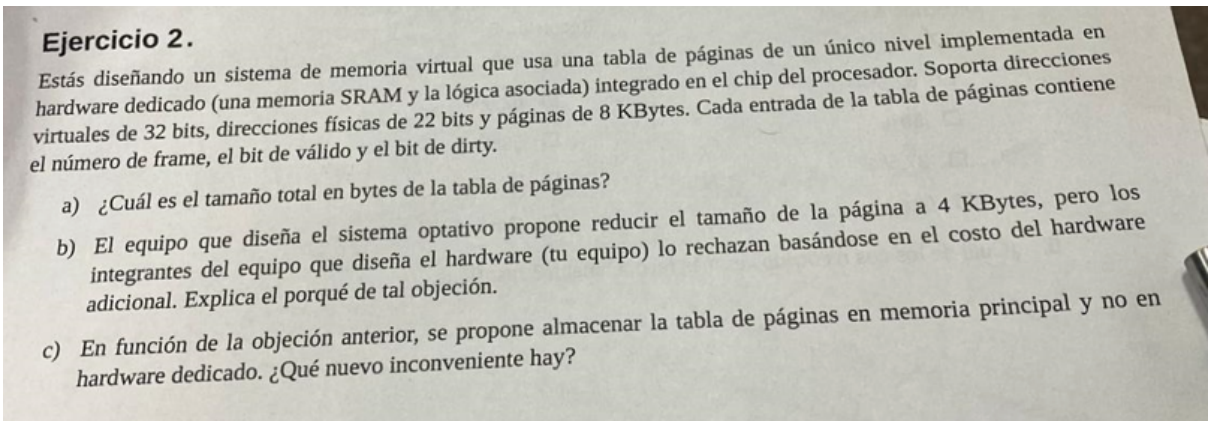


Ejercicio 2.

Estás diseñando un sistema de memoria virtual que usa una tabla de páginas de un único nivel implementada en hardware dedicado (una memoria SRAM y la lógica asociada) integrado en el chip del procesador. Soporta direcciones virtuales de 32 bits, direcciones físicas de 22 bits y páginas de 8 KBytes. Cada entrada de la tabla de páginas contiene el número de frame, el bit de válido y el bit de dirty.

a) *¿Cuál es el tamaño total en bytes de la tabla de páginas?*

b) *El equipo que diseña el sistema optativo propone reducir el tamaño de la página a 4 KBytes, pero los integrantes del equipo que diseña el hardware (tu equipo) lo rechazan basándose en el costo del hardware adicional. Explica el porqué de tal objeción.*

c) *En función de la objeción anterior, se propone almacenar la tabla de páginas en memoria principal y no en hardware dedicado. ¿Qué nuevo inconveniente hay?*

1 byte = 8 bits
Direcciones virtuales de 32 bits
Direcciones físicas 22 bits
Páginas de 8 KB -> 8192 bytes (2^13) -> 65536 bits
Cada entrada tiene el número de frame, el bit de valido y el bit de dirty.

a)
Tamaño de tabla de páginas = (número de frame + BITS EXTRA) x cantidad pag
Offset: 2^13 = 13 bits
número de frame = 22 bits - 13 bits = 9 bits

cantidad de páginas = 2^(dirección lógica - offset) = 2^(32-13) = 2^19= **524288 paginas**

Luego,
Tamaño de la tabla de páginas = ((9 bits + 2 bits(válido y dirty)) x 524288 páginas)
= 5767168 bits -> **720896 bytes**

b) La reducción del tamaño de la página de 8 KBytes a 4 KBytes aumentará el número de entradas en la tabla de páginas en un factor de 2. Esto significa que el tamaño de la tabla de páginas aumentará a 8 MB. Si bien el costo de la memoria SRAM necesaria para implementar la tabla de páginas no aumentará significativamente, el costo de la lógica asociada y el área de chip requerida aumentará significativamente. Por lo tanto, el equipo de diseño de hardware puede rechazar la propuesta debido a limitaciones de costo y área de chip.

c) Si la tabla de páginas se almacena en memoria principal en lugar de en hardware dedicado, cada acceso a la tabla de páginas requerirá una búsqueda en memoria principal y, por lo tanto, será significativamente más lento en comparación con el acceso a la tabla de páginas en hardware dedicado (ya que está integrado en el chip del procesador). Además, cualquier falla de página requerirá una búsqueda en memoria principal para acceder a la tabla de páginas, lo que aumentará significativamente el tiempo de respuesta del sistema. En resumen, el inconveniente de almacenar la tabla de páginas en memoria principal es la disminución significativa del rendimiento en comparación con el acceso a la tabla de páginas en hardware dedicado.

Ejercicio 4.

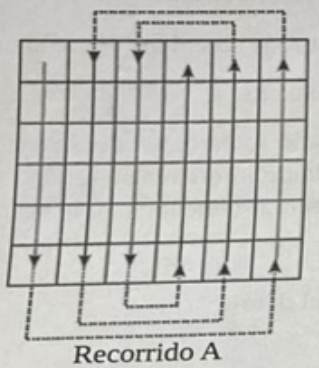
A es una matriz cuadrada de $N \times N$ enteros, donde $N = 8$ y los enteros ocupan 4 bytes. La matriz se encuentra almacenada en memoria siguiendo un orden *row-major*, es decir el elemento *A[i][j]* se encuentra en la posición de memoria *dirección_base_de_A + (i * N + j) * tamaño_elemento*, donde el primer índice representa filas y el segundo columnas.

La dirección base en memoria principal de la matriz A es 0280H. Asumiendo que se tienen direcciones de memoria física de 16 bits y caché física de 128 bytes para datos, implementada con mapeo directo y líneas de 16 bytes.

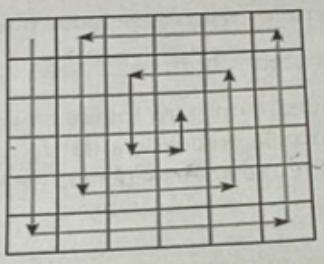
a) Indique cómo se divide la dirección física para acceder a la caché y qué cantidad de bits tiene cada parte.

b) Indique en cuántos bloques se divide la matriz, cuál es la dirección base de cada bloque y cuál es el index de caché asociado a cada uno.

c) Hay dos posibles recorridos de la matriz que resuelven un problema particular (ilustrado en matrices más pequeñas):



Recorrido A



Recorrido B

Asumiendo que al comenzar cada recorrido en la caché de datos no hay nada útil y que operar con los índices involucrados en el recorrido no implica accesos a memoria porque están almacenados en registros.

i. Para cada recorrido ¿cuál fue la tasa de fallos? Marque la opción correcta:

Recorrido A

- ☑ *100%*
- ☐ *62,5%*
- ☐ *50%*
- ☐ *37,5%*
- ☐ *Otro: .....*

Recorrido B

- ☐ 100%
- ☐ 62,5%
- ☐ 50%
- ☐ 37,5%
- ☐ Otro: .....

ii. ¿Cuál de los dos recorridos es más recomendable? Justifique.

Datos:
N = 8 bits
Dirección base = 0280H
Enteros = 4 bytes
Dirección física = 16 bits
Cache física = 128 bytes
m = 1
Lineas = 16 bytes

a) Geometría de la cache; (S,m,L) -> (0.128KB,1,16 bytes)

Donde:

- S (tamaño de caché) = **128 bytes**
- L (tamaño de Línea) = **16 bytes**
- m = 1 (mapeo directo)
- C = (S/L) = (128/16) = **8 líneas**

Index = log_2(C/m) = log_2(8/1) =log_2(8) = **3 bits**
Offset: log_2(L) = log_2(16) = **4 bits**

Tag : 16 (Bits asociados a la dirección física) - Index - Offset = 16 - 3 - 4 = **9 bits**

Por ende, la dirección física queda de la siguiente manera

TAG	**INDEX**	**OFFSET**
9 bits	**3 bits**	**4 bits**

b)
Tamaño total de la Matriz:
NxN = 8x8 = 64 enteros -> cada uno de esos enteros ocupa 4 bytes
-> 64 enteros * 4 bytes c/u = **256 bytes**

Sabemos que el tamaño del bloque es equivalente al tamaño de la línea, por lo tanto:
Cantidad de bloques = 256 bytes (tamaño de la matriz) / 16 bytes (tamaño de la línea)
= 256/16 = **16 bloques**

Como cada fila es de 8 enteros, y cada entero es de 4 bytes entonces tenemos 32 Bytes por fila, por lo tanto, tenemos 2 bloques de 16 bytes por fila.
Como cada fila es de 8 enteros, tenes 4 enteros por cada línea.
Cada bloque es de 16 bytes, y cada elemento es de 4 bytes, entonces entran 4 elementos por bloque o línea.

Nuestra dirección base es **0280H = 640 (base 10)**
direccion_base_de_A + (i x N +j) x tamaño_elemento

Fila 0:

Bloque 1: 640_{10} + [(0x8 +0)x 4 bytes] = 640_{10} -> **0280H** -> 00000000 101 0000
Index = 0

Bloque 2: 640_{10} + [(0x8 + 4)x 4 bytes] = 656_{10} ->**0290H** -> 000000101 001 0000
Index = 1

Fila 1:

Bloque 1: 640_{10} + [(1x8 +0)x 4 bytes] = 672_{10} -> **02A0H** -> 000000101 010 0000
Index = 2

Bloque 2: 640_{10} + [(1x8 +4)x 4 bytes] = 688_{10}-> **02B0H** -> 000000101 011 0000
Index = 3

Fila 2:

Bloque 1: 640_{10} + [(2x8 +0)x 4 bytes] = 704_{10} -> **02C0H** -> 000000101 100 0000
Index = 4

Bloque 2: 640_{10} + [(2x8 +4)x 4 bytes] = 720_{10}-> **02D0H** -> 000000101 101 0000
Index = 5

Fila 3:

Bloque 1: 640_{10} + [(3x8 +0)x 4 bytes] = 736_{10} -> **02E0H** -> 000000101 110 0000
Index = 6

Bloque 2: 640_{10} + [(3x8 +4)x 4 bytes] = 752_{10}-> **02F0H** -> 000000101 111 0000
Index = 7

Fila 4:

Bloque 1: 640_{10} + [(4x8 +0)x 4 bytes] = 768_{10} -> **0300H** -> 000000110 000 0000
Index = 0

Bloque 2: 640_{10} + [(4x8 +4)x 4 bytes] = 784_{10}-> **0310H** -> 000000110 001 0000
Index = 1

→ Tamaño de la línea/bloque es 16 bytes, y cada entero ocupa 4 bytes, por ende (16/4 = 4, 4 enteros por linea/bloque). Cada fila esta compuesta por dos bloques, porque cada FILA tiene 8 enteros, entonces para cada FILA calculo la posicion del primer y segundo bloque, manteniendo la fila (i) correspondiente y utilizando columna j=0 para el primer bloque y columna j=4 para el segundo bloque(en cada bloque entraban 4 enteros) para reemplazar en la formula dada, donde la direccion base de la matriz se mantiene y el tamaño del elemento es 4 (porque cada entero ocupa 4 bytes)
Para obtener el index, paso la dirección obtenida a binario y separo en tag index y offset y listo

Dado que cada bloque ocupa 16 bytes, y la matriz contiene 16 bloques, en total la matriz A ocupa 256 bytes. Como la caché ocupa 128 bytes, solo podemos almacenar en caché la mitad de la matriz

c) ***¿Para cada recorrido, cual fue la tasa de fallos?***

Ocurren 64 fallos ⇒ 1 por cada acceso

Los primeros 8 fallos ocurren ya que no hay nada en caché, los próximos 8 fallos porque los "bloques" impares de cache ya están ocupados, por lo tanto voy reemplazar lo que había en caché.

Luego los próximos 8 fallos ocurren ya que no hay nada en caché en los "bloques" pares y los 8 próximos ocurren por reemplazo.

Todos los accesos siguientes ocasionan un fallo ya que hay que reemplazar, porque en caché se encuentra almacenado el cuadrante de la matriz incorrecto.

$$tasa\ de\ fallos = \frac{Cantidad\ de\ Fallos}{cantidad\ de\ accesos\ totales} = \frac{64}{64} = 100\%$$

1) Se decide mejorar un sistema de memoria virtual agregando un TLB.

Supongamos que el sistema tiene las siguientes características.

- Dirección lógica de 32 bits.
- Páginas de 8kb
- Memoria física máxima de 64mb.

Memory Unit	Access Time	Miss Rate
TLB	1	0.05%
CACHE	1	1.00%
MAIN MEMORY	100	0.03%
HARD DRIVE	150000	0.00%

El miss rate en el TLB y en la cache indica cuan seguido no se encuentra la entrada, mientras que en la memoria principal indica cuan seguido ocurre un Page Fault.

a) En el caso de un acceso a memoria, hay seis posibles eventos que puedan ocurrir: CACHE MISS/HIT, TLB HIT/MISS y PAGE TABLE HIT/MISS (page fault). Analice la viabilidad de las 8 posibles combinaciones del producto cartesiano: (CACHE MISS,CACHE HIT) x (TLB HIT, TLB MISS) x (PAGE TABLE HIT, PAGE FAULT) justificando con un ejemplo aquellas combinaciones factibles y con una explicación aquellas imposibles.

b) Suponiendo que la tabla de paginas se encuentra residente en memoria y nunca se almacena en cache, indique para cada uno de los casos posibles del inciso anterior cual es el tiempo promedio de acceso. Luego, ¿Cuál es el tiempo promedio total de acceso a memoria?

A continuación, analizaré las ocho posibles combinaciones de los eventos CACHE MISS/HIT, TLB HIT/MISS y PAGE TABLE HIT/MISS (page fault):

1. **CACHE MISS, TLB HIT, PAGE TABLE HIT:** Esta combinación es factible. Significa que la dirección virtual solicitada no está en la caché, pero está en la tabla de páginas y en la TLB. Por lo tanto, el sistema operativo puede utilizar la dirección física correspondiente directamente desde la TLB.
2. **CACHE MISS, TLB HIT, PAGE FAULT:** Esta combinación es imposible. Si la dirección virtual está en la TLB, significa que el sistema operativo ya encontró la entrada correspondiente en la tabla de páginas. Por lo tanto, no puede haber un fallo de página en este caso.
3. **CACHE MISS, TLB MISS, PAGE TABLE HIT:** Esta combinación es factible. Significa que la dirección virtual solicitada no está en la caché ni en la TLB, pero está en la tabla de páginas. El sistema operativo debe buscar la dirección física correspondiente en la tabla de páginas y luego cargarla en la caché y en la TLB para futuras referencias.
4. **CACHE MISS, TLB MISS, PAGE FAULT:** Esta combinación es factible. Significa que la dirección virtual solicitada no está en la caché ni en la TLB, y no está en la tabla de páginas. El sistema operativo debe buscar la página correspondiente en el disco duro y cargarla en la memoria principal y la tabla de páginas. Después de eso, el sistema operativo puede buscar la dirección física correspondiente en la tabla de páginas y cargarla en la caché y la TLB.
5. **CACHE HIT, TLB HIT, PAGE TABLE HIT:** Esta combinación es factible. Significa que la dirección virtual está en la caché, la TLB y la tabla de páginas. Por lo tanto, el sistema operativo puede utilizar la dirección física correspondiente directamente desde la caché.
6. **CACHE HIT, TLB HIT, PAGE FAULT:** Esta combinación es imposible. Si la dirección virtual está en la caché y la TLB, significa que el sistema operativo ya encontró la entrada correspondiente en la tabla de páginas. Por lo tanto, no puede haber un fallo de página en este caso.
7. **CACHE HIT, TLB MISS, PAGE TABLE HIT:** Esta combinación es factible. Significa que la dirección virtual está en la caché y la tabla de páginas, pero no está en la TLB. El sistema operativo debe actualizar la TLB con la entrada correspondiente para futuras referencias.
8. **CACHE HIT, TLB MISS, PAGE FAULT:** Esta combinación es imposible. Si la dirección virtual está en la caché, significa que el sistema operativo ya encontró la entrada correspondiente en la tabla de páginas. Por lo tanto, no puede haber un fallo de página en este caso.

En resumen, las combinaciones factibles son: (1) CACHE MISS, TLB HIT, PAGE TABLE HIT; (3) CACHE MISS, TLB MISS, PAGE TABLE HIT; (4) CACHE MISS, TLB MISS, PAGE FAULT; (5) CACHE HIT, TLB HIT, PAGE TABLE HIT; y (7) CACHE HIT, TLB MISS, PAGE TABLE HIT. Las combinaciones imposibles son: (2) CACHE MISS, TLB HIT, PAGE FAULT; (6) CACHE HIT, TLB HIT, PAGE FAULT; y (8) CACHE

En términos generales, el procesador accede primero a la TLB (Translation Lookaside Buffer) y luego a la caché, antes de acceder a la memoria principal o al disco duro. La TLB es una memoria caché especial que almacena las traducciones de direcciones virtuales a direcciones físicas en la memoria principal. La caché es una memoria más rápida y pequeña que la memoria principal, que almacena copias de datos y/o instrucciones utilizados con frecuencia por el procesador. Si la información solicitada no se encuentra en la caché, entonces el procesador accederá a la memoria principal. Si la información solicitada no se encuentra en la memoria principal, entonces se buscará en el disco duro.

La TLB (Translation Lookaside Buffer) es una memoria caché especial que almacena las traducciones de direcciones virtuales a direcciones físicas en la memoria principal. La función principal de la TLB es mejorar el rendimiento del acceso a la memoria virtual, ya que evita la necesidad de buscar la tabla de páginas en la memoria principal para cada acceso a la memoria.

El proceso de funcionamiento de la TLB es el siguiente:

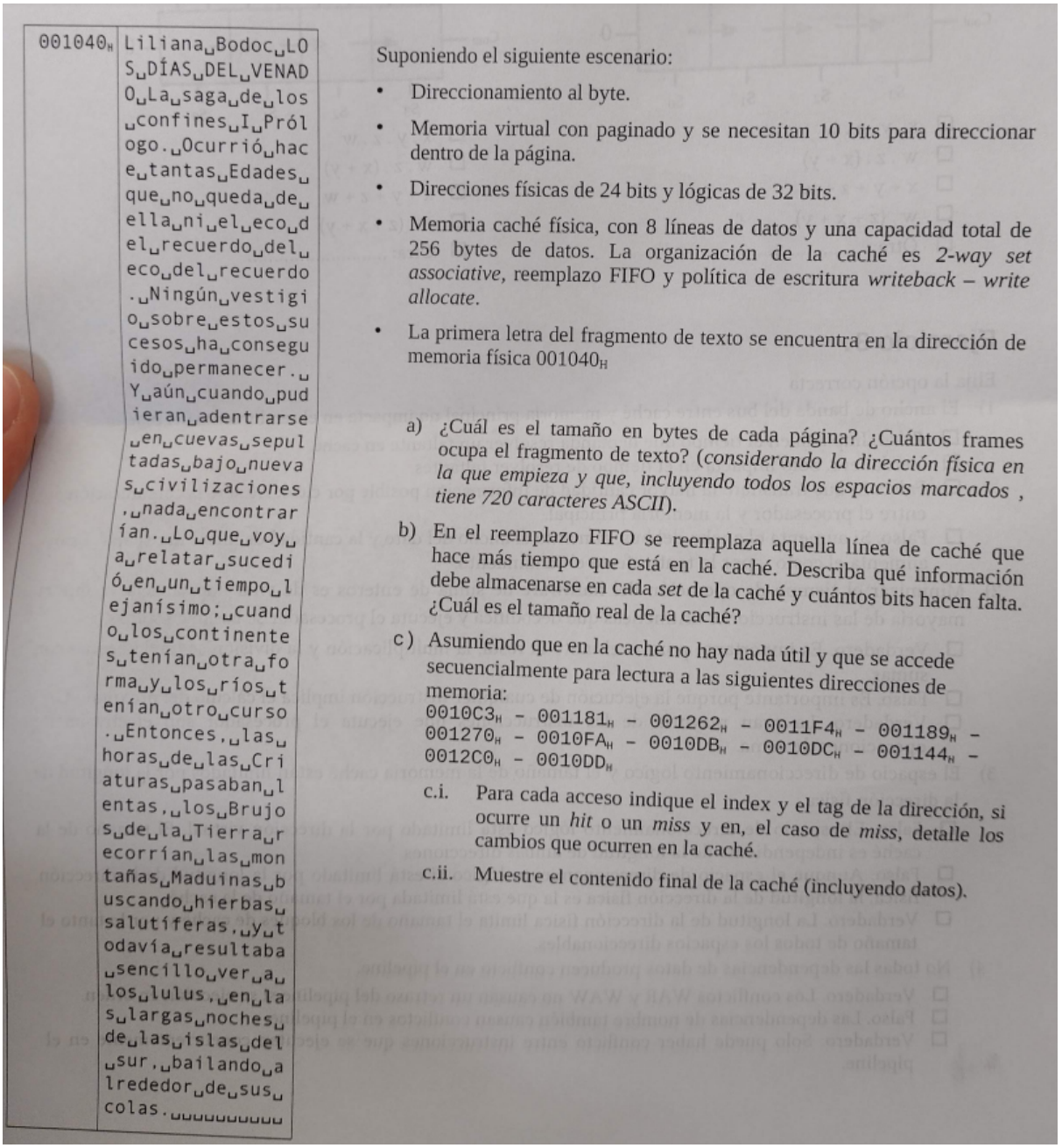
1. Cuando el procesador intenta acceder a una dirección virtual, el TLB busca en su memoria caché para ver si ya se ha realizado la traducción de esa dirección virtual a una dirección física.
2. Si la traducción ya está en la TLB, entonces la dirección física se proporciona al procesador de inmediato y el acceso a la memoria se realiza de manera eficiente.
3. Si la traducción no está en la TLB, entonces se busca en la tabla de páginas en la memoria principal y se actualiza la TLB con la traducción recién obtenida. Luego, se proporciona la dirección física al procesador y el acceso a la memoria se realiza de manera eficiente.
4. Si la tabla de páginas no tiene la traducción de la dirección virtual, entonces se produce una excepción y se realiza una búsqueda completa en la tabla de páginas.
5. Cuando la tabla de páginas se actualiza, la TLB también se actualiza para incluir la traducción recién obtenida. La TLB tiene un tamaño limitado, por lo que si se alcanza su capacidad, se reemplazan las traducciones menos utilizadas.

En resumen, la TLB es una memoria caché que almacena las traducciones de direcciones virtuales a direcciones físicas en la memoria principal. Su función principal es mejorar el rendimiento del acceso a la memoria virtual al evitar la necesidad de buscar la tabla de páginas en la memoria principal para cada acceso a la memoria.

2) Una implementación reducida del PIPELINE RISC de 5 etapas puede aprovechar la etapa EXECUTE para evaluar la condición de un BRANCH y actualizar el PC para la etapa FETCH recién cuando la instrucción BRANCH alcanza la etapa MEMORY. Esto genera STALLS debido a un conflicto de control que puede reducirse si la condición del BRANCH se resuelve en la etapa DECODE.

Explique por qué esta última estrategia puede incrementar los ciclos de STALLS por conflictos de datos.

2) La estrategia de resolver la condición del BRANCH en la etapa DECODE puede generar un incremento en los ciclos de STALLS por conflictos de datos debido a que la resolución de la condición del BRANCH puede requerir el uso de datos que aún no están disponibles en la etapa DECODE.
En una implementación del PIPELINE RISC de 5 etapas, la etapa DECODE es la que se encarga de leer los registros y decodificar la instrucción. Si la condición del BRANCH se resuelve en esta etapa, entonces se pueden tomar decisiones sobre la ejecución del BRANCH antes de que se necesiten los datos en la etapa EXECUTE. Sin embargo, si la resolución de la condición del BRANCH requiere datos que aún no están disponibles en la etapa DECODE, entonces se puede generar un conflicto de datos que resulta en ciclos de STALLS adicionales.
En este escenario, el PIPELINE se detiene en la etapa DECODE mientras espera a que los datos estén disponibles, lo que puede aumentar el número de ciclos de STALLS en comparación con la estrategia de resolver la condición del BRANCH en la etapa EXECUTE. En esta última estrategia, la resolución de la condición del BRANCH se realiza en una etapa posterior donde los datos necesarios para la evaluación de la condición del BRANCH ya están disponibles, lo que reduce la probabilidad de conflictos de datos y disminuye los ciclos de STALLS.



001040_H	Liliana␣Bodoc␣LOS␣DÍAS␣DEL␣VENADO␣La␣saga␣de␣los␣confines␣I␣Prólogo.␣Ocurrió␣hace␣tantas␣Edades␣que␣no␣queda␣de␣ella␣ni␣el␣eco␣del␣recuerdo␣del␣eco␣del␣recuerdo.␣Ningún␣vestigio␣sobre␣estos␣sucesos␣ha␣conseguido␣permanecer.␣Y␣aún␣cuando␣pudieran␣adentrarse␣en␣cuevas␣sepultadas␣bajo␣nuevas␣civilizaciones,␣nada␣encontrarían.␣Lo␣que␣voy␣a␣relatar␣sucedió␣en␣un␣tiempo␣lejanísimo;␣cuando␣los␣continentes␣tenían␣otra␣forma␣y␣los␣ríos␣tenían␣otro␣curso.␣Entonces,␣las␣horas␣de␣las␣Criaturas␣pasaban␣lentas,␣los␣Brujos␣de␣la␣Tierra␣recorrían␣las␣montañas␣Maduinas␣buscando␣hierbas␣salutíferas,␣y␣todavía␣resultaba␣sencillo␣ver␣a␣los␣lulus,␣en␣las␣largas␣noches␣de␣las␣islas␣del␣sur,␣bailando␣alrededor␣de␣sus␣colas.␣␣␣␣␣␣␣␣␣␣

Suponiendo el siguiente escenario:

- Direccionamiento al byte.
- Memoria virtual con paginado y se necesitan 10 bits para direccionar dentro de la página.
- Direcciones físicas de 24 bits y lógicas de 32 bits.
- Memoria caché física, con 8 líneas de datos y una capacidad total de 256 bytes de datos. La organización de la caché es *2-way set associative*, reemplazo FIFO y política de escritura *writeback – write allocate*.
- La primera letra del fragmento de texto se encuentra en la dirección de memoria física 001040_H

a) ¿Cuál es el tamaño en bytes de cada página? ¿Cuántos frames ocupa el fragmento de texto? (*considerando la dirección física en la que empieza y que, incluyendo todos los espacios marcados , tiene 720 caracteres ASCII*).

b) En el reemplazo FIFO se reemplaza aquella línea de caché que hace más tiempo que está en la caché. Describa qué información debe almacenarse en cada *set* de la caché y cuántos bits hacen falta. ¿Cuál es el tamaño real de la caché?

c) Asumiendo que en la caché no hay nada útil y que se accede secuencialmente para lectura a las siguientes direcciones de memoria:
$0010C3_H$ - 001181_H - 001262_H - $0011F4_H$ - 001189_H - 001270_H - $0010FA_H$ - $0010DB_H$ - $0010DC_H$ - 001144_H - $0012C0_H$ - $0010DD_H$

c.i. Para cada acceso indique el index y el tag de la dirección, si ocurre un *hit* o un *miss* y en, el caso de *miss*, detalle los cambios que ocurren en la caché.

c.ii. Muestre el contenido final de la caché (incluyendo datos).

Integrador:

1) ¿Cuál es el tamaño en bytes de cada página? ¿Cuántos frames ocupa el fragmento de texto?

Paginación:
Tamaño de página en bytes: 2^(offset) = 2^10 = **1024 bytes**
Espacio de direccionamiento físico = 2^(direccion fisica) = 2^24 = **16777216 bytes**
Cantidad de frames: espacio de direccionamiento fisico/ el tamaño del frame (pagina) =
16777216 bytes/1024 bytes = 16384 bytes paginas/frames
720 caracteres ASCII -> cada caracter ASCII ocupa 1 byte -> 720x1 = **720 bytes**

720 bytes de texto/1024 bytes de tamaño de frame = **0.7031 frames -> redondeo a 1 frame**

Convertir Decimal a Hexadecimal, Octal y Binario

Decimal Son de base 10 y usan 10 dígitos para expresarse 0 1 2 3 4 5 6 7 8 9

Binario 0 1
Son de base 2 y usan 2 dígitos para expresarse

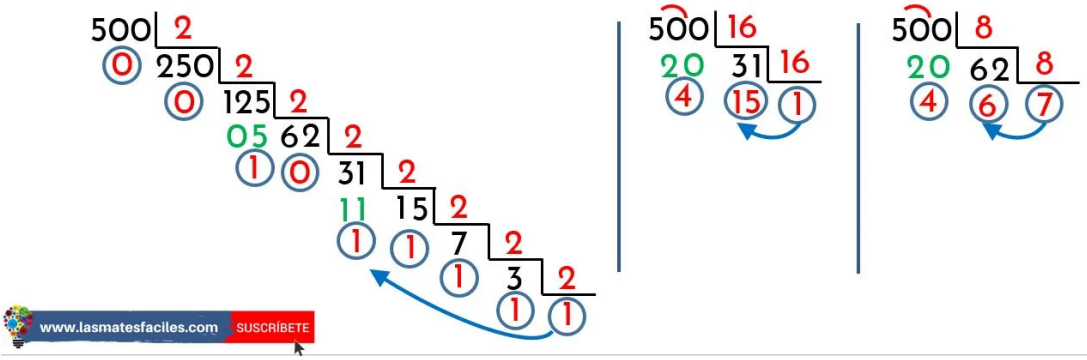
Hexadecimal 0 1 2 3 4 5 6 7 8 9

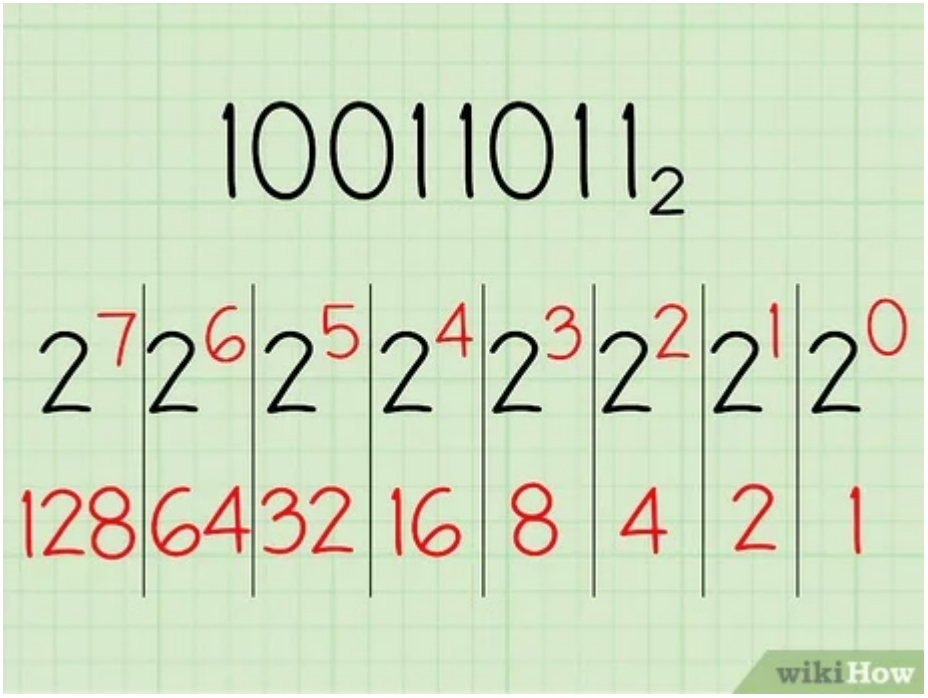
A	B	C	D	E	F
10	11	12	13	14	15

Son de base 16 y usan 16 dígitos para expresarse

Octal 0 1 2 3 4 5 6 7
Son de base 8 y usan 8 dígitos para expresarse

$500_{(10)} = 111110100_{(2)} = 1F4_{(16)} = 764_{(8)}$





$$39B2C_{(16} = 3 \cdot 16^4 + 9 \cdot 16^3 +$$

$$+11 \cdot 16^2 + 2 \cdot 16 +$$

$$+12 \cdot 1 =$$

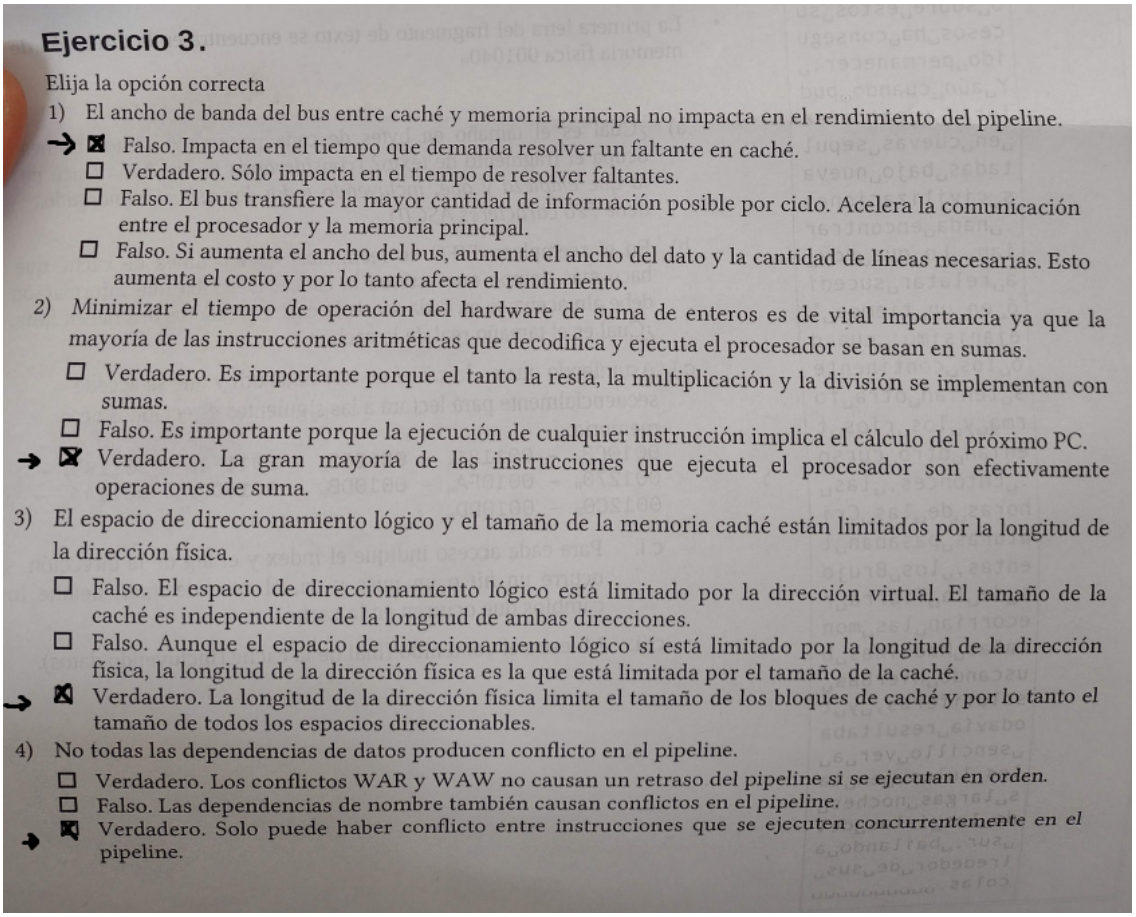
$$= 236332_{(10}$$

Hexadecimal	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

Los bits "V", "D" y "U" se utilizan en la memoria caché para diferentes propósitos:

- El bit "V" (de validez) indica si la línea de caché contiene datos válidos o no. Si el bit está en 1, significa que los datos en la línea de caché son válidos y se pueden utilizar. Si está en 0, significa que la línea de caché no contiene datos válidos y que los datos deben ser cargados desde la memoria principal antes de poder ser utilizados.
- El bit "D" (de suciedad o dirty bit) indica si la línea de caché ha sido modificada o no. Si el bit está en 1, significa que la línea de caché ha sido modificada y que los datos en la caché son diferentes de los datos en la memoria principal. Si está en 0, significa que la línea de caché no ha sido modificada y que los datos en la caché son iguales a los datos en la memoria principal.
- El bit "U" (de uso o de último acceso) se utiliza para implementar algoritmos de reemplazo en la caché, como LRU (menos recientemente utilizado) o FIFO (primero en entrar, primero en salir). El bit "U" indica cuándo se accedió por última vez a la línea de caché. Cuando se necesita reemplazar una línea de caché, se utiliza el bit "U" para determinar cuál de las líneas de caché no se ha utilizado durante más tiempo, y se reemplaza esa línea.

En general, estos bits se utilizan para mejorar el rendimiento y la eficiencia de la caché, minimizando la cantidad de veces que se necesita acceder a la memoria principal y maximizando el uso de los datos almacenados en la caché.



Ejercicio 3.

Elija la opción correcta

1) El ancho de banda del bus entre caché y memoria principal no impacta en el rendimiento del pipeline.
   - ☒ Falso. Impacta en el tiempo que demanda resolver un faltante en caché.
   - ☐ Verdadero. Sólo impacta en el tiempo de resolver faltantes.
   - ☐ Falso. El bus transfiere la mayor cantidad de información posible por ciclo. Acelera la comunicación entre el procesador y la memoria principal.
   - ☐ Falso. Si aumenta el ancho del bus, aumenta el ancho del dato y la cantidad de líneas necesarias. Esto aumenta el costo y por lo tanto afecta el rendimiento.

2) Minimizar el tiempo de operación del hardware de suma de enteros es de vital importancia ya que la mayoría de las instrucciones aritméticas que decodifica y ejecuta el procesador se basan en sumas.
   - ☐ Verdadero. Es importante porque el tanto la resta, la multiplicación y la división se implementan con sumas.
   - ☐ Falso. Es importante porque la ejecución de cualquier instrucción implica el cálculo del próximo PC.
   - ☒ Verdadero. La gran mayoría de las instrucciones que ejecuta el procesador son efectivamente operaciones de suma.

3) El espacio de direccionamiento lógico y el tamaño de la memoria caché están limitados por la longitud de la dirección física.
   - ☐ Falso. El espacio de direccionamiento lógico está limitado por la dirección virtual. El tamaño de la caché es independiente de la longitud de ambas direcciones.
   - ☐ Falso. Aunque el espacio de direccionamiento lógico sí está limitado por la longitud de la dirección física, la longitud de la dirección física es la que está limitada por el tamaño de la caché.
   - ☒ Verdadero. La longitud de la dirección física limita el tamaño de los bloques de caché y por lo tanto el tamaño de todos los espacios direccionables.

4) No todas las dependencias de datos producen conflicto en el pipeline.
   - ☐ Verdadero. Los conflictos WAR y WAW no causan un retraso del pipeline si se ejecutan en orden.
   - ☐ Falso. Las dependencias de nombre también causan conflictos en el pipeline.
   - ☒ Verdadero. Solo puede haber conflicto entre instrucciones que se ejecuten concurrentemente en el pipeline.

Un pipeline es una técnica utilizada en la arquitectura de computadoras para mejorar el rendimiento de los procesadores. Consiste en dividir la ejecución de una instrucción en varias etapas, de modo que cada etapa se encargue de una parte específica del procesamiento de la instrucción. De esta manera, se pueden ejecutar varias instrucciones de forma simultánea, lo que aumenta la tasa de instrucciones por segundo que el procesador puede procesar.
Las etapas de un pipeline típico en el contexto de la arquitectura de computadoras son las siguientes:

1. **Fetch (obtención)**: En esta etapa se lee la instrucción desde la memoria y se almacena en el registro de instrucciones. El contador de programa (PC) también se actualiza para apuntar a la siguiente instrucción.
2. **Decode (decodificación)**: En esta etapa se determina el tipo de instrucción y los operandos que se deben utilizar. Se identifican los registros y las direcciones de memoria que se necesitan para completar la instrucción.
3. **Execute (ejecución)**: En esta etapa se realiza la operación o el cálculo especificado por la instrucción. Los operandos se leen desde los registros o la memoria, se realizan las operaciones y se almacenan los resultados en los registros o en la memoria.
4. **Memory (memoria)**: En esta etapa se accede a la memoria para leer o escribir datos. Si la instrucción no implica ninguna operación de memoria, esta etapa se omite.
5. **Writeback (escritura)**: En esta etapa se escriben los resultados de la instrucción en los registros.

Cada etapa del pipeline procesa una instrucción diferente al mismo tiempo. Esto permite que, en teoría, el procesador pueda emitir una nueva instrucción en cada ciclo de reloj. Sin embargo, para que el pipeline funcione correctamente, es necesario evitar que las instrucciones se intercalen entre sí. Esto se logra mediante el uso de buffers y registros de desplazamiento que retienen temporalmente las instrucciones mientras se procesan.

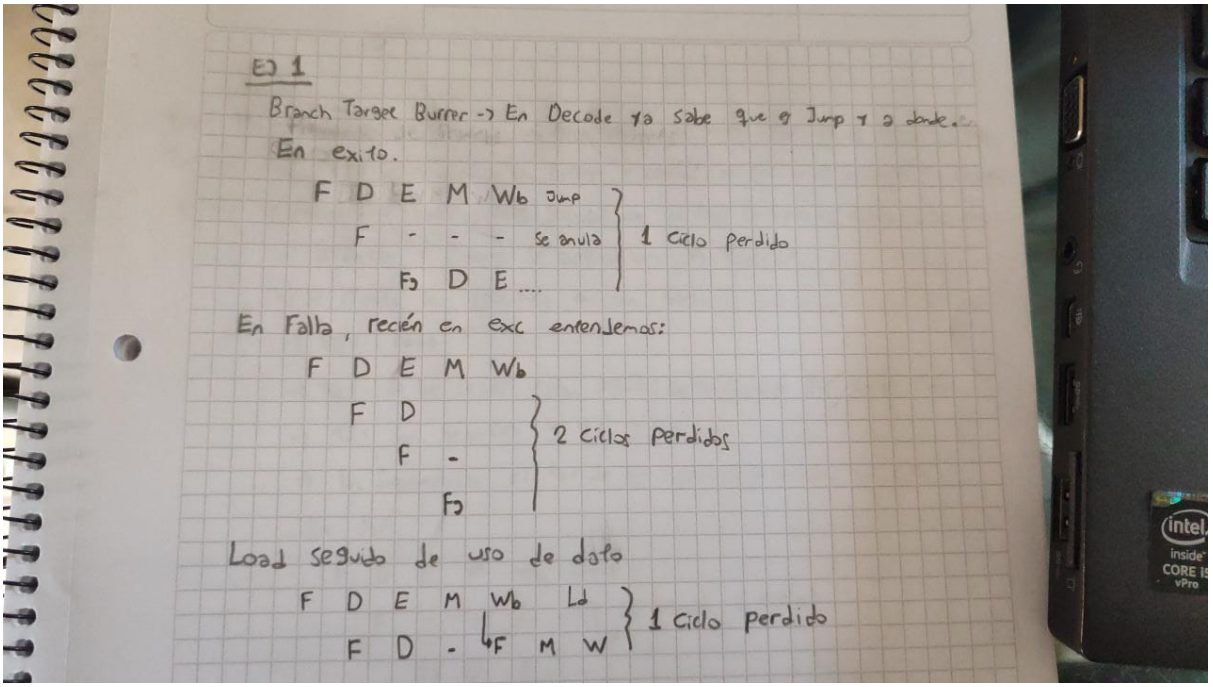
Ejercicio 1.

Se tiene un procesador implementado con pipeline escalar con las siguientes características:

- Predicción de Branch con un branch target buffer que incluye Jumps. El 10% de las veces el PC no se encuentra en buffer y, además, el 10% de predicciones que realiza son erróneas.
- Forwarding tanto para los datos generados por la ALU como para los obtenidos desde la memoria.
- Las instrucciones Branch y Jump actualizan el PC en la etapa Execute.

Se decide adaptar la implementación para lograr un superescalar de grado 4, pero, para mantener los requerimientos energéticos acotados, se reduce la frecuencia del reloj en un 10%. Calcule la eficiencia teórica alcanzada por haber cuadruplicado el ancho del pipeline si la secuencia de instrucciones a ejecutar tiene las siguientes características:

- 25% son load
- el 10% de los loads están seguidos de una instrucción que usa el dato
- un 15% son branch y un 4% son jump.
- Al ejecutarse el mismo código en el procesador superescalar, el 10% de las instrucciones pierden un ciclo porque no era posible paralelizarlas completamente debido a conflictos de datos.



Caso Base = (19%*(80%*1 + 20%*2) + 25%*10%*1)freq

Caso Mejorado = (19%*(80%*1 + 20%*2) + 25%*10%*1+10%*1)*4*(9/10)freq

CB = (0.19(0.8+0.4)+0.025)freq = 0.253 freq
CM = (0.19(0.8+0.4)+0.025+0.1).4.(9/10)freq = 1.2708 freq

Eficiencia = ((CM/CB))/n = 5.02/4 = 1.255 -> **125% más eficiente**
n en este caso es 4 por el grado 4 del superescalar