Universidad Carlos III de Madrid

Departamento de Informática
Grado en Ingeniería Informática
Arquitectura de Computadores

Examen final convocatoria ordinaria
16 de Mayo de 2012

ATENCIÓN:

- Lea atentamente todo el enunciado antes de comenzar a contestar.
- No se podrán utilizar libros ni apuntes, ni calculadoras de ningún tipo.
- Los teléfonos móviles deberán permanecer desconectados durante la prueba (apagados, no silenciados).
- Solamente se corregirán los ejercicios contestados con bolígrafo. Por favor no utilice lápiz.
- **Las duración del examen es de 2 horas y treinta minutos.**

NOMBRE:

APELLIDOS:

NIA:

Ejercicio 1: (1 punto)

Explique en qué consiste las optimizaciones de memoria caché *palabra crítica primero* y *reinicio temprano*.

Solución:

Estas optimizaciones se aplican en fallos caché cuando se transfiere un bloque de memoria a la memoria caché. La idea de la optimización consiste en transferir la palabra que necesita al procesador antes de que todo el bloque haya sido enviado a la cache, reduciendo así la latencia de espera. Existen dos alternativas: palabra crítica primero y reinicio temprano.

- **Palabra crítica primero: el bloque se envía reordenado con la palabra que necesita el procesador al principio.**
- **Reinicio temprano: el bloque se envía sin reordenar y tan pronto se recibe la palabra necesitada por el procesador, éste prosigue (sin esperar a que finalice la transferencia del bloque).**

Ejercicio 2: (1 punto)

¿Qué alternativa es más escalable? ¿Un protocolo de *snooping* o uno basado en directorio? Justifique la respuesta.

Los protocolos de snooping observan la actividad del bus y ejecutan, mediante broadcasts, los pasos necesarios para mantener la coherencia. Es costoso en terminos de bandwitth. Quien escribe avisa con un broadcast y todos los procesadores sondean el bus para detectar modificaciones que le incumban.
Esta forma de trabajo produce que cuando más procesadores estén conectados al bus producirá más ancho de banda y mayor probabilidad de saturación del bus (menos escalable)
En los protocolos de directorio los comandos de actualización se envían sólo a los caches afectados por una actualización. Almacena información en un directorio acerca de dónde se encuentran las copias de los bloques. Cuando un procesador quiere escribir una posición, debe solicitar autorización al controlador, quien luego invalida las demás copias. El directorio puede estar centralizado (probabilidad de que ocurra un cuello de botella) o distribuido (cada cache sabe lo que tiene). Esta última solución produce menos tráfico global resultante y por tanto la arquitectura es más escalable.

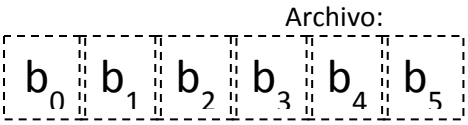
Universidad Carlos III de Madrid

Departamento de Informática
Grado en Ingeniería Informática
Arquitectura de Computadores

Examen final convocatoria ordinaria
16 de Mayo de 2012

Ejercicio 3: (1 punto)

Dado un archivo que consta de 6 bloques de 16KB. Explique usando una figura el modelo de almacenamiento RAID-5 con 4 discos e indique al menos una distribución posible de los bloques del archivo en cada uno de los discos del RAID.



Solución:

La distribución de bloque en un RAID 5 es la siguiente: los bloques del fichero están distribuidos en todos los discos mediante una distribución *round-robin*. La paridad está también distribuida en todos los discos, esto elimina el cuello de botella del RAID 4, para el que la paridad estaba en un solo disco.

D_0	D_1	D_2	D_3
b_0	b_1	b_2	P_{012}
b_3	b_4	P_{345}	b_5

Ejercicio 4: (1 punto)

Explique y describa qué se conoce como mecanismo de arbitración de buses y sus posibles configuraciones.

Solución:

El árbitro de bus determina qué dispositivo puede escribir en el bus en cada momento. La lectura del bus no debe ser arbitrada, dado que todos los dispositivos pueden leer simultáneamente.

Existen dos tipos de arbitración: centralizada y distribuida. En la centralizada un único dispositivo controla el acceso a bus. En la distribuida, cada módulo puede reclamar el acceso al bus y todos ellos (de forma conjunta) deciden quién accede en cada instante. La lógica de control está distribuida en todos los módulos.

Ejercicio 5: (2 puntos)

Dada la siguiente sección de código:

```
I1              DADDUI R3, R1, #40
I2    BUCLE:    L.D F0, 0(R1)
I3              L.D F2, 0(R2)
I4              ADD.D F4, F0, F2
```

```
I5          S.D F4, 0(R1)
I6          DADDUI R1, R1, #8
I7          DADDUI R2, R2, #8
I8          BLE R1, R3, BUCLE
```

Y teniendo en cuenta que se ejecuta en una máquina con las siguientes latencias adicionales entre instrucciones:

Instrucción que produce el resultado (previa)	Instrucción que usa el resultado (posterior)	Latencia
Operación ALU FP	Operación ALU FP	5
Operación ALU FP	Cargar/Almacenar un doble	4
Operación ALU FP	Instrucción de salto	4
Cargar doble	Operación ALU FP	2
Cargar doble	Cargar doble	1
Almacenar doble	Operación ALU FP	2

La instrucción de salto (*branch)* tiene una latencia de un ciclo y no tiene *delay slot*.

Suponga además que la máquina en la que ejecuta es capaz de emitir una instrucción por ciclo, salvo las esperas debidas a detenciones y que es un procesador segmentado con un único camino de datos (*pipeline*).

Se pide:

a) Determine todas las dependencias de datos.

Se producen las siguientes dependencias:

I4 → I2 (F0: RAW), I4 → I3 (F2: RAW)
I5 → I4 (F4: RAW)
I6 → I1 (R1: WAR), I6 → I2 (R1: WAR), I6 → I5 (R1: WAR)
I7 → I3 (R2: WAR)
I8 → I7 (R2: RAW), I8 → I1 (R3: RAW)

b) Determine el número de ciclos total que se necesita para ejecutar la sección de código completa.

```
            DADDUI R3, R1, #40 ← Solamente la primera vez
BUCLE:      L.D F0, 0(R1)
            L.D F2, 0(R2)
            Detención
            Detención
            ADD.D F4, F0, F2
            Detención
            Detención
            Detención
            Detención
            S.D F4, 0(R1)
```

```
DADDUI R1, R1, #8
DADDUI R2, R2, #8
BLE R2, R3, BUCLE
```

Se necesita un ciclo para el código de iniciación. Cada iteración necesita 13 ciclos. Como el bucle se ejecuta 5 veces, se tiene un total de 1+5 * 13 = 66

Debido a que no existe *delay slot* no es necesario añadir un ciclo de stall después de la instrucción de salto (BLE).

c) Modifique el código para reducir las detenciones mediante la técnica de planificación de bucle. Determine la aceleración obtenida respecto a la versión sin planificar (apartado b).

```
        DADDUI R3, R1, #40 ← Solamente la primera vez
BUCLE   L.D F0, 0(R1)
        L.D F2, 0(R2)
        DADDUI R1, R1, #8
        DADDUI R2, R2, #8
        ADD.D F4, F0, F2
        Detención
        Detención
        Detención
        Detención
        S.D F4, -8(R1)
        BLE R1, R3, BUCLE
```

Ahora se requiere un tiempo total de 1+5*11 = 56. De nuevo, se necesita un ciclo para el código de iniciación. Cada iteración necesita 13 ciclos. Debido a que no existe *delay slot* no es necesario añadir un ciclo de stall después de la instrucción de salto (BLE).

Speedup = 66/56 = 1,17

d) Modifique el código realizando un desenrrollamiento de bucle con dos iteraciones por bucle. Determine la aceleración obtenida respecto a la versión sin planificar (apartado b).

```
        DADDUI R3, R1, #32 ← Solamente la primera vez
BUCLE   L.D F0, 0(R1)
        L.D F2, 0(R2)
        Detención
        Detención
        ADD.D F4, F0, F2
        Detención
        Detención
        Detención
```

```
                Detención
                S.D F4, 0(R1)
                L.D F6, 8(R1)
                L.D F8, 8(R2)
                Detención
                Detención
                ADD.D F10, F6, F8
                Detención
                Detención
                Detención
                Detención
                S.D F10, 8(R1)
                DADDUI R1, R1, #16
                DADDUI R2, R2, #16
                BLE R2, R3, BUCLE
L.D F0, 0(R1)
L.D F2, 0(R2)
Detención
Detención
ADD.D F4, F0, F2
Detención
Detención
Detención
Detención
S.D F4, 0(R1)
```

En total ahora se tienen 2 iteraciones dentro del bucle. Además la quinta iteración del bucle original se hace ahora al final. El tiempo requerido es 1 + 2 * 23 + 10 = 57

Si además se planifica, se puede tener:

```
                DADDUI R3, R1, #32 ← Solamente la primera vez
BUCLE           L.D F0, 0(R1)
                L.D F2, 0(R2)
                L.D F6, 8(R1)
                L.D F8, 8(R2)
                ADD.D F4, F0, F2
                Detención
                ADD.D F10, F6, F8
                DADDUI R1, R1, #16
                DADDUI R2, R2, #16
                S.D F4, -16(R1)
                Detención
                S.D F10, -8(R1)
                BLE R2, R3, BUCLE
L.D F0, 0(R1)
L.D F2, 0(R2)
```

Detención
Detención
ADD.D F4, F0, F2
Detención
Detención
Detención
Detención
S.D F4, 0(R1)

El tiempo pasa a 1+ 2 * 13 + 10 = 37 ciclos

Ejercicio 6: (2.5 puntos)

Un computador dispone de tres procesadores conectados por un bus a un sistema de memoria compartida. El computador utiliza el protocolo de coherencia de caché MSI**.** El coste de una invalidación es de 2 bytes. El coste de un fallo caché es de 16 Bytes para solicitar un bloque y 64 bytes para transferirlo. La siguiente tabla muestra la secuencia de instrucciones ejecutadas por cada procesador a lo largo del tiempo.

Orden	Procesador y acción	Transición P1	Transición P2	Transición P3	Acción Bus	Tráfico bus
1	P1: Lectura x					
2	P2: Escritura x					
3	P2: Escritura x					
4	P3: Escritura x					
5	P1: Lectura x					
6	P2: Lectura x					

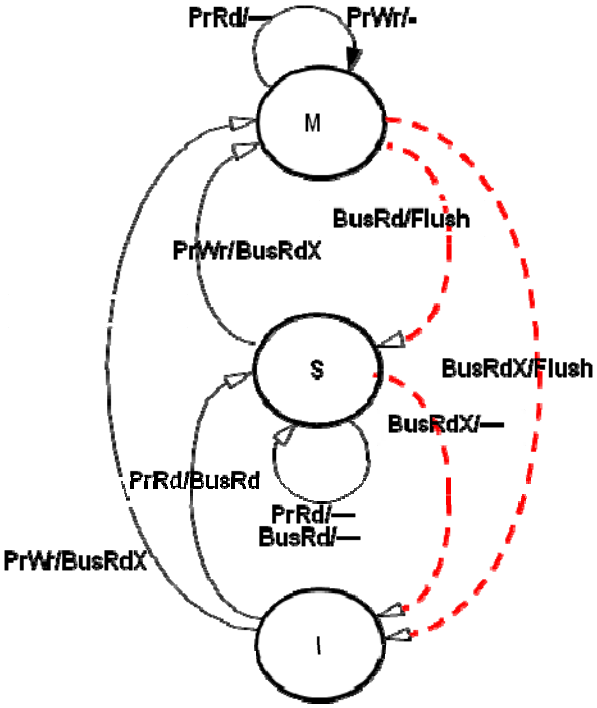
Universidad Carlos III de Madrid

Departamento de Informática
Grado en Ingeniería Informática
Arquitectura de Computadores

Examen final convocatoria ordinaria
16 de Mayo de 2012

Se pide:

1. ¿Qué tipo de política de escritura entre caché y memoria aplica el protocolo de coherencia de caché MSI? ¿escritura directa (*write through*) o postescritura (*write back*)? Razone la respuesta.
2. Para la secuencia dada de acciones indique todas las transiciones ligadas al protocolo MSI, todas las acciones del bus, y el tráfico de bus generado en bytes rellenando la tabla adjunta. **Inicialmente todos los procesadores tienen una copia del bloque que contiene x en su caché.** Para indicar la transición marque el estado inicial y final realizado en cada caso. Por ejemplo: I -> S. Si no hay transición, indique el estado en el que se encuentra el bloque cache. Por ejemplo M.



Solución

1.- WriteBack es la politica de escritura asociada a un protoclo MSI ya que permite reducir el ancho de banda generado. Por ejemplo: Escrituras consecutivas sobre bloques de caché en estado Modified no producen una transacción de bus hasta que ese bloque es requerido por otro procesador. En ese momento es en el que se actualiza además el bloque en memoria.

Tiempo	Transición P1	Transición P2	Transición P3	Acción Bus	Tráfico bus
1 seg.	S -> S	S	S	-	-
2 seg.	S -> I	S -> M	S -> I	BusRdX	4B
3 seg.	I	M	I	-	-
4 seg.	I	M -> I	I -> M	BusRdX Flush	16+64+4
5 seg.	I -> S	I	M -> S	BusRd Flush	16 + 64
6 seg.	S	I -> S	S	BusRd	16 + 64

Ejercicio 7: (1.5 punto)

Dada la siguiente clase que implemente un *spin-lock*:

```
class spinlock_mutex {
private:
  std::atomic_flag f;
public:
  spinlock_mutex() : f(ATOMIC_FLAG_INIT) {}
  void lock() {
    while (f.test_and_set(std::memory_order_acquire)) {}
  }
  void unlock() {
    flag.clear(std::memory_order_release);
};
```

Explique cuál será el efecto (y razone por qué) de usarla para sincronizar un programa multi-hilo en los siguientes casos:

a) El programa lanza 8 hilos en un procesador con 8 núcleos.
b) El programa lanza 256 hilos en un procesador con 8 núcleos.

Solución:

a) El código corresponde con una posible implementación de un mutex mediante mecanismos atómicos. Para ello se hace uso de la variable atómica *f*, la cual, mediante el método *test_and_set* gestionará la entrada en una sección crítica. El mecanismo de parada es inicialmente lento (PARADA ACTIVA). Sin embargo el uso de variables ATÓMICAS permite liberar de carga al sistema operativo de la pesada carga de la condición de bloqueo.

En este caso, al contar con un proceso ligero por núcleo la sobrecarga de la parada activa es pequeña.

b) Partiendo del razonamiento anterior, en este caso contamos con un mayor número de procesos ligeros por núcleo en el sistema, 32 hilos/núcleo. En este caso tendremos que la sobrecarga con 32 hilos por núcleo es mucho mayor que en el caso anterior. Sin embargo esta sobrecarga es reducida mediante lo implementación atómica frente a una implementación basada en MUTEX TRADICIONALES.