ARCO Pràctica 2 - Informe

Ixent Cornella

SESIÓN 3

8)

a) **¿Cuántas iteraciones hace el bucle? ¿Qué hace el código? ¿En qué posiciones de memoria escribe y qué valores?**

El bucle hace un total de 49 iteraciones.

El código consiste en, dados dos vectors de words X (16 words) e Y (espacio para 16 words también), suma a los 4 últimos elementos de X (X[12], X[13], X[14], X[15]) 10, y lo guarda en las posiciones de memoria de Y correspondientes a Y[4], Y[3], Y[2] e Y[1]. Esto es debido a que se accede a los datos de X desde X[12] (posición de memoria equivalente a 0x10010030) en orden ascendente, mientras que el resultado se guarda desde Y[4] (0x10010050) en orden descendente. La condición del bucle es que las posiciones de memoria de X e Y no coincidan, y se de este caso a la 4ª iteración. Por tanto, el código escribe:

Y[4] = 10, **Y[3]** = 15, **Y[2]** = 9, **Y[1]** = 10.

O, equivalentemente:

0x10010050 = 10, **0x1001004c** = 15, **0x10010048** = 9, **0x10010044** = 10,

b) **Cronograma**

El cronograma queda adjunto en la tarea como “Crongrama 8b”.

El código tarda en ejecutarse 6+3+10N ciclos = 10N + 9 ciclos. En nuestro caso, N = 4, por lo tanto tenemos que tarda 40 + 9 = 49 ciclos, que es justo lo que da el simulador.

9)

a) ¿Qué ocurre cuando intentas ejecutar? ¿Por qué crees que ocurre? Para darte cuenta de qué pasa ejecuta paso a paso hasta llegar al salto bne, ¿qué ha cambiado respecto a la configuración anterior?

Al intentar ejecutar el código, salta una excepción: Acceso a memoria no válido.

Esto se debe a la naturaleza de salto retardado y como está estructurado el código, ya que después del **bne** hay un addi $t0, $zero, 0 (de relleno), y al ser salto retardado, esta instrucción se ejecuta igualmente, haciendo que el lw que usa $t0 acceda a una posición inválida de la memoria.

b) Para solucionar el problema pon una instrucción de dentro del bucle detrás de bne. ¿Cuál es la única instrucción posible de poner para que el código haga lo mismo? Si es necesario modifica ligeramente la instrucción movida.

De hecho, hay 3 instrucciones que podrían ser movidas para que funcione el código.

sw $s3, 0($t1)

addi $t0, $t0, 4

addi $t1, $t1, -4

De estas 3, el **sw** es la más viable, ya que tan solo hay que cambiar el inmediato para que el código haga lo mismo. En las otras habría que hacer más modificaciones. Por lo tanto, el código del bucle quedaría así trás la modificación:

```
lw $s1, 0($t0)

addi $s3, $s1, 10

addi $t0, $t0, 4
```

```
addi $t1, $t1, -4

bne $t0, $t1, bucle

sw $s3, 4($t1)

addi $t0, $zero, 0

addi $t1, $zero, 0

addi $s3, $zero, 0

addi $s1, $zero, 0
```

c) Vuelve a ejecutar ahora asegurándote que el resultado es correcto, ¿cuántos ciclos tarda? ¿Tarda menos? Si es así, ¿por qué tarda menos? Haz un cronograma de las dos primeras iteraciones

El programa ahora tarda 3 ciclos menos, 46, resultado que parece tener sentido ya que hay 4 iteraciones del bucle (la última iteración ya aprovechaba la última instrucción). Es por esto que, al aprovechar la instrucción posterior al **bne,** apreciamos esta reducción de 3 ciclos. El resultado del cronograma tiene sentido, ya que nos da **9N + 10**, que para N = 4, da 46.

El cronograma queda adjunto en la tarea como **Cronograma 9c**

Fin del INFORME.