APELLIDOS **NOMBRE** **BLOQUE A**

ESTRUCTURA DE COMPUTADORES 21 de junio de 2023

Normas: Escribe los apellidos y el nombre en cada hoja de examen y en cada hoja adicional que utilices. Es posible escribir la respuesta en la misma hoja de cada ejercicio, pero si utilizas hojas adicionales no mezcles en la misma hoja respuestas de bloques diferentes: **se entregará cada bloque por separado**.

[0.75p] 1. Un *benchmark* en un procesador determinado tarda 5 s en finalizar, con un total de 20×10^9 instrucciones ejecutadas. Sabemos que en este *benchmark* el 25% del tiempo se emplea en la ejecución de instrucciones de punto flotante. Contesta razonadamente estas cuestiones:

(a) **[0.2p]** ¿Cuál es el MIPS alcanzado?

$$MIPS = \frac{\text{Num. instrucións}}{\text{Tempo de execución (s)} \times 10^6} = \frac{2 \times 10^{10}}{5 \times 10^6} = 4000$$

(b) **[0.2p]** ¿Cuál es el GFLOPS alcanzado?

$$GFLOPS = \frac{\text{Num. instrucións P. F.}}{\text{Tempo de execución (s)} \times 10^9} = \frac{2 \times 10^{10} \times 0.25}{5 \times 10^9} = 1$$

(c) **[0.35p]** Aplicando la ley de Amdhal, ¿cuánto tendríamos que mejorar la unidad de punto flotante si queremos conseguir una aceleración del 10%?

Ley de Amdhal:

$$A = \frac{1}{(1 - F_{PF}) + \frac{F_{PF}}{A_{PF}}}$$

Temos $A = 1.1$ *e* $F_{PF} = 0.25$ *co cal:*

$$1.1 = \frac{1}{(1 - 0.25) + \frac{0.25}{A_{PF}}}$$

Por tanto:

$$A_{PF} = \frac{0.25 \times 1.1}{1 - 0.75 \times 1.1} \approx 1.57$$

Teríamos que mellorar a unidade de punto flotante un 57%.

[**2.0p**] 2. El siguiente código se ejecuta en el procesador segmentado MIPS de 5 etapas estudiado en clase: IF, ID, EX, MEM e WB. El salto se decide en la etapa ID, se usa la técnica de salto fijo no efectivo, tiene una unidad de detección de riesgos en la etapa ID y una unidad de anticipación en la etapa EX. La ejecución de una operación de suma en la unidad de punto flotante requiere 3 ciclos y está segmentada. Las etapas MEM e WB solo pueden albergar una instrucción.

```
1   addi $s0, $0, 4
  Loop:
2   lwc1 $f1, 0($a1)
3   add.s $f2, $f1, $f2
4   addi $a1, $a1, 4
5   addi $s0, $s0, -1
6   bne $s0, $0, Loop
  End:
7   swc1 $f2, 0($a2)
```

Contesta **razonadamente** las siguientes cuestiones:

(a) [**0.2p**] Indica dos dependencias verdaderas que haya dentro del bucle.

- Entre as instrucións 2 e 3, xa que na 2 `lwc1` modifica `$f1` e a continuación na 3 `add.s` utiliza ese rexistro.
- Entre as instrucións 5 e 6, xa que na 5 `addi` modifica `$s0` e a continuación na 6 `bne` compara ese valor con 0.

(b) [**0.1p**] Indica una antidependencia que haya dentro del bucle.

Entre as instrucións 2 e 4, xa que na 2 `lwc1` le a posición de memoria a partir do enderezo almacenado en `$a1` e na 4 a `addi` modifica ese valor.

(c) [**0.2p**] ¿Se produce alguna anticipación en la ejecución de este código? En caso afirmativo indica entre qué instrucciones y etapas.

Si, entre as instrucións 2 e 3. A instrución 3 (`add.s`) ten que esperar a que a 2 (`lwc1`) lea o valor de memoria, pero antes de que se escriba en `$f1` na etapa WB xa se pode ler na etapa EX desde o rexistro de segmentación MEM/WB.

(d) [**0.3p**] Indica los riesgos de datos que se producen en el bucle del código y de qué tipo son.

Hai dous de tipo RAW (lectura despois de escritura):

- Entre as instrucións 2 e 3 pola escritura de `$f1` en 2 e a lectura en 3.
- Entre as instrucións 5 e 6 pola escritura de `$s0` en 5 e a lectura en 6.

(e) **[0.3p]** ¿Cómo se podría optimizar la ejecución del código reordenando las instrucciones 3, 4 y 5?

Poñendo a instrución 5 despois da 2 evítase o bloqueo entre a 2 e a 3 e tamén o que se produce entre a 5 e a 6 debido aos riscos de datos, conseguindo executar o código en menos ciclos. Quedaría:

```
1 addi $s0, $0, 4
2 Loop: lwc1 $f1, 0($a1)
5 addi $s0, $s0, -1
3 add.s $f2, $f1, $f2
4 addi $a1, $a1, 4
6 bne $s0, $0, Loop
7 End: swc1 $f2, 0($a2)
```

(f) **[0.5p]** Partiendo del código original, si utilizásemos salto retardado en el lugar de salto fijo no efectivo, ¿qué instrucción colocarías en el hueco de retardo?

Podería colocarse tanto a 3, como a 4, como a 5:

- Se poñemos a 3 evitamos o bloqueo con 2, que é dun ciclo, e o código se executa correctamente. Crea un risco RAW de 2 ciclos á saída do bucle coa instrución 7.
- Se poñemos a 4 non evitamos ningún bloqueo e creamos un risco de datos coa 2 no cambio de iteración.
- Se poñemos a 5 temos que modificar o valor inicial en 1: `addi $0, $0, 3`. Evítase o bloqueo entre 5 e a 6, que demora 2 ciclos.
- Se deixamos a 7, o código funcionaría igual, pero non se melloraría o rendemento.
- Se poñemos a 2, habería que duplicala antes de entrar no bucle e podería dar un erro na iteración final por acceso a unha posición non válida.

A mellor solución é poñer a 5 porque reduce a latencia de cada iteración do bucle en 2 ciclos.

(g) **[0.4p]** ¿Qué etapas de **manera efectiva** son utilizadas por la instrucción `bne`? Si el salto se decidiese en la etapa EX, ¿cuáles serían las etapas que esta instrucción usaría?

Como o salto se decide na etapa ID, a `bne` só utiliza as etapas IF, para cargar de memoria a instrución, e a etapa ID, onde se comparan os rexistros e se calcula a dirección de salto, que se fai efectivo se os valores dos rexistros son distintos.

No caso de que o salto se decidise na etapa EX, é nesta etapa onde se calcula a dirección de salto e onde se fai a comparación cunha resta. Logo na etapa MEM se activa o salto modificando o valor de PC no caso de que a comparación feita na etapa anterior indicase que os rexistros teñen valores distintos. Así neste caso a `bne` utilizaría de maneira efectiva as etapas IF, ID, EX e MEM.

APELLIDOS **NOMBRE** **BLOQUE B**

[2p] 3. Un sistema incluye una caché L1 de 32 KiB, asociativa por conjuntos de 2 vías, con líneas de 16 B. El sistema de memoria sigue una política de postescritura. Estando la caché inicialmente vacía, se ejecuta el siguiente código.

```
int i, A[256], B[128];
for (i = 0; i < 4; i++)
  B[i] = A[i+1] + A[i+128];
```

El vector `A` se encuentra almacenado a partir de la dirección de memoria 0x0A000, mientras que `B` se encuentra almacenado inmediatamente a continuación de `A`. El resto de variables no provocan accesos a memoria caché. Cada elemento de tipo `int` ocupa exactamente 4 bytes.

(a) **[0.3p]** Calcula las direcciones de memoria a las que accede este código.

&A = 0x0A000

&B = &A + 256 × 4 = 0x0A000 + 0x400 = 0x0A400

- Iteración i = 0:
  - A[1] `0x0A004` (lectura)
  - A[128] `0x0A200` (lectura)
  - B[0] `0x0A400` (escritura)
- Iteración i = 1:
  - A[2] `0x0A008` (lectura)
  - A[129] `0x0A204` (lectura)
  - B[1] `0x0A404` (escritura)
- Iteración i = 2:
  - A[3] `0x0A00C` (lectura)
  - A[130] `0x0A208` (lectura)
  - B[2] `0x0A408` (escritura)
- Iteración i = 3:
  - A[4] `0x0A010` (lectura)
  - A[131] `0x0A20C` (lectura)
  - B[3] `0x0A40C` (escritura)

(b) **[0.3p]** Determina cómo se divide una dirección física desde el punto de vista de la caché.

Caché de 32KiB, líneas de 16B, asociativa por conjuntos de 2 vías

- Tamaño de línea: 16 $B = 2^4$ $B \rightarrow 4$ *bits desplazamiento*
- Número de líneas: $\#L = \frac{32\ KiB}{16\ B/linea} = \frac{2^{15}\ B}{2^4\ B/linea} = 2^{11} lineas$
- Número de conjuntos: $\frac{2^{11}\ lineas}{2\ vias/conjunto} = 2^{10} conjuntos \rightarrow 10\ bits\ indice$
- Etiqueta: 6 *bits* (bits restantes hasta 20)

Dirección física		
Etiqueta	Índice	Desplazamiento
6 bits	*10 bits*	*4 bits*

(c) **[0.4p]** Rellena los campos de la siguiente tabla para los accesos a memoria caché: dirección, etiqueta e índice/conjunto (en **hexadecimal**); si el acceso es un acierto o un fallo; en este último caso, qué tipo de fallo; y si se produce una escritura en memoria principal (MP WR).

Dirección	Etiqueta	Conjunto	Acierto/Fallo	Tipo de fallo	MP WR
0x0A004	0x02	0x200	F	Forzoso	—
0x0A200	0x02	0x220	F	Forzoso	—
0x0A400	0x02	0x240	F	Forzoso	N
0x0A008	0x02	0x200	A	—	—
0x0A204	0x02	0x220	A	—	—
0x0A404	0x02	0x240	A	—	N
0x0A00C	0x02	0x200	A	—	—
0x0A208	0x02	0x220	A	—	—
0x0A408	0x02	0x240	A	—	N
0x0A010	0x02	0x201	F	Forzoso	—
0x0A20C	0x02	0x220	A	—	—
0x0A40C	0x02	0x240	A	—	N

(d) **[0.3p]** Indica cómo quedaría el directorio caché en las líneas ocupadas tras la ejecución de este código. BV = Bit de validez, BM = Bit de modificación

Conjunto	Vía	BV	BM	Etiqueta
0x200	0	1	0	0x02
0x201	0	1	0	0x02
0x220	0	1	0	0x02
0x240	0	1	1	0x02

(e) **[0.3p]** Sabiendo que el tiempo medio de acceso a memoria durante la ejecución del apartado anterior fue de 52 ciclos, y que el tiempo de acierto caché es de 1 ciclo, calcula el tiempo medio de acceso al nivel inferior.

El tiempo medio de acceso al nivel inferior es la penalización por fallo en este nivel.

- 12 accessos → 8 aciertos, 4 fallos.
- T = 52 ciclos = 1 ciclo/acierto + 4/12 fallos × P_{fallo}
- $P_{fallo} = (52 - 1) \times \frac{12}{4} = 51 \times 3 =$ $\boxed{153\ ciclos}$

(f) **[0.4p]** Calcula el tamaño mínimo de la caché L1 tal que, manteniendo constante el resto de su configuración (tamaño de línea, asociatividad), preserva la tasa de fallos observada.

Se producirá un conflicto en el momento en que las líneas que en este caso se asignan a los conjuntos 0x200 y 0x240 se asignen al mismo conjunto, es decir, en el momento en que el nº de conjuntos disminuya a 0x40 → Tamaño caché = 0x40 × 2 vías/conjunto × 16B/línea = 2 KiB.

APELLIDOS **NOMBRE** **BLOQUE C**

[2p] 4. Un sistema cuenta con 64 GiB de memoria principal (2^{36} B) y un espacio virtual paginado de 1 TiB (2^{40} B) con un esquema de traducción en 2 niveles. El tamaño de página es de 64 KiB (2^{16} B). Cada tabla de páginas de primer o segundo nivel tiene 4096 entradas (2^{12}) de 4 bytes cada una, donde el bit más significativo de cada entrada es el bit de residencia o validez, y el número de página física está codificado en los bits menos significativos. Los bits restantes son bits de control. La CPU solicita los datos correspondientes a la dirección virtual `0x01 F004 D38C`. En la entrada de la tabla de páginas de primer nivel se encuentra el contenido `0xA274 14A0` y en la entrada de la tabla de páginas de segundo nivel `0x8002 4605`.

(a) **0.3p** Determina en qué campos (longitud y valores) se divide la dirección virtual.

El tamaño de página es 2^{16} bytes, por lo que se necesitan 16 bits para el desplazamineto. Cada TP de primer o segundo nivel tiene 2^{12} entradas. Entonces son 12 bits para PV1, 12 bits para PV2 y 16 bits para desplazamiento.

Dirección virtual		
numPV1	numPV2	Δ_p
12 bits	*12 bits*	*16 bits*
0x01F	0x004	0xD38C

(b) **0.3p** Calcula la dirección física de la entrada correspondiente de las tabla de páginas de nivel 2.

Del contenido de la entrada de la TP1, que está en el enunciado: 0xA27414A0, comprobamos el bit de residencia: comienza por 0xA, en binario **1**010, y obtenemos la página física en la que se encuentra la tabla de nivel 2 (los 20 bits menos significativos): 0x414A0. El registro base de la tabla es la dirección física en la que comienza esa tabla: 0x4 14A0 0000. Partiendo de esa dirección desplazamos 4 bytes/entrada hasta la entrada 0x004, correspondiente al número de PV de nivel 2.

- DirE2 = 0x4 14A0 0000 + 0x004 × 4 = 0x4 14A0 0010

(c) **0.4p** ¿Cuál es la dirección física resultante del proceso de traducción?

La página física son los 20 bits menos significativos de la entrada de la TP2: 0x24605. También es necesario comprobar el bit de residencia: comienza por 0x8 → **1**000.
A esa página física le concatenamos el desplazamiento 0xD38C.

Dirección física	
numPF1	Δ_p
20 bits	*16 bits*
0x24605	0xD38C

La dirección física resultante es 0x2 4605 D38C.

(d) **0.4p** Calcula cuánto ocupan las tablas de páginas que es necesario que residan en memoria física para realizar esta traducción

Es suficiente con que resida la TP de primer nivel y una TP de segundo nivel. Cada tabla ocupa $2^{12} \times 4\ bytes = 2^{14}\ bytes = 16\ KiB$. Por tanto, las tablas necesarias para la traducción ocupan $2 \times 16\ KiB = 32\ KiB$

(e) **0.2p** Determina si existe fragmentación interna o externa en este sistema en la gestión de la memoria virtual (pista: puedes utilizar los cálculos del apartado anterior).

Cada TP hemos calculado que ocupa 16 KiB, pero el tamaño de página es de 64 KiB. Por lo tanto, habrá fragmentación interna porque de cada página utilizada en el proceso de traducción sólo se utilizará la cuarta parte.

No existe fragmentación externa en un sistema paginado.

(f) **0.4p** La caché L1 es de 32 KiB, asociativa por conjuntos de 8 vías con un tamaño de línea de 64 bytes. Determina si podría ser una caché con índices virtuales y etiquetas físicas (VIPT).

Las direcciones caché en L1 tienen 6 bits de desplazamiento (el tamaño de línea es de 64 bytes) y 6 bits de índice (512 líneas en conjuntos de 8 vías $\rightarrow$ 64 conjuntos).

Para implementar la caché VIPT es necesario y suficiente que los bits de desplazamiento (6) e índice (6) de la dirección física coindidan dentro de los bits de desplazamiento de la dirección virtual (16). Como $6 + 6 < 16$, la caché sí podría ser VIPT.

[0.5p] 5. Disponemos de 4 discos de 1 TB en una configuración RAID 0.

(a) **0.3p** ¿Cuántos discos necesitaríamos para mantener en el sistema la misma cantidad de información neta pero si lo quisiéramos configurar como RAID 4, 5 o 6?

RAID 0 no tiene redundancia, por lo que tiene 4 TB de información neta.

- RAID 4 tiene un disco adicional para mantener información de paridad: 5 discos en total.
- RAID 5 también mantiene información redundante equivalente a un disco, pero distribuida por todo el conjunto: 5 discos en total.
- RAID 6 tiene doble paridad distribuida, de tamaño equivalente a 2 discos: 6 discos en total.

(b) **0.2p** ¿Qué ventajas o inconvenientes tendría utilizar RAID 5 en este sistema sobre RAID 4?

La paridad en RAID 5 está distribuida, lo que evita un cuello de botella al acceder constantemente el disco de paridad de RAID 4 cada vez que se realiza una escritura. Esto también mejora el rendimiento de lecturas/escrituras concurrentes.

En cuanto a fiabilidad, ambos pueden recuperarse ante el fallo de un único disco cualquiera, reconstruyendo su información gracias al resto de información junto con la información de paridad. RAID 4 puede considerarse menos fiable porque el disco de paridad se escribe con mayor frecuencia (en cada escritura), lo que puede reducir su vida útil.

[**0.75p**] 6. Sea un computador con las siguientes características:

- Un sistema de memoria y de bus que soporta el acceso a bloques de 64 palabras de 32 bits cada una.
- Un bus síncrono de 64 bits a 2 GHz, en el que tanto una transferencia de 64 bits como el envío de una dirección de memoria requieren 1 ciclo de reloj.
- El tiempo de acceso a memoria para cada 4 palabras es de 1 ns.
- Las transferencias por el bus y los accesos a memoria pueden solaparse. Se supone que el bus está disponible antes de cada acceso.

Calcula la latencia y el ancho de banda para la lectura de 1024 palabras.

$T_{ciclo} = \frac{1}{f} = \frac{1}{2GHz} = \frac{1}{2}10^{-9}\ s = 0.5\ ns$

El sistema soporta bloques de 64 palabras. Por tanto, para transferir 1024 palabras, se requieren 1024 p / (64 p/transaccion) = 16 transacciones.

Para leer las 64 palabras de 1 transacción necesitamos:

- 1 ciclo para enviar la dirección.
- Un tiempo de acceso a memoria de 1 $ns/(0.5\ ns/ciclo) = 2\ ciclos$ para leer las 4 primeras palabras.
- 2 ciclos para enviar los datos del grupo de 4 palabras leído (el bus es de 64 bits, por lo que enviamos 2 palabras por ciclo), que se solapa con el tiempo de acceso del siguiente grupo de 4 palabras.
  - Como cada uno de estos grupos está formado por 4 palabras, necesitamos repetir 16 veces para transferir las 64 palabras que forman 1 transacción.

Por tanto, para la lectura de las 64 palabras de 1 transacción necesitamos $1 + 2 + 16 \times 2 = 35$ ciclos/transaccion

La latencia para la lectura de 1024 palabras será $16\ transacciones \times 35\ ciclos/tr \times 0.5\ ns/ciclo =$ 280 ns
Y el ancho de banda será $(1024\ palabras \times 4\ B/palabra)/(136 \times 10^{-9}\ s) = 14.63 \times 10^9\ B/s =$ 14.63 GB/s