Universidad del País Vasco
Facultad de Informática

Departamento de Arquitectura y Tecnología de Computadores

Arquitectura de Computadores I

Sistema de memoria 1 (Solución): Paginación + módulos entrelazados

El sistema de memoria de un computador utiliza palabras de 4 bytes y la unidad de direccionamiento es el byte. Las características del sistema de memoria son las siguientes:

- **Memoria virtual:** memoria paginada de 2 MB (megabytes), con páginas de 256 bytes. Para la traducción de las direcciones se utiliza un TLB. El tiempo de acceso al TLB es de 20 ciclos en caso de fallo y de un ciclo en caso de acierto. Al principio el TLB está vacío.
- **Memoria principal:** es de 256 kB y está formada por 4 módulos entrelazados. El tiempo de acceso es de 10 ciclos (1 ciclo desde el buffer de entrelazado).

En este computador se ejecuta el siguiente programa:

```
for (i=0;i<252;i++)
        B[i]=(A[i]*A[i+4])+C[0];
```

```
                movi r1,#1520
                movi r2,#0
                movi r3,#251
bucle:          load r5,A[r2]
                load r6,A[r2+16]
                mul r5,r5,r6
                load r10,[r1]
                add r6,r6,r10
                store r6,B[r2]
                addi r2,r2,#4
                subi r3,r3,#1
                bge r3,bucle
```

El programa está almacenado a partir de la dirección lógica 320. El vector A comienza en la dirección lógica 2048, el vector B, en la dirección lógica 512 y el vector C, en la dirección lógica 1520. Tanto las instrucciones como los elementos de los vectores ocupan una palabra.

a) Dibuja la estructura del sistema de memoria, indicando claramente para qué se utilizan los distintos campos de las direcciones, tanto para la memoria virtual como para la memoria principal. Expresa el número de entradas en la tabla de páginas y el tamaño de dichas entradas. En cuanto al TLB, ¿cuál es el tamaño de una de sus entradas?

b) Traduce todas las direcciones que se crean en la primera pasada del bucle del programa y calcula el tiempo necesario para acceder a cada dirección en el sistema de memoria. Haz uso de una tabla para explicar claramente todos los pasos que hagas.

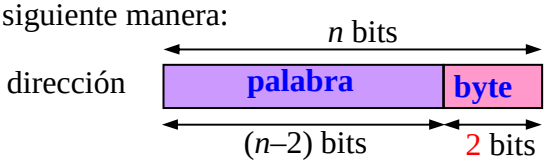
c) Calcula el tiempo de acceso al sistema de memoria (tiempo necesario para traducir las direcciones + tiempo necesario para acceder a la memoria principal) para la ejecución de todo el programa.

Información de la tabla de páginas:

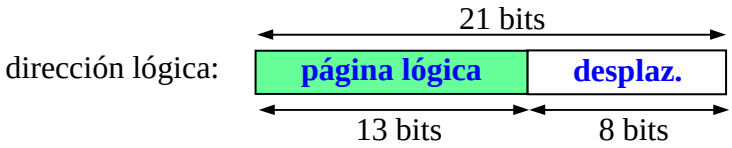
Página lógica:	1	8	5	2	9	3	10	4	11	....
Página física:	2	0	5	1	15	20	17	10	3	....

Solución

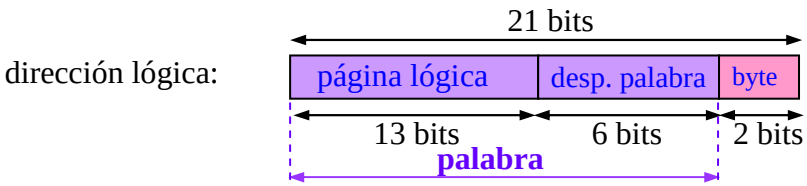
En primer lugar hay que analizar en profundidad las características del sistema de memoria en el enunciado, para inferir toda la información necesaria. En este caso, debemos tener en cuenta que este sistema de memoria utiliza **palabras de 4 bytes** y que la **unidad de direccionamiento es el byte**, es decir, que es posible conseguir o distinguir los bytes de la palabra uno a uno. Por ello, los bits de la dirección estarán divididos en dos campos: por un lado, los bits que identifican la **palabra** a la que se quiere acceder y, por otro, los bits que identifican el **byte concreto** al que que se quiere acceder dentro de esa palabra. Dado que hay 4 bytes en la palabra, se necesitan 2 bits para identificar cada byte (dado que $2^2 = 4$). Debemos seguir leyendo el enunciado para saber el tamaño de la memoria y determinar el número de bits para representar las direcciones, pero de momento si suponemos que se necesitan *n* bits, el esquema de las direcciones queda de la siguiente manera:



Sigamos leyendo el enunciado. Tras lo anterior se indican las características de la **memoria virtual**. La memoria virtual es de 2 MB (megabytes) y se nos indica que es paginada con páginas de 256 bytes. De toda esa información podemos extraer muchas conclusiones sobre la estructura de las **direcciones lógicas**. Por un lado, las direcciones lógicas al byte necesitan 21 bits, ya que $2 \text{ MB} = 2 \times 2^{20} = 2^{21}$. Por otro lado, dado que la memoria virtual está paginada, la dirección lógica tienen dos campos: los bits que indican la **página lógica** y los bits que indican el **desplazamiento** que presenta el byte dentro de la página lógica. Dado que las páginas lógicas son de 256 bytes, se necesitan 8 bits para indicar los desplazamientos de los bytes ($256 = 2^8$). En consecuencia, tenemos $21 - 8 = 13$ bits para representar las páginas lógicas (por lo que un programa puede tener $2^{13} = 8192$ páginas). El esquema de la dirección lógica queda como sigue:

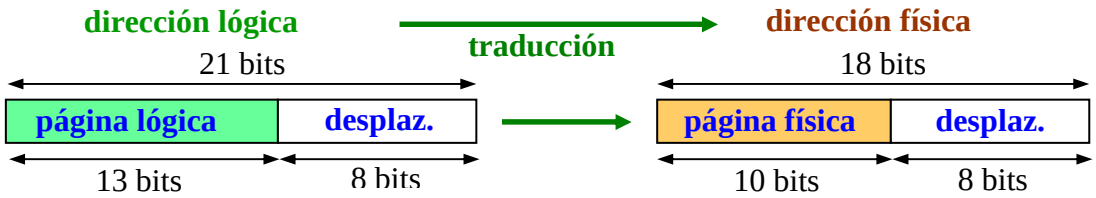


Aunque no se ha dicho nada, dicha dirección lógica es al byte, ya que ésta es la unidad de direccionamiento. Por tanto, si consideramos lo dicho al principio de la resolución, esto es, que para indicar el byte de la palabra se necesitan 2 bits y que el resto identifican la palabra, en este caso se pueden distinguir también estos tres campos: página lógica, desplazamiento de la palabra dentro de la página (6 bits) y byte (2 bits).



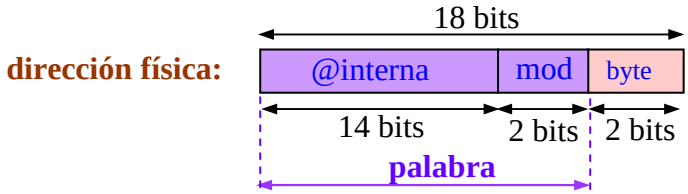
De todas formas, para resolver los ejercicios descartaremos esta última opción, ya que distinguir las palabras y los bytes en las direcciones lógicas no aporta información especial.

En cuanto a la **memoria principal**, sabemos que es de 256 kB (kilobytes). Por tanto, para expresar la **dirección física** se necesitan 18 bits ($256\ kB = 2^8 \times 2^{10} = 2^{18}$). Por ello, en el proceso de conversión de direcciones lógicas a físicas, 21 bits se convierten en 18 bits. Dado que las páginas lógicas y físicas son del mismo tamaño, en ambos casos el desplazamiento será el mismo, y lo que se reduce es la cantidad de bits para expresar la página física, que será de 10 bits (18 – 8). Esto es, la memoria física tendrá $2^{10} = 1024$ páginas físicas. La traducción se hará de la siguiente manera:

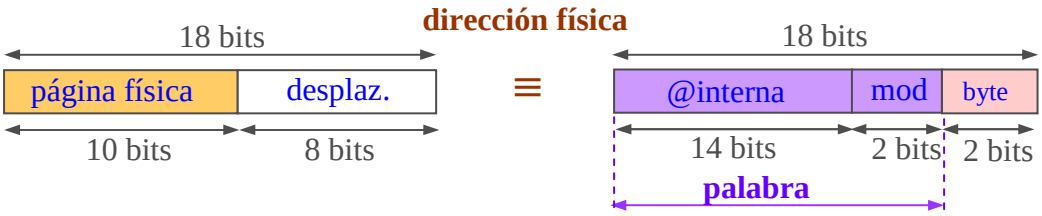


Sabemos que el sistema dispone de un TLB para hacer la traducción y más adelante dibujaremos el esquema hardware para hacer la traducción – apartado (a).

La dirección física que se obtiene tras el proceso de traducción va a memoria principal para conseguir la palabra deseada. En este caso sí que distinguiremos la palabra y el byte, ya que gracias a esa distinción podremos hacer más fáciles algunos de los cálculos. Por tanto, como hemos dicho al principio, de los 18 bits necesarios para expresar las direcciones físicas los 2 bits de menos peso indican el byte concreto dentro de la palabra y, por tanto, los otros 16 bits expresan la palabra. Por otro lado, hay que distinguir dos campos dentro del campo de la palabra: por un lado, dado que la memoria principal está compuesta de 4 módulos entrelazados, necesitamos el campo que indique en qué módulo está la palabra (**mod**, de 2 bits, dado que $2^2 = 4$ y 4 es el número de módulos) y, por otro, el campo que nos indica en qué posición concreta está la palabra dentro de ese módulo (**@interna**, de 16 – 2 = 14 bits). La estructura de la dirección física queda así:



Nota de aclaración: Vistas las dos figuras anteriores, parece que hay dos direcciones físicas diferentes. Por un lado, la que se obtiene del proceso de traducción y por otro la que va a memoria principal. Pero, como se ha dicho, ambas son la misma, es decir, los 18 bits que se obtienen del proceso de traducción son los que van a memoria principal; lo único que cambia es la interpretación que se da a los bits, desde el punto de vista de la traducción o de la memoria principal, y si forman parte de un campo o de otro, sin más. Por tanto, ambas direcciones son completamente equivalentes.



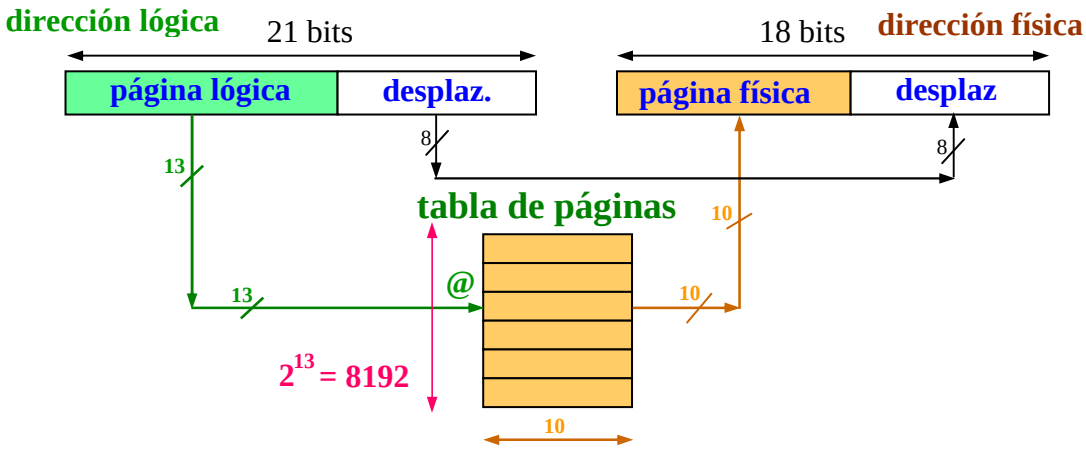
Tras inferir toda esta información de las primeras líneas del enunciado, lo mejor que podemos hacer a partir de ahora es ir por apartados.

(a) Debemos dibujar la estructura del sistema de memoria, mostrando cómo se utilizan los bits de los campos de las direcciones lógicas y físicas en casa caso.

En cuanto a la traducción, sabemos que los bits que representan la página lógica son los que hay que enviar a las entradas de la tabla de páginas, para ahí analizar en qué página física ha ubicado el sistema dicha página lógica al cargarla en memoria principal. En cuanto al desplazamiento, como es el mismo en las páginas lógicas y en las físicas, los bits que lo representan serán los mismos en las direcciones lógicas y físicas.

Como la tabla de páginas está en memoria, los 13 bits que se reciben de la dirección lógica son los que utilizan para el acceso a la tabla, por loq ue la tabla de páginas tiene 2^{13} = 8192 entradas (tantas como páginas lógicas, ya que a cada página lógica le corresponde una entrada en la tabla de páginas). Por otro lado, en cada entrada de la tabla de páginas figuran los 10 bits necesarios para expresar la página física.

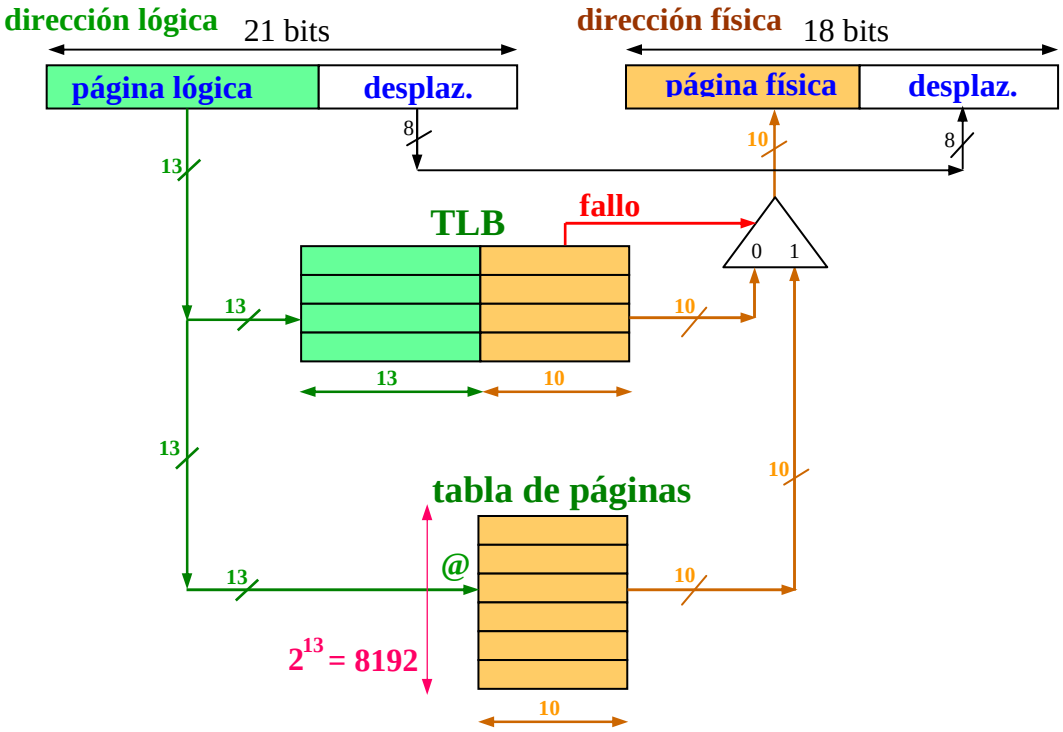
En resumen, así queda el esquema del hardware encargado de hacer la traducción "dirección lógica → dirección física":



Pero por lo dicho en el enunciado, para acelerar las traducciones en este sistema hay un TLB, ya que por sus características se lee más rápido del TLB que de la tabla de páginas. Así, cuando quiere traducirse una dirección lógica y obtener su dirección física correspondiente, primero se hace la búsqueda en el TLB: si es la primera vez que aparece el identificador de la página lógica en la lista de direcciones generadas por el procesador, entonces esa información no está en el TLB y por ello se produce un fallo; en este caso, el sistema debe mirar en la tabla de páginas en qué página física está cargada la página lógica y, además, escribirá en el TLB la página física que corresponde a esa página lógica. De esta forma, en las próximas búsquedas esa referencia de la página lógica estará ya en el TLB, por lo que, como resultado de la búsqueda, se producirá un acierto y el TLB proporcionará los bits que corresponden a la página física que guarda esa página lógica, en un tiempo muy rápido.

Como el TLB es una memoria asociativa, sus búsquedas se realizan por contenido. Por ello, los 13 bits que recibe no forman la dirección, como en el caso de la tabla de páginas, sino que parte del contenido. Así, en cada posición del TLB una parte la forman los 13 bits que identifican las páginas lógicas y otra parte los 10 bits que identifican la página física correspondiente a esa página lógica. Es esa segunda parte la que se lleva a la dirección física como salida del TLB. Por ello, cada entrada del TLB tiene por lo menos 23 bits (13 + 10). Pero la cantidad de entradas del TLB no tiene que ver con el número de bits que recibe; en principio, el TLB puede tener cualquier número de entradas y eso lo decide el diseñador del sistema.

Así queda el esquema del hardware con TLB:



Una vez conseguida la dirección física analizaremos la estructura de memoria principal. Al ser la memoria principal de 256 kB (2^{18} byte), y la palabra de 4 (2^2) bytes, podemos asegurar que en memoria principal entran 65536 (2^{16}) palabras:

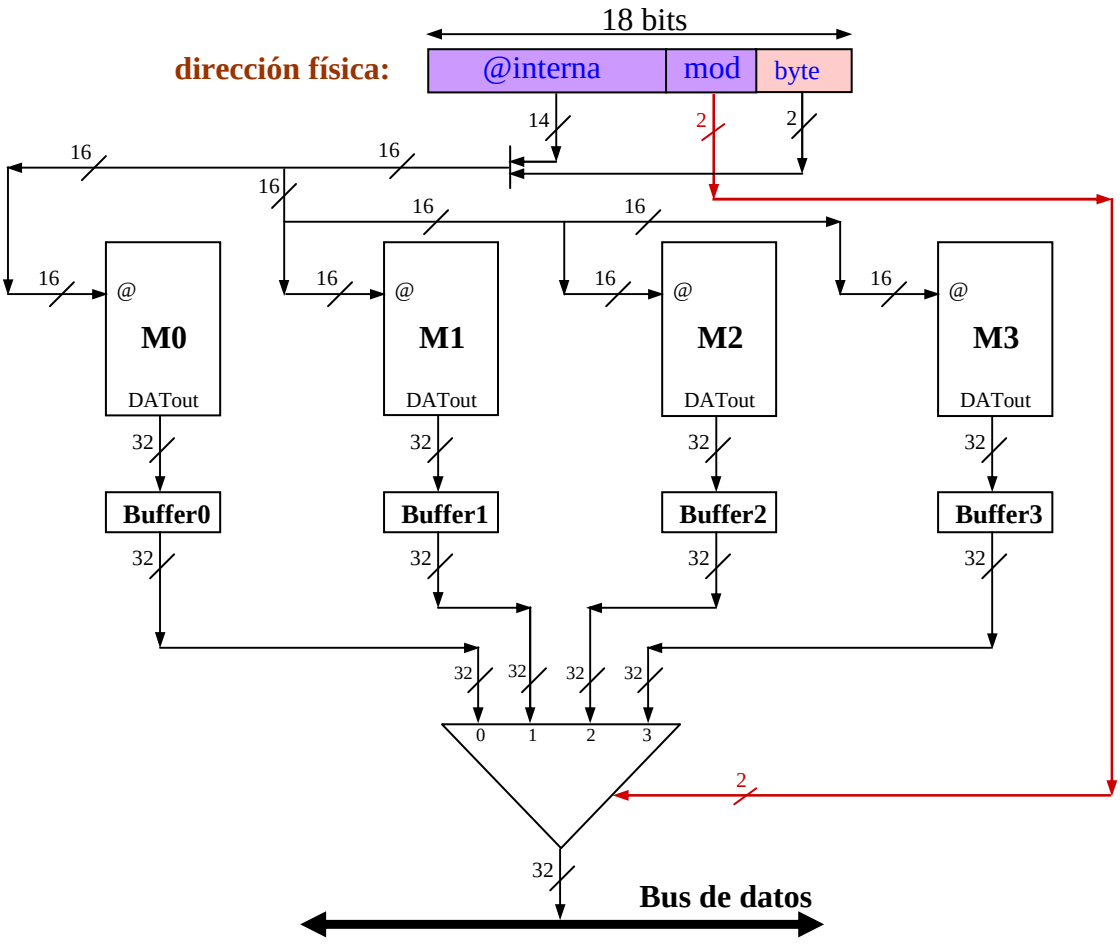
$$\frac{256\text{ kB}}{4\text{ bytes/palabra}} = \frac{2^{18}\text{ bytes}}{2^2\text{ bytes/palabra}} = 2^{16}\text{ palabras} = 65536\text{ palabras}$$

Del mismo modo, como la memoria principal la componen 4 (2^2) módulos, cada módulo es de 64 kB, o 16384 (2^{14}) palabras:

$$\frac{256\text{ kB}}{4\text{ módulos}} = 64\text{ kB/módulo}$$

$$\frac{2^{16}\text{ palabras}}{2^2\text{ módulos}} = 2^{14}\text{ palabras/módulo} = 16384\text{ palabras/módulo}$$

Así queda el esquema hardware de la memoria principal:



Atendiendo al esquema de la memoria en la figura, se puede extraer el comportamiento de los módulos entrelazados: se leen de todos los módulos a la vez las palabras con la misma dirección interna y se almacenan en los buffers de entrelazado. Gracias a ello, si en los siguientes accesos se quiere conseguir alguna de esas palabras, no es necesario acceder a los módulos, ya que dicha palabra está en los buffers, y el tiempo de acceso se reduce notablemente, tal y como veremos en el siguiente apartado.

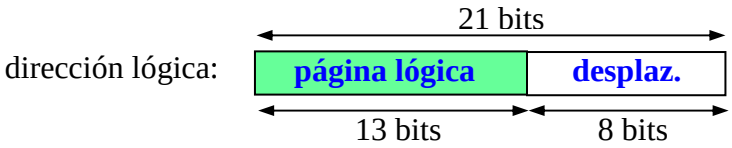
(b) Al ejecutar el programa que nos indican, debemos calcular cómo se traducen las direcciones generadas en la primera pasada del bucle, cuánto tiempo se necesita en hacer la traducción y cuánto tiempo se necesita para acceder a las posiciones de memoria principal. Para hacer todo esto de un modo ordenado, iremos paso a paso y resumiremos en una tabla los resultados de los pasos.

Para ello, debemos tener en cuenta las direcciones lógicas de instrucciones y datos, ya que esos son los valores que el procesador pondrá en el bus de direcciones para acceder a la instrucción y el dato actual. Según lo dicho en el enunciado, el programa (es decir, las instrucciones) está almacenado a partir de la dirección lógica 320. El vector A comienza en la dirección lógica 2048 (es decir, en esa dirección está el elemento A[0], y a partir de ahí los demás elementos, cada

uno ocupando 4 bytes, ya que su tamaño es de una palabra, es decir, 4 bytes); el vector B, en la dirección lógica 512 y el vector C, en la dirección lógica 1520.

Analicemos ahora las instrucciones del programa una a una, para ver qué direcciones lógicas generará el procesador en cada momento y cómo realizará todo el proceso de acceso a memoria hasta conseguir la instrucción o el dato.

La primera instrucción, `movi r1,#1520`, según lo que indica el enunciado, está en la dirección lógica 320, por tanto, ése es el valor que pondrá el procesador cuando quiera acceder a dicha instrucción. Sabemos que el primer paso para acceder a memoria principal es que el hardware haga la traducción de esa dirección lógica con el fin de obtener su dirección física correspondiente. Para ello, el hardware distinguirá los dos campos en que se divide la dirección lógica: página lógica y desplazamiento.



Por ello, si representásemos el número 320 en binario, los 13 bits de más peso indicarían la página lógica y, por el contrario, los 8 bits de menos peso, el desplazamiento. Pero como representar los números decimales en binario es un trabajo pesado, todos los cálculos los realizaremos en decimal. Para ello, debemos resolver el siguiente problema: ¿cómo distinguir, en decimal, los campos compuestos por bits? La respuesta es bien sencilla, ya que sólo hay que recordar que desplazar un número en binario una posición a la derecha es equivalente a dividir entre dos, es decir, por cada posición que se quiere desplazar a la derecha hay que hacer la operación “entre dos” y que los bits que se pierden por la derecha forman el resto de la división.

Por ejemplo, supongamos que tenemos el número decimal 38 representado en 6 bits y que debemos distinguir ahí dos campos: por un lado, los 2 bits de más peso representan una información (en este momento nos da igual si es la página lógica, la palabra, el módulo o lo que sea) y los 4 bits de menos peso otra cuestión (por ejemplo el desplazamiento):

$$\text{Número} = 38_{(10} = 100110$$

Pongamos que desplazamos el número 4 veces a la derecha metiendo 0s por la izquierda para mantener la representación en 6 bits. Éste es el resultado obtenido: `000010`, y los bits que se han perdido por la derecha: `0110`. Estos dos resultados en decimal son los números 2 y 6, respectivamente.

Dado que el número inicial lo hemos desplazado 4 veces a la derecha, hemos hecho “entre 2” cuatro veces, es decir (((38/2)/2)/2)/2, o $38/16 = 38/2^4$. Si hacemos esa división vemos que el resultado no es entero, ya que 38/16 = 2,375.

Si tomamos la parte entera vemos que sale 2, y lo que falta, $0{,}375 \times 16 = 6$, es el resto. Y precisamente esos son los valores que se consiguen desplazando 38 en binario.

Matemáticamente, se representa así:

El valor del campo que componen los 2 bits de más peso: $38 \texttt{ div } 2^4 = 2$

El valor del campo que componen los 4 bits de menos peso: $38 \texttt{ mod } 2^4 = 6$

Por ello, si tuviéramos que representar el número 320 en binario, para calcular el valor de la página lógica desplazaríamos el número 8 veces a la derecha. Esto es, debemos hacer $320 \texttt{ div } 2^8$ para calcular la página lógica y el desplazamiento será el resto de la división: $320 \texttt{ mod } 2^8$.

@lógica = 320 → página lógica = $320 \texttt{ div } 2^8 = 320 \texttt{ div } 256 = 1$

desplazamiento = $320 \texttt{ mod } 2^8 = 320 \texttt{ mod } 256 = 64$

Se pueden inferir las ecuaciones matemáticas generales, que aplicaremos a todas las direcciones lógicas generadas por el procesador:

página lógica = @lógica `div` tamaño de página en bytes

desplazamiento = @lógica `mod` tamaño de página en bytes

Para resolver el ejercicio de forma ordenada, pondremos en una tabla los resultados de todos los pasos. Empecemos ahora mismo:

Instrucción	Dirección lógica *@lógica*	página lógica *p.l.*	desplazamiento *d*
`movi r1,#1520`	320	1	64

Sigamos con el proceso de traducción de la primera instrucción. Una vez logrados los campos de la dirección lógica, podemos ver que el TLB buscará el identificador de la página lógica (1) entre sus contenidos. Si lo encuentra, se dice que hay acierto, y el TLB ofrecerá en un ciclo como salida la página física correspondiente a esa página lógica. Si no lo encuentra, se dice que hay fallo y ello quiere decir que la página lógica que está buscando todavía no se ha referenciado, es decir, que ésta es la primera aparición de esta página lógica entre las direcciones lógicas generadas por el procesador; por ello, se necesitarán 20 ciclos para buscarla en la tabla de páginas y obtener su página física correspondiente. En el caso de la primera instrucción, el resultado de la búsqueda será fallo, porque el TLB está inicialmente vacío; por ello, se necesitan 20 ciclos para hacer la traducción de la primera instrucción ($t_{trad.}$). Miraremos en la tabla de páginas para saber en qué página física ha cargado esa página lógica el sistema. En este caso, la página lógica 1 está cargada en la página física (*p.f.*) 2.

Instrucción	*@l*	*p.l.*	*d*	$t_{trad.}$	*p.f.*	*d*
`movi r1,#1520`	320	1	64	20	2	64

Cuando se conoce la página física, hay que conseguir la dirección física. Para ello, sabemos lo que hay que hacer para completar la dirección física en binario: poner los bits que componen la página física a la izquierda y añadirles a la derecha los bits del desplazamiento. Pero, ¿cómo hacerlo en decimal? Poner los bits que indican la página física a la izquierda de los 8 bits que componen el desplazamiento quiere decir que los desplazaremos 8 posiciones a la izquierda;

por tanto, la página física la multiplicaremos por 2^8. Poner los bits que componen el desplazamiento a la derecha de los anteriores quiere decir que debemos sumar su valor.

Por si acaso, para despejar dudas, recordaremos el ejemplo del número 38, pero desde el punto de vista contrario. Es decir, $38 = 2 \times 2^4 + 6$.

Así pues, para calcular la dirección física (@*f*) que corresponde a 320 haremos:

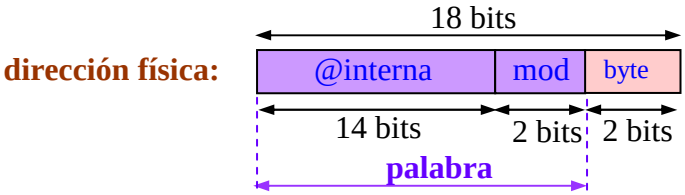
$$@\text{física} = 2 \times 2^8 + 64 = 576$$

Y también podemos inferir la ecuación general:

@física = página física × tamaño de página en bytes + desplazamiento

Instrucción	@*l*	*p.l.*	*d*	$t_{trad.}$	*p.f.*	*d*	@*f*
`movi r1,#1520`	320	1	64	20	2	64	576

Una vez conocida la dirección física, llega el momento de acceder a memoria principal. Para ello hay que tener en cuenta en qué módulo y en qué dirección interna dentro de dicho módulo está dicha dirección física. Vamos a recordar cuáles son los campos de bits de la dirección física:



En este caso y dado que hay tres campos, los distinguiremos uno a uno. Primero, "eliminaremos" el campo de más a la derecha (los 2 bits que indican cuál es byte concreto dentro de la palabra); es decir, antes que nada calcularemos qué palabra identifica esa dirección física. Sabemos cómo hacer esto en binario: desplazando la dirección física dos posiciones a la derecha; además, sabemos cómo hacerlo en decimal: mediante la operación palabra = dirección física `div` 2^2. El resto (dirección física `mod` 2^2) será el byte concreto dentro de la palabra que buscamos. En general, al buscar palabras enteras el resto será 0.

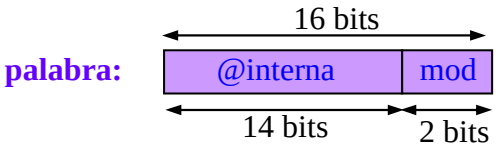
En este caso, por tanto: palabra = 576 div 2^2 = 576 div 4 = 144

Como antes, inferimos la ecuación general:

palabra = @física `div` tamaño palabra en bytes

Instrucción	@*l*	*p.l.*	*d*	$t_{trad.}$	*p.f.*	*d*	@*f*	palabra
`movi r1,#1520`	320	1	64	20	2	64	576	144

Tras calcular la palabra, debemos distinguir sus dos campos, el módulo y la dirección interna dentro del módulo (@interna):



Realizando divisiones por desplazamientos, esto es lo que inferimos:

@interna = palabra `div` 2^2 = palabra `div` 4 = 144 `div` 4 = 36

mod = palabra `mod` 2^2 = palabra `mod` 4 = 144 `mod` 4 = 0

De aquí también inferimos la ecuación general:

@interna = palabra `div` número de módulos
mod = palabra `mod` número de módulos

Instrucción	@ℓ	$p.\ell.$	d	$t_{trad.}$	$p.f.$	d	@f	palabra	interna	mod
`movi r1,#1520`	320	1	64	20	2	64	576	144	36	0

Para acabar con todo el proceso, sólo falta expresar el tiempo necesario para conseguir esa palabra de memoria principal (t_{acceso}). Para ello, debemos tener en cuenta que el tiempo de acceso de memoria principal es de 10 ciclos y, en caso de que la palabra esté ya en el buffer de entrelazado, tan sólo un ciclo. ¿Cuándo estará la palabra en el buffer de entrelazado? La respuesta es muy lógica: cuando en el acceso anterior se haya accedido a la misma dirección interna del módulo, ya que se accede a todos los módulos a la vez. En el caso de la primera instrucción, por tanto, el acceso será de 10 ciclos, ya que es la primera dirección a la que se accede en la ejecución del programa. Por ello, en lo que se refiere a la primera instrucción, así queda la tabla:

@ℓ	$p.\ell.$	d	$t_{trad.}$	$p.f.$	d	@f	palabra	@interna	mod	t_{acceso}
320	1	64	20	2	64	576	144	36	0	10

Al acabar la ejecución de esta instrucción, el procesador pone la dirección de la instrucción `movi r2,#0` en el bus de direcciones. Dado que alas instrucciones son de una palabra, cada una ocupa 4 bytes. Por otro lado, dado que la unidad de direccionamiento es el byte, a la segunda instrucción le corresponde la dirección lógica 324 (la dirección lógica de la anterior, 320, más 4 posiciones, que son las que ocupa la instrucción anterior). Repitiendo todos los cálculos anteriores, así queda la fila en la tabla correspondiente a la segunda instrucción:

@ℓ	$p.\ell.$	d	$t_{trad.}$	$p.f.$	d	@f	palabra	@interna	mod	t_{acceso}
320	1	64	20	2	64	576	144	36	0	10
324	1	68	**1**	2	68	580	145	36	1	**1**

En este caso, incidiremos en dos cuestiones. Por un lado, el tiempo para traducir es de un ciclo ya que en la referencia anterior se ha accedido a la página lógica 1 y, por ello, al hacer la búsqueda en el TLB, se encuentra y ocurre un acierto. Por otro lado, al acceder a memoria principal el tiempo de acceso es también de un único ciclo, ya que las palabras de dirección interna en el módulo igual a 36 estaban ya en el buffer de entrelazado, debido a que ésa era la dirección interna accedida en la instrucción anterior.

Tras esto, la tercera instrucción es `movi r3,#251` y está en la dirección lógica 328 (anterior + 4). Por ello, así queda la tercera línea de la tabla:

@ℓ	$p.\ell.$	d	$t_{trad.}$	$p.f.$	d	@f	palabra	@interna	mod	t_{acceso}
320	1	64	20	2	64	576	144	36	0	10
324	1	68	1	2	68	580	145	36	1	1
328	1	72	1	2	72	584	146	36	2	1

La cuarta instrucción, `load r5,A[r2]`, es la primera dentro del bucle y está en la posición 332. En esta instrucción sucede algo que no ocurría en las tres anteriores: en las tres anteriores había datos inmediatos dentro del propio código (esto es, los valores 1520, 0 y 251), mientras que en la cuarta instrucción el dato está en memoria y se utiliza el direccionamiento indexado para que la instrucción acceda a ese dato. Es decir, en la instrucción viene codificada la dirección base del vector A (2048), y se utiliza un registro índice (`r2`) para calcular la dirección real del dato —para acceder en las pasadas del bucle al elemento `A[i]` del momento, guardando en el registro `r2` el valor del índice `i`—. En este modo de direccionamiento hace falta sumar los valores de los dos parámetros; en este caso, en la primera pasada del bucle ocurre que `r2` = 0 (como consecuencia de la ejecución de la segunda instrucción), por lo que 2048 + 0 = 2048 será la dirección lógica para que el procesador acceda al dato —esto es, el elemento `A[0]` en la primera pasada del bucle—. Por ello, a la cuarta instrucción le corresponden dos líneas en la tabla, la primera a la propia instrucción y la segunda, al dato:

@ℓ	$p.\ell.$	d	$t_{trad.}$	$p.f.$	d	@f	palabra	@interna	mod	t_{acceso}
320	1	64	20	2	64	576	144	36	0	10
324	1	68	1	2	68	580	145	36	1	1
328	1	72	1	2	72	584	146	36	2	1
332	1	76	1	2	76	588	147	36	3	1
2048	8	0	20	0	0	0	0	0	0	10

En cuanto a la dirección del dato, el tiempo de la traducción es de 20 ciclos porque es la primera vez que se accede a la página lógica 8. Por ello, cuando el TLB hace la búsqueda no lo encuentra y se produce un fallo; buscando la traducción en la tabla de páginas. Por otro lado, al acceder a la memoria principal el tiempo de acceso ha sido de 10 ciclos, dado que es la primera vez que se accede a la dirección interna del módulo número 0.

A la quinta instrucción, `load r6,A[r2+16]`, le corresponde la dirección lógica 336, pero en lo que se refiere al dato, en la primera pasada del bucle el programa debe acceder al elemento `A[4]`. Como los elementos del vector ocupan una palabra, cada uno ocupa 4 bytes, el elemento `A[4]` está 16 posiciones después de `A[0]`, es decir su dirección lógica es 2048 + 16 = 2064.

Y podemos seguir así hasta ejecutar todas las instrucciones. Pero en general no rellenaremos la tabla como hemos hecho hasta ahora, horizontalmente, línea a línea, tal y como ocurre en el proceso real. En verdad, rellenarla verticalmente, columna a columna, es más cómodo.

Resumiendo, primero calcularemos las direcciones que generará el procesador al ejecutar ese programa, teniendo en cuenta el tipo de instrucción y el modo de direccionamiento. Es decir, primero rellenaremos las columnas de la izquierda de

la tabla, con todas las direcciones que genere el procesador en la primera pasada del bucle, tanto de instrucciones como de datos.

Instrucción / dato	Dirección lógica
`movi r1,#1520`	320
`movi r2,#0`	324
`movi r3,#251`	328
`bucle:` `load r5,A[r2]`	332 / 2048
`load r6,A[r2+16]`	336 / 2064
`mul r5,r5,r6`	340
`load r10,[r1]`	344 / 1520
`add r6,r6,r10`	348
`store r6,B[r2]`	352 / 512
`addi r2,r2,#4`	356
`subi r3,r3,#1`	360
`bge r3,bucle`	364

Y a partir de ahí completaremos la tabla:

@*l*	*p.l.*	*d*	$t_{trad.}$	*p.f.*	*d*	@*f*	palabra	@interna	mod	t_{acceso}
320	1	64	20	2	64	576	144	36	0	10
324	1	68	1	2	68	580	145	36	1	1
328	1	72	1	2	72	584	146	36	2	1
332	1	76	1	2	76	588	147	36	3	1
2048	8	0	20	0	0	0	0	0	0	10
336	1	80	1	2	80	592	148	37	0	10
2064	8	16	1	0	16	16	4	1	0	10
340	1	84	1	2	84	596	149	37	1	$10^{(1)}$
344	1	88	1	2	88	600	150	37	2	1
1520	5	240	20	5	240	1520	380	95	0	10
348	1	92	1	2	92	604	151	37	3	$10^{(2)}$
352	1	96	1	2	96	608	152	38	0	10
512	2	0	20	1	20	276	69	17	1	10
356	1	100	1	2	100	612	153	38	1	$10^{(3)}$
360	1	104	1	2	104	616	154	38	2	1
364	1	108	1	2	108	620	155	38	3	1

Aclaraciones:

(1) El tiempo de acceso es de 10 ciclos porque las palabras que antes estaban en la dirección interna 37 han desaparecido al acceder a la dirección interna 1. Por ello, al acceder de nuevo a la dirección interna 37 hay que acceder de nuevo a memoria ya que en los buffers de entrelazado se han cargado las palabras de la dirección interna 1.

(2) Como en el caso anterior, pero ahora se ha metido la dirección interna 95 en medio.

(3) Como en los dos casos anteriores, pero esta vez son las palabras de la dirección interna 17 las que se han metido entre dos accesos de la dirección interna 38.

Resumiendo, para traducir las direcciones que genera el programa en la primera pasada del bucle se necesitan 92 ciclos (4 fallos $\times$ 20 ciclos + 12 aciertos $\times$ 1 ciclo), y para acceder a la información de memoria principal se necesitan 106 ciclos (10 accesos de memoria $\times$ 10 ciclos + 6 lecturas en el buffer de entrelazado $\times$ 1 ciclo).

(c) En el apartado anterior hemos calculado el tiempo necesario para la primera pasada. En éste, por el contrario, vamos a calcular el tiempo necesario para traducir las direcciones lógicas y acceder a memoria principal para todas las direcciones generadas por el procesador.

Rellenar la tabla sería demasiado largo. Por ello, haremos unas reflexiones y, obteniendo unas conclusiones generales, calcularemos el tiempo pedido.

Para calcular el tiempo para traducir las direcciones lógicas es suficiente con generalizar la ecuación que hemos obtenido en el apartado anterior:

$$92 \text{ ciclos} = 4 \text{ fallos} \times 20 \text{ ciclos} + 12 \text{ aciertos} \times 1 \text{ ciclo}$$

Es decir, debemos saber el número de fallos y aciertos que ocurren en el TLB. Como hemos dicho, los fallos ocurren la primera vez que se accede a una página lógica (supondremos que una vez cargadas las páginas lógicas en memoria permanecerán ahí hasta acabar la ejecución del programa, es decir, que hay suficiente capacidad en memoria principal para cargar todas las páginas lógicas del programa y que el sistema no las reemplazará por otras por falta de espacio). Así mismo, supondremos que el TLB tiene el número suficiente de entradas para poder almacenar todas las páginas traducidas.

Así, es suficiente saber el número de páginas lógicas que ocupa el programa (instrucciones y datos). Analicemos los componentes uno a uno. Como hemos apreciado en la tabla, todas las instrucciones están en la página lógica 1. Por ello, sólo habrá un fallo debido a las direcciones de las instrucciones.

En cuanto al vector A, hemos visto que los primeros elementos (en la tabla, A[0] y A[4]) están en la página lógica 8. Si analizamos el bucle, vemos que en la última pasada (cuando `i` = 251) se accede a los elementos A[251] y A[255]. Por tanto se accede a 256 elementos de A desde A[0] hasta A[255]. Debido a que cada elemento ocupa 4 bytes, el vector A ocupa al menos 1024 bytes. Dado que en cada página entran 256 bytes, se aprecia que el vector A ocupa 4 páginas. En cualquier caso, hay que verificar si eso es cierto o no, ya que puede ocurrir que el vector A esté en 5 páginas si el elemento A[0] no está al comienzo de una página. No es el caso, ya que hemos calculado que A[0] está en la página lógica 8 con desplazamiento 0. Pero siempre hay que verificarlo. Para ello, calcularemos la dirección lógica del último elemento para obtener en qué página lógica está. Para

calcular la dirección lógica del último elemento tendremos en cuenta la dirección inicial del vector, cuántos elementos hay hasta ahí, y cuántas posiciones ocupa cada elemento:

$$@A[255] = 2048 + 255 \times 4 = 3068$$

Y la página lógica y desplazamiento que corresponden a esa dirección son:

@lógica = 3068 → página lógica = 3068 `div` 2^8 = 3068 `div` 256 = 11

desplazamiento = 3068 `mod` 2^8 = 3068 `mod` 256 = 252

Es decir, las páginas lógicas que ocupa el vector A son: 8, 9, 10 y 11.

En cuanto al vector B, como hemos visto en la tabla, el elemento B[0] está en la página lógica 2. Analizando el comportamiento del bucle, en la última pasada se accede al elemento B[251]. Por tanto, podemos calcular la dirección lógica del último elemento:

$$@B[251] = 512 + 251 \times 4 = 1516$$

Y la página lógica y desplazamiento que corresponden a esa dirección:

@lógica = 1516 → página lógica = 1516 `div` 256 = 5

desplazamiento = 1516 `mod` 256 = 236

Es decir, las páginas lógicas que ocupa el vector B son: 2, 3, 4 y 5.

Finalmente, en lo referente al vector C, tan sólo se accede al elemento C[0] en la dirección lógica 1520 que hemos visto en la tabla que está en la página lógica 5. Es decir, la página lógica 5 la comparten los vectores B y C.

Como hemos visto en la tabla, la página lógica y el desplazamiento asociados a su dirección lógica son las siguientes:

@ lógica = 1520 → página lógica = 1520 `div` 256 = 5

desplazamiento = 1520 `mod` 256 = 240

Resumiendo, entre las instrucciones y los datos se accede a estas páginas lógicas: 1, 2, 3, 4, 5, 8, 9, 10 y 11. Es decir, 7 páginas lógicas, lo que suponen 7 fallos de TLB.

Para calcular los aciertos debemos calcular el número de accesos. Para ello analizaremos el programa. Fuera del bucle hay tres instrucciones que se acceden una única vez cada al empezar la ejecución del programa. Dentro del bucle, en cambio, hay 9 instrucciones y se accede a 4 elementos de los vectores. Es decir, hay 13 accesos en cada pasada. Dado que el bucle se repite 252 veces, el número de accesos total es:

$$\text{número de accesos} = 3 + (9 + 4) \times 252 = 1020$$

De esos accesos 7 son fallos en el TLB y el resto, (1020 – 7) = 1013, aciertos. Por ello, el tiempo invertido en la traducción de direcciones es el siguiente:

$$t_{trad.} = 7 \text{ fallos} \times 20 \text{ ciclos} + 1013 \text{ aciertos} \times 1 \text{ ciclo} = 1153 \text{ ciclos}$$

En cuanto al tiempo de acceso, por el contrario, nos basta con mirar la tabla, ya que ahí se muestra el comportamiento de los accesos. En la tabla vemos que los accesos a las direcciones de datos se intercalan con accesos a direcciones de instrucciones y que por ello, aunque algunas instrucciones coinciden en direcciones internas, desaparecen de los buffers de entrelazado debido a los datos y en el siguiente acceso hay que cargarlos nuevamente. Este comportamiento se repite en todas las pasadas del bucle, ya que los accesos se hacen del mismo modo.

No obstante, hay una excepción que corresponde a la primera instrucción del bucle. De hecho, como se aprecia en la tabla, en la primera pasada se necesita 1 ciclo para acceder a esa instrucción, ya que está en la dirección interna 36, como las instrucciones de fuera del bucle (palabras 144, 145, 146 y 147), que se han cargado al acceder a la primera instrucción con la misma dirección interna en los módulos 0, 1, 2 y 3. Pero a partir de la segunda pasada el comportamiento cambia, ya que antes de esa instrucción se ejecuta la que está en la palabra 155, con la dirección interna 38. Así pues, cuando se necesite la palabra 147 (esto es, la primera del bucle) se necesitarán 10 ciclos porque está en otra dirección interna. Por ello, para calcular los tiempos de acceso debemos distinguir la primera pasada del resto de pasadas del bucle.

Fuera del bucle: $10 + 1 + 1 = 12$ ciclos

Primera pasada del bucle: $9 \times 10 + 4 \times 1 = 94$ ciclos

Siguientes pasadas del bucle: $10 \times 10 + 3 \times 1 = 103$ ciclos

Dado que el bucle se repite 252 veces, excepto la primera vez, el comportamiento se repite 251 veces más. Por ello, en total son necesarios los siguientes ciclos para el acceso a memoria principal:

$$t_{acceso} = 12 + 94 + 103 \times 251 = 25959 \text{ ciclos}$$

En conclusión, se necesitan (1153 + 25959) = 27112 ciclos en total para traducir y acceder a todas las referencias de memoria del programa.