EJERCICIO MÓDULO II. SIMULACIONES CON WINMIPS64

Guarde este código en un archivo llamado Codigo.s para simular en WINMIPS64:

```
; Add values from two vectors.
; The result is stored in F6 register.

.data

X:      .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
        .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
Y:      .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
        .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09

.text

main:  daddi r3,r0,20          ; r3 <-- Number of vector components to process
       daddi r4,r0,0           ; r0 <-- cont
       daddi r1,r0,X       ; r1 will be my base register X
       daddi r2,r0,Y       ; r2 will be my base register Y

loop:
       l.d f0, 0(r1)     ; F0 <-- X(i)
       l.d f2, 0(r2)     ; F2 <-- Y(i)
       add.d f4,f0,f2  ; F4 <-- X(i) + Y(i)
     add.d f6,f6,f4  ; F6 <-- F6+f4

       daddi r1,r1,8  ; X(i+1)
       daddi r2,r2,8  ; Y(i+1)
       daddi r4,r4,1  ; Y(i+1)
       bne r4,r3,loop ; branch if r4!=r3

       halt
```

1. Ahora realice simulaciones sin ningún tipo de optimizaciones. Observe el resultado de las mismas.

2. Aplicando las optimizaciones realizadas en la primera clase, vuelva a simular el código y revise el resultado.

3. Aplique la técnica de desenrollado de bucles al código anterior, guarde el resultado en un fichero diferente, llamado Codigo2.s. Este fichero es el que subir en la entrega del Bloque II.

    a. Ahora realice simulaciones sin ningún tipo de optimizaciones. Observe el resultado de las mismas.
    b. Aplicando las optimizaciones realizadas en la primera clase, vuelva a simular el código y revise el resultado.

4. Adjunte en un .doc el código con cada paso dado en el desenrollado que subirá en la entrega junto al .s del código resultante del último paso.

1.

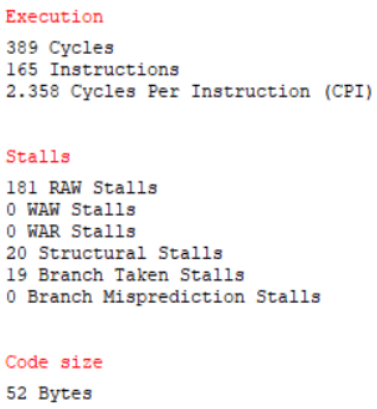

Ilustración 1.Ejecución de code.s sin mejoras

Nótese que para el resto se optimizaciones y mejoras no se tendrá en cuenta el **Branch Target Buffer** (Búfer de destino o **predictor** de saltos) por simplicidad.

2.

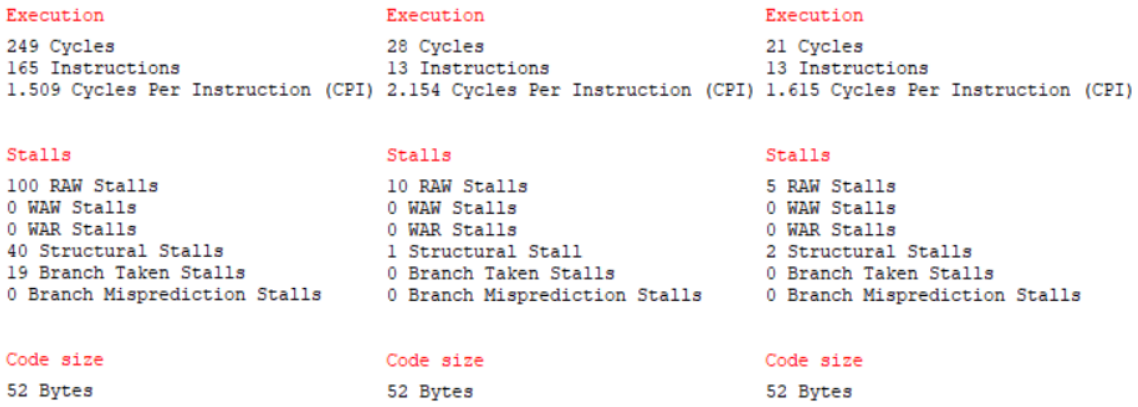

Ilustración 2. Comparativa de jecución de code.s con Forwarding, Delay Slot y Delay Slot + Forwarding resp.

Como podemos ver, activar el Delay Slot (DS) parece una gran mejora, más aún si lo combinamos con el Forwarding (anticipación).

Sin embargo, la realidad es que el DS en este punto no es viable; en el registro $f6 da un resultado completamente distinto dado que el bucle solo lo hace una vez, ya que el DS, cuando está activado, ejecuta la instrucción posterior a la del salto **bne**, que en este caso es ‘halt’, que equivale a terminar el programa.

La prueba de ello la encontramos en la siguente imagen, donde se ve el banco de registros tras acabar el programa. Registros de iteración $R4=1, $R1=8 y $R2=a8 (168 en decimal) solo han hecho su respectivo incremento una vez.

Así pues, el Delay Slot no será viable hasta que la instrucción inmediatamente posterior del salto no sea la última.

```
R0=  0000000000000000  F0=  0000001.00000000
R1=  0000000000000008  F1=  0000000.00000000
R2=  00000000000000a8  F2=  0000001.00000000
R3=  0000000000000014  F3=  0000000.00000000
R4=  0000000000000001  F4=  0000002.00000000
R5=  0000000000000000  F5=  0000000.00000000
R6=  0000000000000000  F6=  0000002.00000000
R7=  0000000000000000  F7=  0000000.00000000
R8=  0000000000000000  F8=  0000000.00000000
R9=  0000000000000000  F9=  0000000.00000000
R10= 0000000000000000  F10= 0000000.00000000
R11= 0000000000000000  F11= 0000000.00000000
R12= 0000000000000000  F12= 0000000.00000000
R13= 0000000000000000  F13= 0000000.00000000
R14= 0000000000000000  F14= 0000000.00000000
R15= 0000000000000000  F15= 0000000.00000000
R16= 0000000000000000  F16= 0000000.00000000
R17= 0000000000000000  F17= 0000000.00000000
R18= 0000000000000000  F18= 0000000.00000000
R19= 0000000000000000  F19= 0000000.00000000
R20= 0000000000000000  F20= 0000000.00000000
R21= 0000000000000000  F21= 0000000.00000000
R22= 0000000000000000  F22= 0000000.00000000
R23= 0000000000000000  F23= 0000000.00000000
```

Ilustración 3. Banco de registros tras ejecutarse code.s con DS

Por lo tanto, nos quedamos con el forwarding, que ha permitido bajar los Read After Write de 181 a 100, también bajan los ciclos y CPI. Su única pega es que se incrementarán los Structural Stalls.

3.

Paso 1

```
.data

X:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
Y:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09

.text

main:
    daddi r3,r0,20
    daddi r4,r0,0
    daddi r1,r0,X
    daddi r2,r0,Y

loop:
    l.d f0, 0(r1)
    l.d f2, 0(r2)
    add.d f4,f0,f2
    add.d f6,f6,f4

    l.d f0, 8(r1)   ;Clono las cuatro instrucciones anteriores
    l.d f2, 8(r2)   ;Y uso desplazamientos para acceder a la siguiente direccion de memoria
    add.d f4,f0,f2
    add.d f6,f6,f4

    l.d f0, 16(r1)
    l.d f2, 16(r2)
    add.d f4,f0,f2
    add.d f6,f6,f4

    l.d f0, 24(r1)
    l.d f2, 24(r2)
    add.d f4,f0,f2
    add.d f6,f6,f4

    daddi r1,r1,32  ;Como he clonado cuatro veces, 4*8 direcciones = 32
    daddi r2,r2,32
    daddi r4,r4,4   ;4 iteraciones de golpe (1*4=4)
    bne r4,r3,loop

    halt
```

Ilustración 4. Implementación del paso 1 para el desenroscado de bucles

El primer paso para poder desenrollar un bucle será replicar código, copiando las instrucciones que se ejecutarán en una misma iteración. En este caso, vamos a desenroscar el bucle en 'cuatro bloques', de manera que en una misma iteración vamos a utilizar 8 direcciones de memoria en lugar de 2 e incrementar el registro iterador de cuatro en cuatro. Con esto **logramos pasar por la instrucción de salto 4 veces menos.**

```
Execution
269 Cycles
105 Instructions
2.562 Cycles Per Instruction (CPI)

Stalls
151 RAW Stalls
0 WAW Stalls
0 WAR Stalls
5 Structural Stalls
4 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
100 Bytes
```

```
Execution
159 Cycles
105 Instructions
1.514 Cycles Per Instruction (CPI)

Stalls
85 RAW Stalls
0 WAW Stalls
0 WAR Stalls
25 Structural Stalls
4 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
100 Bytes
```

Ilustración 5. Comparativa de ejecución del primer paso de desenrollado de bucles. Sin y con forwarding

Como podemos ver, el **tamaño del código ha aumentado**, ya que hemos escrito más líneas en el código fuente. Esta es la única desventaja del desenroscado de bucles. No obstante, tenemos un **menor número de instrucciones, número de ciclos**, menores **RAW** Stalls y **Structural Stalls**. La activación del Forwarding disminuye drásticamente el número de RAW, aunque aumenten los Structural Stalls.

Paso 2

```
.data

X:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
Y:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09

.text

main:
    daddi r3,r0,20
    daddi r4,r0,0
    daddi r1,r0,X
    daddi r2,r0,Y

loop:
    l.d f0, 0(r1)
    l.d f2, 0(r2)
    add.d f4,f0,f2 #Usaremos mas registros para evitar esperas por usar constantemente
    add.d f6,f6,f4 #f0 y f2. El resultado del nuevo se seguira acumulando en f6

    l.d f8, 8(r1)
    l.d f10, 8(r2)
    add.d f12,f8,f10
    add.d f6,f6,f12

    l.d f14, 16(r1)
    l.d f16, 16(r2)
    add.d f18,f14,f16
    add.d f6,f6,f18

    l.d f20, 24(r1)
    l.d f22, 24(r2)
    add.d f24,f20,f22
    add.d f6,f6,f24

    daddi r1,r1,32
    daddi r2,r2,32
    daddi r4,r4,4

    bne r4,r3,loop

    halt
```

Ilustración 6. Implementación del paso 2 de desenroscado de bucles

El siguiente paso para nuestro desenrollado es **utilizar más registros** (en este caso registros **de tipo coma flotante**) en la medida de lo posible, para evitar esperas de lectura y escritura de los mismos registros (en este caso $f0 y $f2). De nuevo, a costa del aumento de recursos de memoria, aceleramos la velocidad de ejecución.

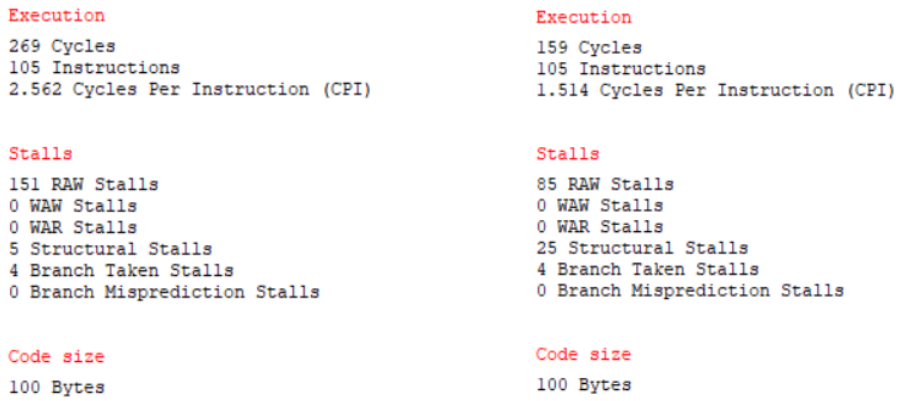

Ilustración 7. Comparativa de ejecución del segundo paso de desenrollado de bucles. Sin y con forwarding

Como podemos apreciar, **no hay ninguna mejora** en comparación con el paso anterior. Esto se debe a que, aunque usemos más registros, las instrucciones de suma no están colocadas de manera eficiente junto con las instrucciones de carga en registro, tal y como sucede en el Paso 1.

Paso 3

```
.data

X:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
Y:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09

.text

main:
    daddi r3,r0,20
    daddi r4,r0,0
    daddi r1,r0,X
    daddi r2,r0,Y

loop:
    l.d f0, 0(r1)
    l.d f2, 0(r2)
    l.d f8, 8(r1)
    l.d f10, 8(r2)
    l.d f14, 16(r1)
    l.d f16, 16(r2)
    l.d f20, 24(r1)
    l.d f22, 24(r2) #Las instrucciones de carga en registro las colocamos todas seguidas

    add.d f4,f0,f2
    add.d f6,f6,f4

    add.d f12,f8,f10
    add.d f6,f6,f12

    add.d f18,f14,f16
    add.d f6,f6,f18

    add.d f24,f20,f22
    add.d f6,f6,f24 #Lo mismo para las sumas y acumulaciones en f6

    daddi r1,r1,32
    daddi r2,r2,32
    daddi r4,r4,4

    bne r4,r3,loop

    halt
```

Ilustración 8. Implementación del paso 3 de desenroscado de bucles

Para este tercer paso, se han **agrupado todas las instrucciones de carga de registro y de suma**, con el fin de, junto con el mayor uso de registros, disminuir el número de Stalls.

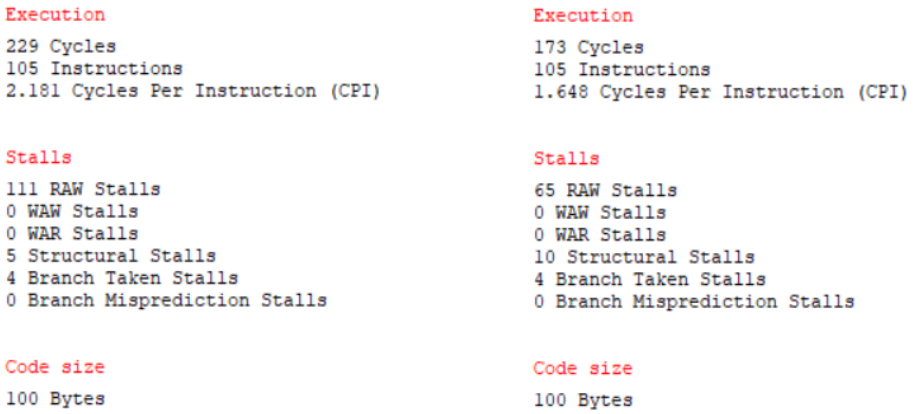

Ilustración 9. Comparativa de ejecución del tercer paso de desenrollado de bucles. Sin y con forwarding

Como podemos comprobar, sin forwarding, del segundo al tercer paso, tenemos una mejoría tanto en número de ciclos como en Stalls. No obstante, con forwarding hay una leve empeora en el número de ciclos.

Esto se debe a que del paso 2 al paso 3 hemos **quitado las instrucciones intermedias** que existían entre acumulaciones en el registro $f6. La consecuencia de esto es que el **procesador esperará más tiempo a que se complete la suma en $f6** y no puede ejecutar tantas instrucciones entre medias. Esto lo solucionaremos en el cuarto paso, con el adelantamiento de instrucciones y uso del desplazamiento negativo en los *l.d*.

Paso 4 – Codigo2.s

```
#Sumar valores de dos vectores. El resultado se guarda en el registro F6
#@autores: Pablo Blazquez Sanchez, Raul Jimenez de la Cruz
.data

X:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09
Y:  .double 1.00, 1.01, 1.02, 1.03, 1.04, 1.05, 1.06, 1.07, 1.08, 1.09
    .double 2.00, 2.01, 2.02, 2.03, 2.04, 2.05, 2.06, 2.07, 2.08, 2.09

.text

main:
    daddi r1,r0,X   #; r1 como registro base X
    daddi r2,r0,Y   #; r2 como registro base Y
    daddi r3,r0,20  #; r3 <-- Numero de elementos de cada vector (2x10)
    daddi r4,r0,0   #; r0 <-- contador // iterador
loop:
    l.d f0, 0(r1)   #; F0 <-- X(i)
    l.d f2, 0(r2)   #; F2 <-- Y(i)
    l.d f8, 8(r1)
    add.d f4,f0,f2  #; F4 <-- X(i) + Y(i) (suma elementos vector X e Y)

    l.d f10, 8(r2)
    l.d f14, 16(r1)
    add.d f12,f8,f10
    add.d f6,f6,f4  #; F6 <-- F6+f4 (acumulo en f6)

    l.d f16, 16(r2)

    daddi r1,r1,32  #Incremento direcciones de memoria r1. 8*4=32
    l.d f20, -8(r1) #Como queria acceder al elemento con desplazamiento 24, empleo -8 tras incremento
    add.d f18,f14,f16
    add.d f6,f6,f12

    l.d f22, 24(r2)
    daddi r2,r2,32
    add.d f24,f20,f22
    add.d f6,f6,f18

    daddi r4,r4,4 #aumento iterador. Incremento dirs de 4 en 4

    bne r4,r3,loop  #; salto si r4!=20 (que es r3)
    add.d f6,f6,f24
    halt
```

Ilustración 10. Implementación cuarto paso desenrollado de bucles. Código final de Codigo2.s

El cuarto y último paso será **reordenar instrucciones**. Dado a que se presentan algunos Stalls por la constante adición de números en el registro f6, vamos a adeltantar algunas instrucciones para evitar esperas innecesarias.

Así pues, adelantaremos cargas de registros, algunas sumas e incrementos de direcciones de memoria. Esto implica en alguna ocasión de usar un desplazamiento negativo para poder acceder al espacio de memoria correcto.

También se ha colocado una instrucción después del salto, por lo que **para ejecutar este código requerirá de tener activado el Delay Slot**, de lo contrario el resultado del programa será incorrecto.

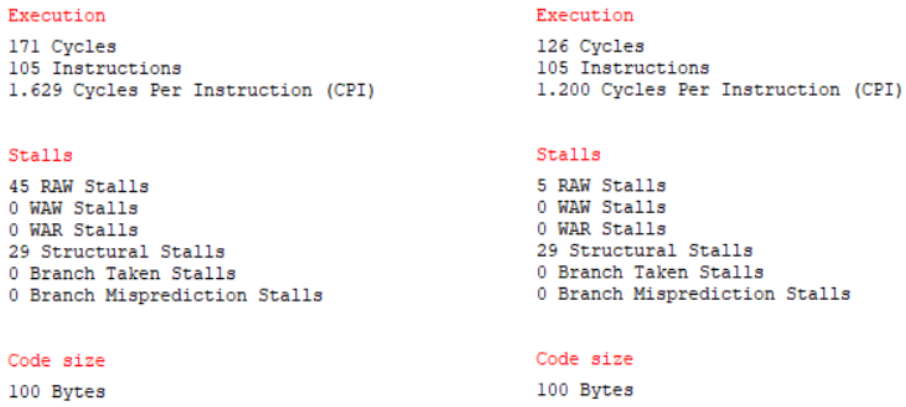

Ilustración 11. Comparativa de ejecución de Codigo2.s. Delay Slot activado. Sin y con forwarding

Como se puede apreciar, han disminuido considerablemente los ciclos y Stalls, obteniendo finalmente **126 ciclos**.[1]

Esto se debe a que hemos minimizado todo lo posible las esperas de escritura en $f6, sacrificando para ello las paradas de tipo *Structural*. Las únicas paradas que quedan de tipo RAW se deben a las insuficientes instrucciones que hay entre `add.d f6,f6,f18` y `add.d f6,f6,f24`.

[1] Esto significa que el nuevo programa se ejecuta, con todas las mejoras, 3 veces más rápido que el base.