UNIVERSIDAD CARLOS III DE MADRID

Trabajo 1

Departamento de Ingeniería de Sistemas y Automática

CESAR AUGUSTO ARISMENDI GUTIERREZ
RAÚL PÉRULA MARTÍNEZ
LUIS ENRIQUE MORENO LORENTE
ALBERTO BRUNETE GONZALEZ
DOMINGO MIGUEL GUINEA GARCIA ALEGRE
JOSÉ CARLOS CASTILLO MONTOYA

Universidad Carlos III de Madrid

Enunciado

El objetivo de este trabajo es programar variantes del bucle SAXPY en ensamblador. Este bucle implementa la operación vectorial Y=a*X+Y.

El código del bucle es el siguiente:

```
   Loop:
1       l.d   f2,0(r1)  ; load x(i)
2       mul.d f4,f2,f0  ; multiply a*X(i)
3       l.d   f6,0(r2)  ; load Y(i)
4       add.d f6,f4,f6  ; add a*X(i)+Y(i)
5       s.d   f6,0(r2)  ; store Y(i)
6       daddi r1,r1,8   ; increaseindex X
7       daddi r2,r2,8   ; increaseindex Y
8       daddi r3,r3,-1  ; decreasecounter
9       bnez  r3,Loop   ; checkfortermination
```

[A mano]

1. Obtener el grafo de dependencias de las instrucciones del bucle.
2. Determinar el cronograma de una iteración del bucle:
   a) Con interbloqueo hardware sin *forwarding* (sin predicción de salto).
   b) Con interbloqueo hardware y *forwarding* con técnica de salto retardado.
   c) Con planificación dinámica usando el algoritmo de *Scoreboard* y predicción de salto dinámico.

[Con WINMIPS64]

Latencias WinMPIS64:

Suma	2
Multiplicación	5

3. Completar el programa con las directivas e instrucciones necesarias para reservar e inicializar la memoria (array de12 valores *double* con la directiva *.double* que se muestran a continuación) para los vectores X y Y. Cargar las direcciones de X y Y en los registros r1 y r2, el valor (*double*) 2.0 en f0, y el valor entero 12 en r3. Comprobar el programa con **WinMIPS64** analizando el contenido de la memoria.
4. Indicar los valores del vector Y al final de la ejecución del programa.
   Tenga en cuenta que el valor en la memoria de datos está en formato hexadecimal doble. Al hacer doble clic con el botón derecho del ratón, se muestra el mismo número en formato decimal.

X = [12.0, 11.0, 10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0]

Y = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.10, 0.11, 0.12]

5. Calcular el CPI del programa con:
   a) Con *forwarding*.
   b) Sin *forwarding*.

A partir de aquí se supone *forwarding* y que el programa siempre tiene que dar el resultado correcto.

6. Calcular el CPI del programa con:
   a) *Branch Target Buffer*.
   b) *Delay Slot*. ¿Qué ocurre? ¿Hay que modificar el programa?
7. Desenrollar el bucle para que cada nueva iteración corresponda con 4 del código actual. Comprobar que el programa sigue funcionando y dando el mismo resultado.
8. Calcular el nuevo CPI y el *speedup* con:
   a) Branch Target Buffer.
   b) Delay Slot
9. Reordenar las instrucciones del bucle y renombrar los registros para optimizar aun más el código. Comprobar que el programa sigue funcionando y dando el mismo resultado. Calcular finalmente el nuevo CPI y el *speedup* con:
   a) Branch Target Buffer.
   b) Delay Slot

Nota: Entregar los ficheros *.s* realizados en los distintos apartados.