	-	-

Por tanto, el tamaño total del TLB será de $16 \times 2 \times (1 + 1 + 5 + 1 + 1 + 8) = 544$ bits.

- c) • Lectura de la dirección $25 = 0x19$.

NPV: 0.

Desplazamiento: $0x19$.

Acierto/fallo de TLB: fallo (el TLB está inicialmente vacío).

Acierto/fallo de página: acierto (según el bit de validez de la entrada 0 de la tabla de páginas).

Conjunto del TLB: 0, y queda de la siguiente manera:

Vía 0					
V	M	Etiqueta (5 bits)	M'	U'	NPF (8 bits)
0	1	1	0	1	8

Vía 1					
V	M	Etiqueta (5 bits)	M'	U'	NPF (8 bits)
0	-	-	-	-	-

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

NPV: 8.

Dirección física resultante: $0x819$.

Entrada de la tabla de páginas: no cambia.

- Escritura en la dirección $904 = 0x388$.

NPV: 3.

Desplazamiento: $0x88$.

Acierto/fallo de TLB: fallo, ya que el conjunto 3 del TLB está vacío.

Acierto/fallo de página: acierto (según el bit de validez de la entrada 3 de la tabla de páginas).

Conjunto del TLB: 3, y queda de la siguiente manera:

Vía 0					
V	M	Etiqueta (5 bits)	M'	U'	NPF (8 bits)
3	1	1	1	1	5

Vía 1					
V	M	Etiqueta (5 bits)	M'	U'	NPF (8 bits)
0	-	-	-	-	-

Ya que el bit de uso U' y el bit de modificación M' se ponen a 1, el bit M del TLB debe ponerse también a 1.

NPV: 5.

Dirección física resultante: $0x588$.

Entrada de la tabla de páginas: no cambia.

- Lectura de la dirección $75 = 0x4B$.

NPV: 0.

Desplazamiento: $0x4B$.

Acierto/fallo de TLB: acierto, ya que el conjunto 0 del TLB tiene la vía 0 ocupada y las etiquetas coinciden.

Conjunto del TLB: no cambia.

NPV: 8.

Dirección física resultante: $0x84B$.

- Escritura en la dirección $0x588 = 0101\ 1000\ 1000)_2$
Nº de bloque: 177.
Acierto/fallo de caché: fallo (la posición 1 de la caché está vacía).
Posición de la caché: 1, y queda de la siguiente manera:

	V	M	Etiqueta (10 bits)	Bloque (4x16=64 bits)
0	0	–	–	–
1	1	1	22	Bloque[177]
2	0	–	–	–
3	1	0	32	Bloque[259]
4	0	–	–	–
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

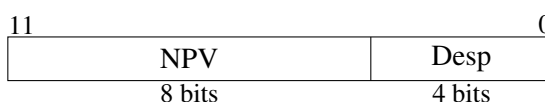
Al ser una caché de postescritura, esta escritura supone leer primero de memoria el bloque 177 que posteriormente es modificado, ya en caché, por la escritura.

- Lectura de la dirección $0x84B = 1000\ 0100\ 1011)_2$
Nº de bloque: 265.
Acierto/fallo de caché: fallo (la posición 1 está ocupada, pero su etiqueta no es la correcta).
Posición de la caché: 1, y queda de la siguiente manera:

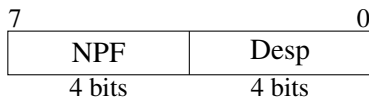
	V	M	Etiqueta (10 bits)	Bloque (4x16=64 bits)
0	0	–	–	–
1	1	0	33	Bloque[265]
2	0	–	–	–
3	1	0	32	Bloque[259]
4	0	–	–	–
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

Al sustituir el bloque de la posición 1 por un bloque distinto, y al estar modificado el bloque que se expulsa (su bit M está a 1), esta lectura supone también la escritura en memoria del bloque 177.

6. a) Si la tabla de páginas tiene 256 entradas, eso es porque hay 256 páginas virtuales, luego el campo NPV de la dirección virtual necesitará $\log_2 256 = 8$ bits. Como las páginas tienen un tamaño de 16 bytes, eso significa que el campo de desplazamiento de la dirección virtual necesita $\log_2 16 = 4$ bits. La dirección virtual, por lo tanto, tiene un tamaño total de $8 + 4 = 12$ bits y queda formada de la siguiente manera:



En el caso de la dirección física, sabemos que la memoria principal tiene un tamaño total de 256 bytes, por lo que la dirección física tiene un tamaño total de $\log_2 256 = 8$ bits. De esos, 4 bits constituyen el desplazamiento y el resto (otros 4 bits) el NPF. La dirección física queda, pues, de la siguiente manera:



Cada entrada de la tabla de páginas necesitará un bit de modificación (M) para permitir la política de postescritura y un bit de uso (U) para permitir la política de reemplazo LRU, además del bit de validez (V). La tabla de páginas queda así:

	V	M	U	NPF (4 bits)
0	1	0	0	12
1	0	–	–	–
2	0	–	–	–
3	0	–	–	–
4	0	–	–	–
5	1	0	0	0
6	0	–	–	–
7	0	–	–	–
8	0	–	–	–
9	0	–	–	–
...
255	0	–	–	–

Como podemos ver, siguiendo el enunciado del ejercicio, las entradas 0 y 5 de la tabla de páginas tienen el bit de validez a 1, lo que indica que las páginas virtuales correspondientes ya se encuentran en memoria principal en las páginas físicas 12 y 0, respectivamente.

Cada entrada de la tabla de páginas tiene (al menos) 7 bits, por lo que en total se necesitarán $256 \times 7 = 1792$ bits. Si se redondea el tamaño de cada entrada a 8 bits (1 byte) se necesitarán en total $256 \times 1 = 256$ bytes.

- b) Debido a que el TLB tiene 2 conjuntos, el bit de menor peso del NPV se utilizará para elegir la posición en el TLB, mientras que los 7 restantes formarán la etiqueta.

Cada conjunto tiene 2 vías. Cada vía necesitará un bit de modificación (M) para permitir la política de postescritura y un bit de validez (V) para saber cuándo la vía contiene información útil. No necesitará, sin embargo, un bit de uso (U) ya que se sigue una política de reemplazo aleatorio. Además de esos bits, cada vía contendrá también los bits M y U de la entrada correspondiente de la tabla de páginas (que mostraremos como M' y U', respectivamente) y el campo NPF con el correspondiente número de página física. El TLB queda, pues, de la siguiente forma:

		Vía 0						Vía 1					
		V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)	V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)
0		0	–	–	–	–	–	0	–	–	–	–	–
1		0	–	–	–	–	–	0	–	–	–	–	–

Por tanto, el tamaño total del TLB será de $2 \times 2 \times (1 + 1 + 7 + 1 + 1 + 4) = 60$ bits.

En el caso de la caché, al decirnos que es de correspondencia directa, que tiene una capacidad de 16 palabras, que las palabras son de 4 bytes y que los bloques son de 2 palabras, lo que nos están diciendo es que la caché tiene $16/2=8$ posiciones y que, por tanto, el campo índice de la dirección física tiene un tamaño de 3 bits. También nos están diciendo que el campo de desplazamiento de palabra es de 1 bit (al ser los bloques de 2 palabras) y que el desplazamiento de byte es de 2 bits (al ser las palabras de 4 bytes). Todo esto hace que una dirección física de 8 bytes se descomponga de la siguiente manera:

2 bits	3 bits	1 bit	2 bits
Etiqueta	Índice	Desp. palabra	Desp. byte

y que la caché quede con la siguiente estructura:

	V	M	Etiqueta (2 bits)	Datos (2 palabras)
0	0	–	–	–
1	0	–	–	–
2	0	–	–	–
3	0	–	–	–
4	0	–	–	–
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

Se puede ver que se ha añadido un bit de modificación (M) ya que la caché es de postescritura. El tamaño total de la caché es, por tanto, $8 \times (1 + 1 + 2 + 2 \times 32) = 544$ bits.

c) Para hacer un seguimiento del TLB, la tabla de páginas y la caché durante la ejecución del programa, lo primero que necesitamos saber es qué direcciones virtuales genera el programa, sabiendo que la primera instrucción se encuentra en la dirección virtual 80 y que el almacen se encuentra en la dirección virtual 0:

li \$s0, 2: como el valor inmediato es pequeño y se puede codificar en 16 bits en complemento a 2, ésta es una pseudoinstrucción que se traduce a una única instrucción real, almacenada en la dirección virtual 80.

li \$s1, 4: es similar a la instrucción anterior. Se almacena en la dirección virtual 84.

la \$s2, almacen: de manera similar a las dos instrucciones anteriores, como “almacen” es 0 y se puede codificar en 16 bits, esta pseudoinstrucción se traduce en una única instrucción real almacenada en la dirección virtual 88.

lb \$t0, 0(\$s2): instrucción real en la dirección virtual 92. Esta instrucción en la primera pasada del bucle accede a la dirección virtual \$s2+0, que es 0 al ser \$s2=0. En la segunda (y última) pasada del bucle, esta instrucción lee de la dirección virtual 4, ya que \$s2 se incrementa en 4 en cada pasada.

addi \$s2, \$s2, 4: instrucción real en la dirección virtual 96.

addi \$s0, \$s0, -1: instrucción real en la dirección virtual 100.

bne \$s0, \$zero, bucle: instrucción real en la dirección virtual 104.

En resumen, el programa, durante su ejecución, genera la siguiente secuencia de referencias a memoria: 80, 84, 88, 92, 0, 96, 100, 104, 92, 4, 96, 100 y 104. Veamos cómo evolucionan el TLB y la tabla de páginas durante esta secuencia de accesos:

- Lectura de la dirección 80 = 0x50.

NPV: 5.

Desplazamiento: 0.

Acierto/fallo de TLB: fallo (el TLB está inicialmente vacío).

Acierto/fallo de página: acierto (según el bit de validez de la entrada 5 de la tabla de páginas).

Conjunto del TLB: 1, y queda de la siguiente manera:

Vía 0						Vía 1						
	V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)	V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)
0	0	–	–	–	–	–	0	–	–	–	–	–
1	1	1	2	0	1	0	0	–	–	–	–	–

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

NPF: 0.

Dirección física resultante: $0x00 = 0$.

Entrada de la tabla de páginas: no cambia.

- Lectura de la dirección $84 = 0x54$.

NPV: 5.

Desplazamiento: 4.

Acierto/fallo de TLB: acierto (la etiqueta es 2 y el conjunto 1 del TLB contienen una entrada con la misma etiqueta).

Conjunto del TLB: El TLB no cambia.

NPF: 0.

Dirección física resultante: $0x04 = 4$.

Entrada de la tabla de páginas: no cambia.

- Lectura de la dirección $88 = 0x58$.

NPV: 5.

Desplazamiento: 8.

Acierto/fallo de TLB: acierto.

Conjunto del TLB: El TLB no cambia.

NPF: 0.

Dirección física resultante: $0x08 = 8$.

Entrada de la tabla de páginas: no cambia.

- Lectura de la dirección $92 = 0x5C$.

NPV: 5.

Desplazamiento: $0xC = 12$.

Acierto/fallo de TLB: acierto.

Conjunto del TLB: El TLB no cambia.

NPF: 0.

Dirección física resultante: $0x0C = 12$.

Entrada de la tabla de páginas: no cambia.

- Lectura de la dirección 0.

NPV: 0.

Desplazamiento: 0.

Acierto/fallo de TLB: fallo, ya que el conjunto 0 del TLB está vacío.

Acierto/fallo de página: acierto (según el bit de validez de la entrada 0 de la tabla de páginas).

Conjunto del TLB: 0, y queda de la siguiente manera:

		Vía 0						Vía 1					
		V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)	V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)
0	1	1	1	0	0	1	12	0	-	-	-	-	-
1	1	1	1	2	0	1	0	0	-	-	-	-	-

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

NPF: $12 = 0xC$.

Dirección física resultante: $0xC0 = 192$.

Entrada de la tabla de páginas: no cambia.

- Lectura de la dirección 96 = 0x60.

NPV: 6.

Desplazamiento: 0.

Acierto/fallo de TLB: fallo, ya que la etiqueta es 3 y el conjunto 0 del TLB no tiene una entrada con esa etiqueta.

Acierto/fallo de página: fallo. Según el enunciado, la página física 2 está libre, por lo que la tabla de páginas queda de la siguiente manera:

	V	M	U	NPF (4 bits)
0	1	0	0	12
1	0	-	-	-
2	0	-	-	-
3	0	-	-	-
4	0	-	-	-
5	1	0	0	0
6	1	0	0	2
7	0	-	-	-
8	0	-	-	-
9	0	-	-	-
...
255	0	-	-	-

Conjunto del TLB: 0, y queda de la siguiente manera:

		Vía 0					Vía 1					
	V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)	V	M	Etiqueta (7 bits)	M'	U'	NPF (4 bits)
0	1	1	0	0	1	12	1	1	3	0	1	2
1	1	1	2	0	1	0	0	-	-	-	-	-

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

NPV: 2.

Dirección física resultante: 0x20 = 32.

- La dos referencias a memoria que quedan (la 100 y la 104) para finalizar la primera iteración del bucle producen aciertos de TLB porque pertenecen a la página virtual 6 cuya información acabamos de cargar en el TLB. En la segunda pasada del bucle, todos los accesos también producen aciertos de TLB. Por lo tanto, ni el TLB ni la tabla de páginas cambian más, quedando como hemos mostrado en el acceso anterior.

En el caso de la caché, debemos considerar las direcciones físicas producidas por los accesos a memoria. Teniendo en cuenta las traducciones que acabamos de realizar, estas direcciones físicas son: 0, 4, 8, 12, 192, 32, 36, 40, 12, 196, 32, 36 y 40. La evolución de la caché es la siguiente:

- Lectura de la dirección 0.

Nº de bloque: 0.

Acierto/fallo de caché: fallo (la caché está inicialmente vacía).

Posición de la caché: 0, y queda de la siguiente manera:

	V	M	Etiqueta (2 bits)	Datos (2 palabras)
0	1	0	0	Bloque[0]
1	0	–	–	–
2	0	–	–	–
3	0	–	–	–
4	0	–	–	–
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

- Lectura de la dirección 4.
Nº de bloque: 0.
Acierto/fallo de caché: acierto y la caché no cambia.
Posición de la caché: 0.
- Lectura de la dirección 8.
Nº de bloque: 1.
Acierto/fallo de caché: fallo.
Posición de la caché: 1, y queda de la siguiente manera:

	V	M	Etiqueta (2 bits)	Datos (2 palabras)
0	1	0	0	Bloque[0]
1	1	0	0	Bloque[1]
2	0	–	–	–
3	0	–	–	–
4	0	–	–	–
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

- Lectura de la dirección 12 = 0xC.
Nº de bloque: 1.
Acierto/fallo de caché: acierto y la caché no cambia.
Posición de la caché: 1.
- Lectura de la dirección 192 = 0xC0.
Nº de bloque: 24.
Acierto/fallo de caché: fallo, ya que la etiqueta de la posición 0 no coincide.
Posición de la caché: 0, y queda de la siguiente manera:

	V	M	Etiqueta (2 bits)	Datos (2 palabras)
0	1	0	3	Bloque[24]
1	1	0	0	Bloque[1]
2	0	–	–	–
3	0	–	–	–
4	0	–	–	–
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

- Lectura de la dirección 32 = 0x20.

Nº de bloque: 4.

Acierto/fallo de caché: fallo, ya que esa posición de la caché está vacía.

Posición de la caché: 4, y queda de la siguiente manera:

	V	M	Etiqueta (2 bits)	Datos (2 palabras)
0	1	0	3	Bloque[24]
1	1	0	0	Bloque[1]
2	0	–	–	–
3	0	–	–	–
4	1	0	0	Bloque[4]
5	0	–	–	–
6	0	–	–	–
7	0	–	–	–

- Lectura de la dirección $36 = 0x24$.

Nº de bloque: 4.

Acierto/fallo de caché: acierto y la caché no cambia.

Posición de la caché: 4.

- Lectura de la dirección $40 = 0x28$.

Nº de bloque: 5.

Acierto/fallo de caché: fallo, ya que esa posición de la caché está vacía.

Posición de la caché: 5, y queda de la siguiente manera:

	V	M	Etiqueta (2 bits)	Datos (2 palabras)
0	1	0	3	Bloque[24]
1	1	0	0	Bloque[1]
2	0	–	–	–
3	0	–	–	–
4	1	0	0	Bloque[4]
5	1	0	0	Bloque[5]
6	0	–	–	–
7	0	–	–	–

- En la segunda pasada del bucle, todos los accesos producen aciertos de caché, ya que las palabras de las direcciones 12, 196, 32, 36 y 40, correspondientes a los bloques 1, 24, 4 y 5, ya se encuentran en caché.

d) En el caso de las instrucciones, de las 11 instrucciones que se ejecutan, 4 producen fallo de caché, por lo que la tasa de fallos de instrucciones es $4/11=36,4\%$.

En el caso de los datos, de los 2 accesos que se producen, 1 provoca un fallo de caché, por lo que la tasa de fallos de datos es $1/2=50\%$.

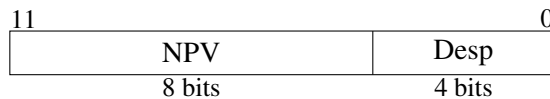
En cuanto al tiempo que el procesador está parado debido a los distintos fallos, si suponemos que los 20 ciclos de acceso a memoria son por palabra, tenemos lo siguiente:

- Fallos de caché: 5, donde en cada uno se leen dos palabras. Tiempo = $5 \times 2 \times 20 = 200$ ciclos.
- Fallos de TLB sin fallos de página: 2. Aunque cada entrada de la tabla de páginas ocupa un único byte, de memoria se lee, como mínimo, una palabra, tardando 20 ciclos por palabra. Tiempo = $2 \times 20 = 40$ ciclos.
- Fallos de TLB con fallos de página: 1, que tarda 2000 ciclos en ser resuelto.

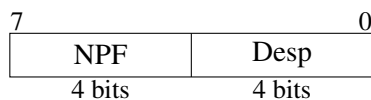
En total, el procesador está parado $200 + 40 + 2000 = 2240$ ciclos por los distintos fallos que se producen.

8. a) Si la tabla de páginas tiene 256 entradas, eso es porque hay 256 páginas virtuales, luego el campo NPV de la dirección virtual necesitará $\log_2 256 = 8$ bits.

Como las páginas tienen un tamaño de 16 bytes, eso significa que el campo de desplazamiento de la dirección virtual necesita $\log_2 16 = 4$ bits. La dirección virtual, por lo tanto, tiene un tamaño total de $8 + 4 = 12$ bits y queda formada de la siguiente manera:



En el caso de la dirección física, sabemos que la memoria principal tiene un tamaño total de 256 bytes, por lo que la dirección física tiene un tamaño total de $\log_2 256 = 8$ bits. De esos, 4 bits constituyen el desplazamiento y el resto (otros 4 bits) el NPF. La dirección física queda, pues, de la siguiente manera:



Cada entrada de la tabla de páginas necesitará un bit de modificación (M) para permitir la política de postescritura y un bit de uso (U) para permitir la política de reemplazo LRU, además del bit de validez (V). La tabla de páginas queda así:

	V	M	U	NPF (4 bits)
0	0	–	–	–
1	1	0	0	0
2	0	–	–	–
3	1	0	0	1
4	0	–	–	–
5	1	0	0	2
6	0	–	–	–
7	1	0	0	3
8	0	–	–	–
9	0	–	–	–
...
255	0	–	–	–

Cada entrada de la tabla de páginas tiene (al menos) 7 bits, por lo que en total se necesitarán $256 \times 7 = 1792$ bits. Si se redondea el tamaño de cada entrada a 8 bits (1 byte) se necesitarán en total $256 \times 1 = 256$ bytes.

- b) Debido a que el TLB tiene 4 conjuntos, los 2 bits de menor peso del NPV se utilizarán para elegir la posición en la caché, mientras que los 6 restantes formarán la etiqueta.

Cada conjunto tiene 2 vías. Cada vía necesitará un bit de modificación (M) para permitir la política de postescritura, un bit de uso (U) por seguir una política de reemplazo LRU y un bit de validez (V) para saber cuándo la vía contiene información útil. Además de esos bits, cada vía contendrá también los bits M y U de la entrada correspondiente de la tabla de páginas (que mostraremos como M' y U', respectivamente) y el campo NPF con el correspondiente número de página física. El TLB queda, pues, de la siguiente forma:

Vía 0							Vía 1								
	V	M	U	Etiqueta (6 bits)	M'	U'	NPF (4 bits)		V	M	U	Etiqueta (6 bits)	M'	U'	NPF (4 bits)
0	0	-	-	-	-	-	-		0	-	-	-	-	-	-
1	0	-	-	-	-	-	-		0	-	-	-	-	-	-
2	0	-	-	-	-	-	-		0	-	-	-	-	-	-
3	0	-	-	-	-	-	-		0	-	-	-	-	-	-

Por tanto, el tamaño total del TLB será de $4 \times 2 \times (1 + 1 + 1 + 6 + 1 + 1 + 4) = 120$ bits.

- c) • Lectura de la dirección $52 = 0x34$.

NPV: 3.

Desplazamiento: 4.

Acierto/fallo de página: acierto (según el bit de validez de la entrada 3 de la tabla de páginas). Habrá que poner a 1 el bit de uso de la entrada correspondiente de la tabla de páginas, quedando ésta como sigue:

	V	M	U	NPF (4 bits)
0	0	-	-	-
1	1	0	0	0
2	0	-	-	-
3	1	0	1	1
4	0	-	-	-
5	1	0	0	2
6	0	-	-	-
7	1	0	0	3
8	0	-	-	-
9	0	-	-	-
...
255	0	-	-	-

NPV: 1.

Dirección física resultante: $0x14$.

- Escritura en la dirección $190 = 0xBE$.

NPV: $0xB = 11$.

Desplazamiento: $0xE = 14$.

Acierto/fallo de página: fallo (según el bit de validez de la entrada 11 de la tabla de páginas). Como nos dicen que la página física 4 está libre, podemos leer la página virtual 11 del disco y colocarla en la página física 4. La tabla de páginas queda de la siguiente manera:

	V	M	U	NPF (4 bits)
0	0	-	-	-
1	1	0	0	0
2	0	-	-	-
3	1	0	1	1
...
11	1	1	1	4
...
255	0	-	-	-

Como vemos, el bit de uso se pone a 1 por la referencia a la página. El bit de modificación también se pone a 1 al tratarse de una escritura.

NPV: 4.

- Dirección física resultante:** 0x4E.
- d) • **Lectura de la dirección** 52 = 0x34.
NPV: 3.
Desplazamiento: 4.
Acierto/fallo de TLB: fallo (el TLB está inicialmente vacío).
Acierto/fallo de página: acierto (según el bit de validez de la entrada 3 de la tabla de páginas).
Conjunto del TLB: 3 (los dos bits menos significativos del NPV) y queda de la siguiente manera:

Vía 0							Vía 1								
	V	M	U	Etiqueta (6 bits)	M'	U'	NPF (4 bits)		V	M	U	Etiqueta (6 bits)	M'	U'	NPF (4 bits)
0	0	-	-	-	-	-	-	0	0	-	-	-	-	-	-
1	0	-	-	-	-	-	-	1	0	-	-	-	-	-	-
2	0	-	-	-	-	-	-	2	0	-	-	-	-	-	-
3	1	1	1	0	0	1	1	3	0	-	-	-	-	-	-

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

- NPF:** 1.
Dirección física resultante: 0x14.
Entrada de la tabla de páginas: no cambia.
- **Escritura en la dirección** 190 = 0xBE.
NPV: 0xB = 11.
Desplazamiento: 0xE = 14.
Acierto/fallo de TLB: fallo, ya que, aunque el conjunto 3 del TLB tiene la primera vía ocupada, las etiquetas no coinciden (0 ≠ 2).
Acierto/fallo de página: fallo (según el bit de validez de la entrada 11 de la tabla de páginas).
 Ya que la página física 4 está libre, leemos la página virtual 11 del disco y la colocamos ahí. La tabla de páginas queda de la siguiente manera:

	V	M	U	NPF (4 bits)
0	0	-	-	-
1	1	0	0	0
2	0	-	-	-
3	1	0	0	1
...
11	1	0	0	4
...
255	0	-	-	-

Conjunto del TLB: 3, y queda de la siguiente manera:

Vía 0							Vía 1								
	V	M	U	Etiqueta (6 bits)	M'	U'	NPF (4 bits)		V	M	U	Etiqueta (6 bits)	M'	U'	NPF (4 bits)
0	0	-	-	-	-	-	-	0	0	-	-	-	-	-	-
1	0	-	-	-	-	-	-	1	0	-	-	-	-	-	-
2	0	-	-	-	-	-	-	2	0	-	-	-	-	-	-
3	1	1	0	0	0	1	1	3	1	1	1	2	1	1	4

Ya que el bit de uso U' y el bit de modificación M' se ponen a 1, el bit M del TLB debe ponerse también a 1.

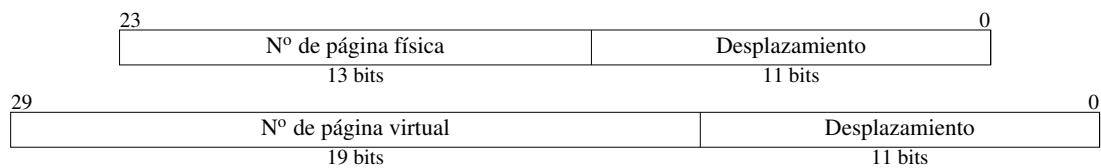
NPF: 4.

Dirección física resultante: 0x4E.

Entrada de la tabla de páginas: cambia la entrada 11, como ya se ha indicado.

9. a) Los 24 bits de las direcciones físicas de memoria indican que disponemos de una memoria física máxima de 16MB (2^{24} bytes), de los cuales reservamos 1/16 para alojar la tabla de páginas cuyo tamaño será de 1MB. Con ese tamaño tendremos 512K-entradas para la tabla de páginas con dos bytes por entrada. En esos dos bytes se debe codificar el bit de uso (U), el bit de modificación (M) y el bit de validez (V), por lo que quedan 13 bits para el número de página física. El campo desplazamiento serán los 11 bits restantes hasta conseguir los 24 bits totales de la dirección física de memoria. Luego el tamaño de página será de 2KB (2^{11} bytes).

Como tenemos 512K-entradas en la tabla de páginas, necesitamos 19 bits para indicar el número de entrada, es decir, el número de página virtual. Luego el tamaño de la memoria virtual será de 1GB (2^{19+11} bytes). Los formatos de la dirección física y de la dirección virtual serán, respectivamente:



La tabla de páginas queda de la siguiente manera:

	V	M	U	NPF (13 bits)
0	0	-	-	-
1	0	-	-	-
2	0	-	-	-
...
$2^{19} - 2$	0	-	-	-
$2^{19} - 1$	0	-	-	-

El bit V (validez) siempre se debe incluir para saber si el contenido de una entrada es válido o no. El bit M (modificación) se debe incluir porque para la tabla de páginas se utiliza una estrategia de post-escritura. Finalmente, el bit U (uso) se debe incluir para implementar un reemplazo pseudo-LRU.

b)

		Vía 0							Vía 1						
		V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)
0	0	0	-	-	-	-	-	-	0	-	-	-	-	-	-
1	0	0	-	-	-	-	-	-	0	-	-	-	-	-	-

Tenemos 2 conjuntos y 2 vías, lo que nos permite almacenar en el TLB hasta 4 entradas distintas de la tabla de páginas. Por lo tanto, de los 19 bits que conforman el NPV, necesitamos 1 bit para el índice y nos quedan 18 bits para la etiqueta.

Tamaño = $2 * 2 * (3+18+2+13)$ bits = 144 bits

El bit V (validez) siempre se debe incluir para saber si el contenido de una entrada es válido o no. El bit M (modificación) se debe incluir porque para el TLB se utiliza una estrategia de post-escritura. Finalmente, el bit U (uso) se debe incluir para implementar un reemplazo LRU. Generalmente, un único bit de uso no es suficiente para implementar un reemplazo LRU puro. Sin embargo, en este caso es más que suficiente porque cada conjunto tiene sólo dos entradas.

- c) Para resolver esta parte, debemos tener en cuenta que las direcciones virtuales son de 30 bits, de los que 19 corresponden al número de página virtual (NPV) y 11 al desplazamiento. En el caso del

TLB, de los 19 bits que corresponden al NPV, el bit menos significativo es el índice que selecciona el conjunto del TLB y los 18 bits restantes son la etiqueta.

- Lectura de la dirección $0x810 = 00\ 0000\ 0000\ 0000\ 0000\ 1000\ 0001\ 0000)_2$.

NPV: $00\ 0000\ 0000\ 0000\ 0000\ 1)_2 = 1$.

Desplazamiento: $000\ 0001\ 0000)_2$.

Acierto/fallo de TLB: fallo (el TLB está inicialmente vacío).

Acierto/fallo de página: fallo (la tabla de páginas está inicialmente vacía). El S.O. lee de disco la página virtual 1 que carga en la página física 0 (la memoria está libre). Se actualiza la tabla de páginas:

	V	M	U	NPF (13 bits)
0	0	–	–	–
1	1	0	0	0
2	0	–	–	–
...
$2^{19} - 2$	0	–	–	–
$2^{19} - 1$	0	–	–	–

Conjunto del TLB: 1 (el bit menos significativos del NPV) y queda de la siguiente manera:

Vía 0							Vía 1							
	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)
0	0	–	–	–	–	–	–	0	–	–	–	–	–	–
1	1	1	1	0	0	1	0	0	–	–	–	–	–	–

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

NPV: 0.

Dirección física resultante: $0000\ 0000\ 0000\ 0000\ 0001\ 0000)_2 = 0x10$.

- Lectura de la dirección $0x1010 = 00\ 0000\ 0000\ 0000\ 0001\ 0000\ 0001\ 0000)_2$.

NPV: $00\ 0000\ 0000\ 0000\ 0001\ 0)_2 = 2$.

Desplazamiento: $000\ 0001\ 0000)_2$.

Acierto/fallo de TLB: fallo; las dos vías del conjunto 0 del TLB están vacías.

Acierto/fallo de página: fallo; la entrada 2 de la tabla de páginas está vacía. El S.O. lee de disco la página virtual 2 que almacena en la página física 1. Se actualiza la tabla de páginas:

	V	M	U	NPF (13 bits)
0	0	–	–	–
1	1	0	0	0
2	1	0	0	1
...
$2^{19} - 2$	0	–	–	–
$2^{19} - 1$	0	–	–	–

Conjunto del TLB: 0 (el bit menos significativos del NPV, quedando la etiqueta con valor 1) y queda de la siguiente manera:

Vía 0							Vía 1							
	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)
0	1	1	1	1	0	1	1	0	–	–	–	–	–	–
1	1	1	1	0	0	1	0	0	–	–	–	–	–	–

Ya que el bit de uso U' se pone a 1, el bit M del TLB debe ponerse también a 1.

NPF: 1.

Dirección física resultante: $0000\ 0000\ 0000\ 1000\ 0001\ 0000)_2 = 0x810$.

- Escritura en la dirección $0x800 = 00\ 0000\ 0000\ 0000\ 0000\ 1000\ 0000\ 0000)_2$.

NPV: $00\ 0000\ 0000\ 0000\ 0000\ 1)_2 = 1$.

Desplazamiento: $000\ 0000\ 0000)_2$.

Acierto/fallo de TLB: la información para la traducción se encuentra en el TLB ya que en el segundo conjunto del TLB hay una entrada cuya etiqueta coincide con la obtenida del NPV. Si tenemos un acierto de TLB, obligatoriamente también hay un acierto en la tabla de páginas. Lo único que hay que hacer es actualizar el TLB para indicar que la página se ha modificado, pero no hay que actualizar la tabla de páginas pues el TLB tiene política de post-escritura:

		Vía 0							Vía 1						
		V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)
0	1	1	1	1	1	0	1	1	0	-	-	-	-	-	-
1	1	1	1	1	0	1	1	0	0	-	-	-	-	-	-

Acierto/fallo de página: acierto y no se actualiza.

NPF: 0.

Dirección física resultante: $0000\ 0000\ 0000\ 0000\ 0000\ 0000)_2 = 0$.

- Escritura en la dirección $0x2000 = 00\ 0000\ 0000\ 0000\ 0010\ 0000\ 0000\ 0000)_2$.

NPV: $00\ 0000\ 0000\ 0000\ 0010\ 0)_2 = 4$.

Desplazamiento: $000\ 0000\ 0000)_2$.

Acierto/fallo de TLB: se produce un fallo de TLB, pues la entrada en el primer conjunto tiene una etiqueta distinta de 2.

Acierto/fallo de página: también se produce un fallo de página al no encontrarse la información en la tabla de páginas. El S.O. lee de disco la página virtual 4 que deja en la página física 2 (la siguiente página libre en memoria). Se actualiza la tabla de páginas:

	V	M	U	NPF (13 bits)
0	0	-	-	-
1	1	0	0	0
2	1	0	0	1
3	0	-	-	-
4	1	0	0	2
5	0	-	-	-
...
$2^{19} - 2$	0	-	-	-
$2^{19} - 1$	0	-	-	-

Conjunto del TLB: 0 (el bit menos significativos del NPV, quedando la etiqueta con valor 2). Al ser el TLB asociativo de 2 vías y al haber en el primer conjunto todavía un bloque libre, no es necesario expulsar nada del TLB. Lo que sí es necesario es poner a 0 el bit de uso del primer bloque del conjunto 0. Esto nos permite saber qué bloque se utilizó hace más tiempo para implementar así una política de reemplazo LRU. El TLB queda de la siguiente manera:

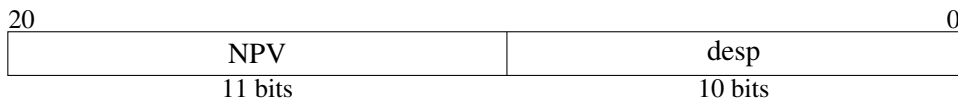
		Vía 0							Vía 1						
		V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)	V	M	U	Etiqueta (18 bits)	M'	U'	NPF (13 bits)
0	1	1	0	1	1	0	1	1	1	1	1	2	1	1	2
1	1	1	1	1	0	1	1	0	0	-	-	-	-	-	-

NPF: 2.

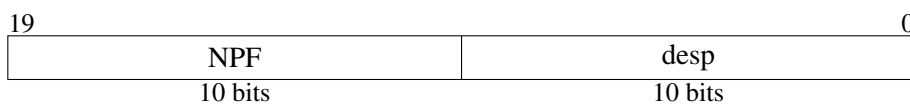
Dirección física resultante: $0000\ 0000\ 0001\ 0000\ 0000\ 0000)_2 = 0x1000$.

12. a) Harán falta 10 bits para codificar el desplazamiento dentro de cada página (desp) de 1 KB, y dado que hay 2048 entradas en la tabla de páginas se necesitarán 11 bits para el número de página virtual (NPV). Por tanto, las direcciones virtuales tendrán un total de 21 bits. Asimismo, debido a que el tamaño de la memoria física es de 1 MB, las direcciones físicas tendrán un total de 20 bits, de los cuales 10 codificarán el desplazamiento y 1 el número de página física (NPF). Por tanto:

Dirección virtual:



Dirección física:



- b) Cada entrada de la tabla de páginas necesitará un bit de modificación (M) para permitir que la política de postescritura, y un bit de uso (U) para permitir la política de reemplazo LRU, además del bit de validez (V).

Tabla de páginas (2048 entradas):

V	M	U	NPF (10 bits)
0	0	0	–
1	0	0	0
0	0	0	–
1	0	0	1
0	0	0	–
1	0	0	2
0	0	0	–
1	0	0	3
0	0	0	–
0	0	0	–
0	0	0	–
... ..			
0	0	0	–

Cada entrada de la tabla de páginas tiene (al menos) 13 bits, por lo que en total se necesitarán 26624 bits. Si se redondea el tamaño de cada entrada a 16 bits, se necesitarán 4 KB.

- c) Debido a que el TLB tiene 4 conjuntos, los 2 bits de menor peso del NPV se utilizarán para elegir la posición en la caché, mientras que los 9 restantes formarán la etiqueta.

Cada entrada necesitará un bit de modificación (M) para permitir que la política de postescritura, y un bit de uso (U) para permitir la política de reemplazo LRU, además del bit de validez (V).

Conjunto 0 del TLB:

V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
0	0	0	–	0	0	–
0	0	0	–	0	0	–
0	0	0	–	0	0	–
0	0	0	–	0	0	–

Conjunto 1 del TLB:

V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
0	0	0	—	0	0	—
0	0	0	—	0	0	—
0	0	0	—	0	0	—
0	0	0	—	0	0	—

M' y U' representan los bits de modificación y uso de la entrada de la tabla de páginas.

Por tanto, el tamaño total del TLB será de $2 \times 4 \text{ times}(1 + 1 + 1 + 9 + 1 + 1 + 10) = 192 \text{ bits}$.

- d) ■ Lectura a la dirección 0x124.

NPV: 0.

Desplazamiento: 0x124.

Acierto/fallo: Fallo.

NPF: 35 (estaba libre).

Dirección física resultante: 0x8d24.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	0	1	35

- Lectura de la dirección 0x422.

NPV: 1.

Desplazamiento: 0x22.

Acierto/fallo: Acierto.

NPF: 0.

Dirección física resultante: 0x22.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	0	1	0

- Escritura a la dirección 0x124.

NPV: 0.

Desplazamiento: 0x124.

Acierto/fallo: Acierto.

NPF: 35.

Dirección física resultante: 0x8d24.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	1	1	35

- Escritura a la dirección 0x1418.

NPV: 5.

Desplazamiento: 0x18.

Acierto/fallo: Acierto.

NPF: 2.

Dirección física resultante: 0x818.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	1	1	2

- Lectura a la dirección 0x1428.

NPV: 5.

Desplazamiento: 0x28.

Acierto/fallo: Acierto.

NPF: 2.

Dirección física resultante: 0x818.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	1	1	2

- Escritura a la dirección 0x1d18.

NPV: 7.

Desplazamiento: 0x118.

Acierto/fallo: Acierto.

NPF: 3.

Dirección física resultante: 0xd18.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	1	1	3

- Lectura a la dirección 0x1620.

NPV: 5.

Desplazamiento: 0x220.

Acierto/fallo: Acierto.

NPF: 2.

Dirección física resultante: 0xa20.

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	1	1	2

- e) ■ Lectura a la dirección 0x124.

NPV: 0.

Conjunto de TLB: 0.

Etiqueta de TLB: 0.

Desplazamiento: 0x124.

Acierto/fallo de TLB: Fallo.

Acierto/fallo de página: Fallo.

NPF: 35 (estaba libre).

Dirección física resultante: 0x8d24.

Entrada en el TLB:

V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
1	1	1	0	0	1	35

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	0	0	35

- Lectura a la dirección 0x422.

NPV: 1.

Conjunto de TLB: 1.

Etiqueta de TLB: 0.

Desplazamiento: 0x22.

Acierto/fallo de TLB: Fallo.

Acierto/fallo de página: Acierto.

NPF: 0.

Dirección física resultante: 0x22.

Entrada en el TLB:	V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
	1	1	1	0	0	1	0

Entrada en la tabla de páginas:	V	M	U	NPF (10 bits)
	1	0	0	0

- Escritura a la dirección 0x124.

NPV: 0.

Conjunto de TLB: 0.

Etiqueta de TLB: 0.

Desplazamiento: 0x124.

Acierto/fallo de TLB: Acierto.

Acierto/fallo de página: No se realiza acceso.

NPF: 35.

Dirección física resultante: 0x8d24.

Entrada en el TLB:	V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
	1	1	1	0	1	1	35

Entrada en la tabla de páginas:	V	M	U	NPF (10 bits)
	1	0	0	35

- Escritura a la dirección 0x1418.

NPV: 5.

Conjunto de TLB: 1.

Etiqueta de TLB: 1.

Desplazamiento: 0x18.

Acierto/fallo de TLB: Fallo.

Acierto/fallo de página: Acierto.

NPF: 2.

Dirección física resultante: 0x818.

Entrada en el TLB:	V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
	1	1	1	1	1	1	2

Entrada en la tabla de páginas:	V	M	U	NPF (10 bits)
	1	0	0	2

- Lectura a la dirección 0x1428.

NPV: 5.

Conjunto de TLB: 1.

Etiqueta de TLB: 1.

Desplazamiento: 0x28.

Acierto/fallo de TLB: Acierto.

Acierto/fallo de página: No se realiza acceso.

NPF: 2.

Dirección física resultante: 0x828.

Entrada en el TLB:

V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
1	1	1	1	1	1	2

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	0	0	2

- Escritura a la dirección 0x1d18.

NPV: 7.

Conjunto de TLB: 3.

Etiqueta de TLB: 1.

Desplazamiento: 0x118.

Acierto/fallo de TLB: Fallo.

Acierto/fallo de página: Acierto.

NPF: 3.

Dirección física resultante: 0xd18.

Entrada en el TLB:

V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
1	1	1	1	1	1	3

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	0	0	3

- Lectura a la dirección 0x1620.

NPV: 5.

Conjunto de TLB: 1.

Etiqueta de TLB: 1.

Desplazamiento: 0x220.

Acierto/fallo de TLB: Acierto.

Acierto/fallo de página: No se realiza acceso.

NPF: 2.

Dirección física resultante: 0xa20.

Entrada en el TLB:

V	M	U	Etiqueta (9 bits)	M'	U'	NPF (10 bits)
1	1	1	1	1	1	2

Entrada en la tabla de páginas:

V	M	U	NPF (10 bits)
1	0	0	2

- f) En los 7 accesos totales, se producen 4 fallos de TLB, de los cuales 1 produce además 1 fallo de página. Por tanto, se realizarán un total de 11 accesos a memoria (7 a datos + 4 a la tabla de páginas) y 1 acceso a disco.