Universidad Carlos III de Madrid

Departamento de Informática
Grado en Ingeniería Informática
Arquitectura de Computadores

Examen

ATENCIÓN:

- Lea atentamente todo el enunciado antes de comenzar a contestar.
- **La duración del examen es de 150 minutos.**

NOMBRE:
APELLIDOS:
NIA:

Ejercicio 1 [2 puntos]:

Sea el siguiente fragmento de código se encuentra almacenado a partir de la dirección de memoria 0x1000100C en una máquina en la que todas las instrucciones ocupan 4 bytes:

```
bucle:  lw $r2, 0($r0)
        addi $r3, $r2, 20
        sw $r3, 0($r1)
        addi $r0, $r0, 4
        addi $r1, $r1, 4
        bnez $r2, bucle
```

Dicho código se ejecuta en una máquina que dispone de una caché L1 para datos asociativa por conjuntos de 2 vías y con un tamaño de 32 KB y una caché L1 para instrucciones de iguales características. También dispone de una caché L2 unificada asociativa por conjuntos de 8 vías con un tamaño de 1 MB. En ambos casos el tamaño de línea es de 32 bytes. Se asume que un acierto en la caché L1 requiere 4 ciclos, y un acierto en la caché L2 requiere 14 ciclos adicionales y la penalización por traer un bloque de memoria principal a la caché de nivel 2 es de 80 ciclos. Todas las cachés tienen una política de escritura write-back.

Inicialmente el valor de los registros es:

- $r0: 0x00010000
- $r1: 0x00080000

A partir de la posición 0x00010000 todos los valores en memoria son distintos de cero hasta la posición 0x000100FC. En la posición de memoria 0x000100FC hay un valor de cero.

1.1: Determine cuál debería ser el tiempo medio de acceso asumiendo que un programa (distinto del facilitado) realiza en promedio 2 accesos a datos por instrucción y tiene las siguientes tasas de fallo:

- L1 instrucciones: 10%
- L1 datos: 5%
- L2: 2%

1.2: Determine el número de fallos que se producen durante la ejecución del fragmento de código facilitado en el enunciado para las cachés L1 de datos, L1 de instrucciones y L2.

1.3: Elabore un diagrama de tiempos para una arquitectura MIPS con un pipeline de 5 etapas para la primera iteración del bucle asumiendo que inicialmente no hay datos ni instrucciones en las cachés y con las siguientes consideraciones:

- No hay hardware de *forwarding*.
- La arquitectura permite que una instrucción escriba en un registro y otra instrucción lea ese mismo registro sin problemas.
- Las bifurcaciones se tratan vaciando el pipeline.
- La dirección efectiva de salto se calcula en la etapa de ejecución.

NOTA: Tenga en cuenta en el cronograma las detenciones debidas a los fallos en toda la jerarquía de caché tanto para instrucciones (etapa IF) como para lecturas y escrituras de datos (etapa M).

1.4: Repita el diagrama de tiempos para la segunda iteración.

SOLUCIÓN:

1.1: En cuanto a los accesos a la caché de nivel 1, se tiene que se realiza 2 accesos a datos por cada acceso a instrucciones. Por tanto la tasa de fallos se obtiene mediante media ponderanda:

m(L1) = (m(L1I)+2*m(L1D))/3

m(L1) = (0.1 + 2 * 0.05) / 3 = 0.2/3 = 0.0667

Por tanto el tiempo medio de acceso sería:

T = 4 + m(L1) * (14 + m(L2) * 80) = 4 + 0.0667 * (14 + 0.02 * 80) = 5.04 ciclos

1.2: El bucle se ejecuta un total de $2^8/4 = 2^6$ =64 iteraciones.

Analizaremos por separado los accesos de instrucciones y datos.

La primera instrucción se almacena en la dirección 0x1000100C. La última instrucción se almacena en la dirección en la dirección 0x1000100C + (6-1) *4 =0x10001020.

En la primera iteración, la primera instrucción genera un fallo de caché y trae el bloque de direcciones 0x10001000 – 0x1000101F, que contiene una instrucción de interés. Las siguientes instrucciones (I2, I3, I4, e I5) generan aciertos. Por último, la instrucción I6 vuelve a generar un fallo. Por tanto, la primera iteración genera 2 fallos y 4 aciertos. El resto de iteraciones generan aciertos en todos los casos.

Como el bucle se ejecuta 64 veces, el acceso a las instrucciones genera los siguientes accesos:

- Fallos L1I: 2
- Aciertos L1I: 4 + 63 * 6
- Fallos L2: 1
- Aciertos L2: 0

En cada iteración del bucle se lee una dirección de memoria en el rango 0x00010000 – 0x000100FC. Esto se corresponde con $2^8/2^5$=8 líneas de caché. Igualmente se escriben valores en las direcciones 0x00080000 – 0x 000800FC, que se escriben en 8 líneas de la caché. Como las cachés son asociativas por conjuntos no se dan conflictos entre los datos leídos y escritos. Como no se expulsa ninguna línea de la caché L1 no hay escrituras en la caché L2.

- Fallos L1D: 8 lecturas + 8 escrituras
- Aciertos L1D: 56 lecturas + 56 escrituras
- Fallos L2: 8 lecturas
- Aciertos L2: 0

1.3: Si se numeran las instrucciones de la siguiente forma:

```
bucle:  lw $r2, 0($r0)      #1
        addi $r3, $r2, 20   #2
        sw $r3, 0($r1)      #3
        addi $r0, $r0, 4    #4
        addi $r1, $r1, 4    #5
        bnez $r2, bucle     #6
```

Se tienen las siguientes dependencias RAW:

- $r2: I2 -> I1
- $r3: I3 -> I2
- $r0: I4 -> I1
- $r1: I3 -> I5

Al no haber forwarding, cuando hay una dependencia RAW debe esperarse al ciclo WB de la instrucción fuente para inicial el ciclo de la instrucción destino.

Instrucción	1-98	99	100	101	102	103-198	199	200	201	202	203	204	205
I1: lw $r2, 0($r0)	IF	ID	EX	M	M	M	WB						
I2: addi $r3, $r2, 20		IF1	IF2	IF3	IF4	stall	ID	EX	M	WB			
I3: sw $r3, 0($r1)							IF1	IF2	IF3	IF4	ID	EX	M
I4: addi $r0, $r0, 4											IF1	IF2	IF3
I5: addi $r1, $r1, 4													

Instrucción	206	207	208	209	210	211	212	213	214	215-302	303	304	305

I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20													
I3: sw $r3, 0($r1)	M	M	M	M	M	M	M	M	M	M	WB		
I4: addi $r0, $r0, 4	IF4	ID	EX	Stall	Stall	Stall	Stall	Stall	Stall	Stall	M	WB	
I5: addi $r1, $r1, 4		IF1	IF2	IF3	IF4	ID	stall	stall	stall	stall	EX	M	WB
I6: bnez $r2, bucle						IF	IF	IF	IF	IF	IF	IF	IF
I7:¿?													
I1: lw $r2, 0($r0)													

Instrucción	306	307	308	309	310	311	312						
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20													
I3: sw $r3, 0($r1)													
I4: addi $r0, $r0, 4													
I5: addi $r1, $r1, 4													
I6: bnez $r2, bucle	IF	IF	IF	ID	EX	M							
I7: ¿?				IF	flush								
I1: lw $r2, 0($r0)						IF							

- La primera instrucción se detiene 98 ciclos (80+14+4) en el fetch por ser un fallo en toda la jerarquía de memoria.
- La lectura de datos de la primera instrucción es un fallo de lectura y requiere 98 ciclos también.
- La segunda instrucción es un acierto y requiere cuatro ciclos para realizar la captación de la caché L1I.
- La instrucción I3 es un acierto y requiere cuatro ciclos para realizar la captación de la caché L1I.
- La escritura de datos de la instrucción I3 es un fallo de escritura y requiere 98 ciclos.
- La instrucción I4 es un acierto y requiere cuatro ciclos para realizar la captación de la caché L1I.
- La instrucción I4 no puede iniciar su ciclo de memoria hasta que no termine el acceso a memoria de I3.
- La instrucción I5 es un acierto y requiere cuatro ciclos para realizar la captación de L1I.
- La instrucción I5 no puede iniciar su ciclo de ejecución hasta que no termine la ejecución de I4.
- La instrucción I6 es un fallo y requiere 98 ciclos para realizar la captación de L1I.
- La instrucción I7 (la siguiente a la bnez) no puede comenzar la captación hasta que no se libera la unidad de fetch.
- Si bien se conoce la dirección de salto al final de la etapa de decodificación de I6, el sentido de salto (tomar o no tomar) no se conoce hasta el final de la etapa de ejecución.

En total se requieren 310 ciclos.

1.4: En esta iteración todos los accesos a instrucciones son aciertos. También todos los accesos a datos son aciertos.

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13
I1: lw $r2, 0($r0)	IF	IF	IF	IF	ID	EX	M	M	M	M	WB		
I2: addi $r3, $r2, 20					IF	IF	IF	IF	Stall	Stall	ID	EX	M
I3: sw $r3, 0($r1)											IF	IF	IF
I4: addi $r0, $r0, 4													
I5: addi $r1, $r1, 4													

Instrucción	14	15	16	17	18	19	20	21	22	23	24	25	26
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20	WB												
I3: sw $r3, 0($r1)	IF	ID	EX	M	M	M	M	WB					
I4: addi $r0, $r0, 4		IF	IF	IF	IF	ID	EX	M	WB				
I5: addi $r1, $r1, 4						IF	IF	IF	IF	ID	EX	M	WB
I6: bnez $r2, bucle										IF	IF	IF	IF

Instrucción	27	28	29	30									
I1: lw $r2, 0($r0)													
I2: addi $r3, $r2, 20													
I3: sw $r3, 0($r1)													
I4: addi $r0, $r0, 1													
I5: addi $r1, $r1, 1													
I6: bnez $r3, bucle	ID	EX	M	WB									
I7: ¿?	IF	flush											
I0: lw $r2, 0($r2)			IF										

En total se requieren 28 ciclos de reloj.

Ejercicio 2 [3 puntos]:

Dado los siguientes fragmentos de código que ejecutan 3 hilos empleando el protocolo de coherencia MSI.

```
// Código del hilo 0

for(i=0;i<16;i++){
    a[i]= 16;
 }
```

```
// Código del hilo 1
tmp=0;
for(i=0;i<16;i++){
    a[i]=tmp;
    tmp+=a[i];
 }
```

```
// Código del hilo 2
cnt=0;
for(i=0;i<4;i++){
    cnt+=b[i];
 }
```

El ordenador es una arquitectura CC-NUMA con:

- 2 procesadores de 32 bits con único nivel de memoria caché que es privado a cada procesador y tiene una correspondencia directa. El tamaño de línea caché es de 16 bytes.
- Las memorias caché están inicialmente vacías.
- Los hilos 0 y 2 se ejecutan en el procesador 0 mientras que el hilo 1 se ejecuta en el procesador 1.
- Las variables i, tmp y cnt se almacenan en registros (no se almacenan en memoria).
- El bloque que contiene a[0] está asociado a la misma línea caché que el bloque que contiene b[0].

Se pide rellenar las siguientes tablas y justificar la respuesta para los siguientes apartados:

-

- Partiendo de la situación inicial, indicar en la siguiente tabla las transiciones de estado del bloque que almacena a[0] cuando se ejecuta primero el hilo 0 completamente y a continuación el hilo 1. Indique el tráfico de bus asociado a cada hilo. Nota: en el caso de haber varias transiciones asociadas a un hilo, indique todas ellas.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0			
Hilo 1			

- Partiendo de la situación inicial, indicar en la siguiente tabla las transiciones de estado del bloque que almacena a[0] cuando se ejecuta primero el hilo 1 completamente y a continuación el hilo 0 Indique el tráfico de bus asociado a cada hilo. Nota: en el caso de haber varias transiciones asociadas a un hilo, indique todas ellas.

Código	Transición P0	Transición P1	Señales de bus
Hilo 1			
Hilo 0			

- Partiendo de la situación inicial, indicar en la siguiente tabla las transiciones de estado del bloque que almacena a[0] cuando se ejecuta primero el hilo 0 completamente, a continuación el hilo 2 completamente y finalmente el hilo 1 Indique el tráfico de bus asociado a cada hilo. Nota: en el caso de haber varias transiciones asociadas a un hilo, indique todas ellas.

Código	Transición P0	Transición P1	Señales de bus
Hilo 0			
Hilo 2			
Hilo 1			

- Para cada uno de los escenarios anteriores, indique el número de fallos caché existente para cada proceso.

Escenario	Fallos caché P0	Fallos caché P1
Hilo 0 -> hilo 1		
Hilo 1 -> hilo 0		
Hilo 0 -> hilo 2 -> hilo 1		

SOLUCIÓN:

Apartado A

Código	**Transición P0**	**Transición P1**	**Señales de bus**
Hilo 0	I -> E	I	Write miss
Hilo 1	E -> I	I -> E ; E -> E	Write miss; Write-back block

Apartado B

Código	**Transición P0**	**Transición P1**	**Señales de bus**
Hilo 1	I	I -> E ; E -> E	Write miss
Hilo 0	I -> E	E -> I	Write miss; Write-back block

Apartado C

Código	**Transición P0**	**Transición P1**	**Señales de bus**
Hilo 0	I -> E (almacena a[0])	I	Write miss
Hilo 2	E -> S (almacena b[0])	I	Read miss; write back block
Hilo 1	S -> S (b[0])*	I -> E ; E -> E	Write miss

Cada bloque tiene 4 palabras. Por lo que los hilos 0 y 1 acceden a 4 bloques y mientras que el hilo 2 accede a 1 bloque. En el hilo 1 la primera línea causa fallo caché mientras que la segunda, acierto. De este modo:

Escenario	Fallos caché P0	Fallos caché P1
Hilo 0 -> hilo 1	4	4
Hilo 1 -> hilo 0	4	4
Hilo 0 -> hilo 2 -> hilo 1	4 + 1	4

Ejercicio 3 [1 punto]

Dado el siguiente código paralelizado con OpenMP, y suponiendo que se disponen de 4 hilos (`export OMP_NUM_THREADS=4`) y que iter = 16:

```
1. #pragma omp parallel for private(j)
2. for (i = 0; i < iter; ++i) {
3.     for (j = iter - (i+1); j < iter; ++j) {
4.         //Función con carga computacional 2s
5.         compute_iteration(i, j, ...);
6.     }
7. }
```

Se pide:

a. Rellene en la siguiente tabla una posible asignación de iteraciones de la ejecución del bucle con índice 'i' con planificación estática, `schedule(static)`. Indique en la tabla qué hilo realiza cada iteración del bucle (cada valor distinto de 'i') y cuánto tiempo tarda dicha iteración. Calcule además el tiempo aproximado de ejecución por hilo y el tiempo de ejecución total.

#Iter	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hilo (ID)																
Tiempo (s)																

b. Rellene en la siguiente tabla una posible asignación de iteraciones ejecutando el bucle con índice 'i' con planificación dinámica y *chunk* 2, `schedule(dynamic, 2)`. Indique en la tabla qué hilo realiza cada iteración del bucle (cada valor distinto de 'i') y cuánto tiempo tarda dicha iteración. Indique además el tiempo aproximado de ejecución por hilo y el tiempo de ejecución total.

#Iter	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hilo (ID)																
Tiempo (s)																

c. Justifique cuál de las planificaciones anteriores sería la mejor para un caso genérico (número variable de iteraciones y de hilos).

SOLUCIÓN:

a. Static

	It 0	It 1	It 2	It 3	It 4	It 5	It 6	It 7	It 8	It 9	It 10	It 11	It 12	It 13	It 14	It 15
Hilo	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3

Tiempo	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

Tiempo por hilo:
- Hilo 0: 20s
- Hilo 1: 52s
- Hilo 2: 84s
- Hilo 3: 116s

Tiempo total: 116s

b. Dynamic

	It 0	It 1	It 2	It 3	It 4	It 5	It 6	It 7	It 8	It 9	It 10	It 11	It 12	It 13	It 14	It 15
Hilo	0	0	1	1	2	2	3	3	0	0	1	1	2	2	3	3
Tiempo	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

Tiempo por hilo:
- Hilo 0: 44s
- Hilo 1: 60s
- Hilo 2: 76s
- Hilo 3: 92s

Tiempo total: 92s

c. Dynamic

Independientemente del número de iteraciones y de hilos, dado que la carga de trabajo por iteración no es uniforme, se adaptará mejor la planificación dinámica que la estática. Se podría aceptar como respuesta que le mejor planificador es guided.

Ejercicio 4 [1 punto]

Se proporciona el siguiente código programado con atómicos. En el punto A, head contiene un 8 y se intenta insertar un 9, por lo que se imprimirá la salida **"8 8 9"**. Si otro hilo intenta insertar un 10 de forma concurrente, indique qué datos se imprimirán por pantalla si se ejecuta la parte B (a partir de la sentencia de la línea 16), y qué datos si se llega a ejecutar la parte C (a partir de la sentencia de la línea 25).

```
1.  struct node
2.  {
3.          std::shared_ptr<T> data;
4.          node* next;
5.          node(T const& data_):data(new T(data_)), next(nullptr) {}
6.  };
7.  std::atomic<node*> head;
8.  void push(T const& data)
9.  {
10.         node* const new_node=new node(data);
11.         new_node->next=head.load();
12.
13.         //A
14.         std::cout << *(head.load()->data) << " "; // 8
15.         std::cout << *(new_node->next->data) <<  " "; // 8
16.         std::cout << *(new_node->data) << std::endl; // 9
17.
18.         if(head.compare_exchange_strong(new_node->next,new_node)) {
19.             //B
20.             std::cout << *(head.load()->data) << " ";
21.             std::cout << *(new_node->next->data) <<  " ";
```

```
22.          std::cout << *(new_node->data) << std::endl;
23.       } else {
24.          //C
25.          std::cout << *(head.load()->data) << " ";
26.          std::cout << *(new_node->next->data) <<  " ";
27.          std::cout << *(new_node->data) << std::endl;
28.       }
29.}
```

SOLUCIÓN:

Caso B: Mensaje por pantalla “9 8 9”

Caso C: Mensaje por pantalla “10 10 9”

No es posible ninguna otra solución, ya que, compare_exchange_strong no admite fallos espurios.