Arquitectura e Ingeniería de Computadores - 3er Curso. **Examen 1er parcial** **6/11/2017**

Responde cada pregunta en una hoja distinta. Tiempo disponible: 3h

1. **(2.5 puntos)** Se desean mejorar las prestaciones de un dispositivo móvil que dispone de un procesador Qualcomm a 1.4GHz y una GPU Mali-G71 MP8, con un coste de producción de 300€. Para ello se desea evaluar desde el punto de vista coste/prestaciones diferentes alternativas.

   Para evaluar las alternativas se empleará un software de test que da un tiempo de ejecución de 10.5 segundos en el dispositivo original. Se comprueba que dicho software emplea un 55 % del tiempo en operaciones con la CPU. El resto del tiempo realiza operaciones con la GPU y otros dispositivos.

   En una primera prueba, cambiando la GPU por una Mali-G71 MP20, que ofrece una mejora del 150 % sobre la GPU original, se obtiene un tiempo de ejecución para el software de test de 8.9 segundos. El incremento de coste de esta nueva GPU es de 50€.

   Responder a las siguientes cuestiones.

   a) ¿Cuál es la aceleración global obtenida al reemplazar la GPU?

   b) En el dispositivo original, ¿qué fracción de tiempo empleaba el software de test en operaciones con la GPU? ¿Y en operaciones con otros dispositivos?

   c) Si se sustituye el procesador del dispositivo original por uno con la misma arquitectura, pero a 2.0GHz y con un aumento del coste de 70€. ¿Qué aceleración global se obtendrá?

   d) ¿Que aceleración global se obtendrá si aplicamos simultáneamente en el dispositivo original los cambios propuestos en los dos apartados anteriores? Considera que este cambio aumenta el coste en 100€.

   e) Finalmente, desde el punto de vista coste/prestaciones. ¿Qué alternativas son interesantes: cambiar la GPU, el procesador o ambos?

2. **(2.5 puntos)** Sobre un microcontrolador Infineon ARM Cortex M4 con un reloj a 200 Mhz y con juego de instrucciones load/store se ejecutan varias tareas concurrentes. Dichas tareas comparten recursos y para sincronizar el acceso a los mismos se emplean operaciones *test-and-set* en las que se garantiza la atomicidad inhabilitando las interrupciones.

```
                  # test-and-set sobre la
                  # variable en memoria: lock
                  #
di                # inhabilita interrupciones
addi r2,r0,#1     # r2 <- 1
lb r1,lock(r0)    # lee valor sobre r1
sb r2,lock(r0)    # lock <- 1
ei                # habilita interrupciones
```

   Tras estudiar la carga que generan las tareas se obtiene la siguiente distribución de operaciones:

Operación	%	CPI
ALU	36	1
Load	24	2
Store	12	2
Saltos	18	1.5
di	4	1
ei	6	1

   Los ingenieros se están planteando introducir una modificación en la arquitectura para que la operación *test-and-set* se implemente en una sola instrucción. De este modo todo el código anterior se sustituiría por:

```
TaS r1,lock(r0) # cargar en r1 el valor de la posición look
                # y luego asigna 1 a dicha posición
```

dicha instrucción tendría un CPI de 3. La frecuencia de reloj no cambia.

a) Calcular el CPI del procesador original, así como el tiempo de ejecución en segundos de una tarea formada por *n* instrucciones.

b) Sabiendo que en el procesador original el 50 % de las instrucciones *di* se emplean en operaciones *test-and-set*. Calcular la nueva distribución de instrucciones en el procesador modificado.

c) Sobre el procesador modificado calcular el CPI, así como el número de instrucciones que tendría una tarea que en el procesador original tenía *n* instrucciones.

d) Cuantifica la aceleración proporcionada por la nueva arquitectura.

3. **(2.5 puntos)** Se dispone de un procesador MIPS en el cual se ejecuta el siguiente bucle:

```
loop:   l.d f1, 0(r1)
        l.d f2, 0(r2)
        add.d f4, f1, f7
        add.d f5, f2, f8
        div.d f7, f6, f1
        div.d f8, f6, f2
        mul.d f5, f4, f5
        sub.d f5, f5, f10
        s.d f5, 0(r3)
        daddi r1, r1, 8
        daddi r2, r2, 8
        daddi r3, r3, 8
        bne r1, r5, loop
        trap 0
        <sgte1>
        <sgte2>
        <sgte3>
```

El procesador dispone de los siguientes operadores multiciclo de coma flotante:

- Sumador/Restador. Lat= 2, IR= 1, etapas A1, A2.
- Multiplicador. Lat= 3, IR= 1, etapas M1, M2, M3.
- Divisor. Lat= 5, IR=$\frac{1}{5}$, etapas D1, D2, D3, D4, D5.

Los riesgos estructurales y de datos se detectan en la fase ID, insertando tantos ciclos de parada como sean necesarios y, en el caso de los riesgos de datos, utilizando cortocircuitos siempre que sea posible.

Los riesgos de control se resuelven mediante la técnica predict-not-taken. El cálculo de la condición de salto, el de la dirección destino, y la escritura del PC, se realizan en la etapa `ID`.

Teniendo en cuenta que las operaciones de enteros y de memoria utilizan las 5 etapas clásicas: `IF, ID, EX, ME` y `WB`, mientras que las instrucciones multiciclo las etapas `IF, ID,` <ejecución en el operador multiciclo correspondiente> y `WB`, y que se dispone de dos bancos de registros (1 flotante y 1 entero) con 2 puertos de lectura y 1 puerto de escritura cada banco, se pide:

a) El diagrama instrucciones-tiempo de la primera iteración, incluyendo la instrucción que se ejecuta después de la `bne`.

b) Si en el apartado 3*a* ha sido necesario introducir ciclos de parada debido a riesgos, identifique para cada caso el tipo de riesgo y las instrucciones involucradas explicando el motivo de la detención.

c) A partir del diagrama anterior, indique el tiempo de ejecución de una iteración (en ciclos) cuando el predictor acierta y cuando el predictor falla.

d) ¿Que ocurriría si este mismo código (**sin ninguna modificación**) se ejecutara en un procesador que resolviera los riesgos de control mediante la técnica del salto retardado? Razone la respuesta.

4. **(2.5 puntos)**

El siguiente programa copia las componentes distintas de cero del vector fuente al vector destino.

```
i = 0;
j = 0;
n = 10;
do
{
   if (fuente[i] != 0) /* Salto b1 */
   {
      destino[j] = fuente[i];
      j++;
   }
   i++;
   n--;
}
while (n != 0) /* Salto b2 */
```

Este programa se traduce por un compilador en el siguiente código ensamblador MIPS64.

```
      daddi r1,r0,0   # r1 = Dirección del vector fuente
      daddi r2,r0,80  # r2 = Dirección del vector destino
      daddi r3,r0,10  # r3 = Guarda del bucle (n)
loop: ld r10,0(r1)
      daddi r1,r1,8
      beqz r10,fi     # Salto b1 (salta si fuente[i] == 0)
      sd r10,0(r2)
      daddi r2,r2, 8
fi:   daddi r3,r3,-1
      bnez r3,loop    # Salto b2 (salta si n != 0)
      trap 0
      nop
      nop
      nop
```

Este código se ejecuta en un procesador MIPS64 segmentado en las 5 etapas habituales, resultando en el siguiente diagrama de instrucciones-tiempo para la primera iteración del bucle.

```
                        1  2  3  4  5  6  7  8  9 10 11 12 13 14
loop: ld r10,0(r1)      IF ID EX ME WB
      daddi r1,r1,#8       IF ID EX ME WB
      beqz r10,fi             IF ID EX ME WB
      sd r10,0(r2)               IF ID X
      daddi r2,r2,#8                IF X
fi:   daddi r3,r3,-1                   IF ID EX ME WB
      bnez r3,loop                        IF ID EX ME WB
      trap #0                                IF ID X
      nop                                       IF X
loop: ld r10,0(r1)                                 IF ID EX ME WB
```

Teniendo en cuenta que el procesador cuenta con un predictor dinámico BTB con 1 bit para la condición de salto, y asumiendo que la BTB se encuentra vacía al principio de la ejecución del programa, se pide, **razonando las respuestas**:

a) ¿En qué etapa se actualiza el PC en caso de salto efectivo? ¿En qué etapa se calcula la condición de salto?

b) Completa en la hoja adjunta la traza de la ejecución del salto `b2`. Con respecto a este salto, ¿cuántas veces acertará el predictor?

c) Asumiendo que el vector fuente contiene las componentes $0, 1, 0, 1, 0, 1, 0, 1, 0, 1$, completa en la hoja adjunta la traza de la ejecución del salto `b1`. Con respecto a este salto, ¿cuántas veces acertará el predictor?

d) Se ha decidido sustituir el predictor de un bit por otro de dos bits con histéresis y estados *Stronly Not Taken* (SNT), *Weakly Not Taken* (WNT), *Weakly Taken* (WT), y *Strongly Taken* (ST). Asumiendo que antes de ejecutar el programa la entrada en la BTB del salto `b1` se encuentra en el estado SNT y que el vector fuente contiene las mismas componentes que el apartado anterior, ¿cuántas veces acertará el predictor con respecto a salto `b1`?

Apellidos y Nombre:	

Ejercicio 3

a) Diagrama instrucciones-tiempo de la primera iteración, incluyendo la instrucción que se ejecuta después de la `beqz`.

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
dadd r1, r0, x																														

Apellidos y Nombre:	

Ejercicio 4

b) Traza de la ejecución del salto `b2`.

Iteración	01	02	03	04	05	06	07	08	09	10
Sí Salta/No Salta										
Predicción (Sí/No)										

c) Traza de la ejecución del salto `b1`.

Iteración	01	02	03	04	05	06	07	08	09	10
fuente[i]	0	1	0	1	0	1	0	1	0	1
Sí Salta/No Salta										
Predicción (Sí/No)										

d) Traza de la ejecución del salto `b1` con predictor de dos bits con histéresis.

Iteración	01	02	03	04	05	06	07	08	09	10
fuente[i]	0	1	0	1	0	1	0	1	0	1
Sí Salta/No Salta										
SNT/WNT/WT/ST										
Predicción (Sí/No)										