Laboratorio de Arquitectura de computadores GITT Entrega 5

Fecha de entrega: Viernes 14/12/2012 Grupo: Viernes

Autores: Manuel Montoya Catalá

**Descripción de la práctica:**

# Para la realización de esta práctica usaremos el emulador DLXV3.1

Las consideraciones a tomar en cuenta son las siguientes:

- Está segmentado en 5 etapas equilibradas: IF, ID, EX, ME, WB



- No tiene en cuenta como ciclos las iniciales que tarda la primera instrucción en pasar por todas las etapas.

# Utiliza "**Adelantamiento de Operandos**" para reducir los parones debidos a dependencias del tipo RAW (Read After Write).

Esta técnica consiste en adelantar el resultado de las etapas EX y MEM de cada instrucción a las dos instrucciones siguientes:

Esta técnica logra:

- Caso de que la instrucción generadora del resultado sea de tipo **ALU**:

=> Elimina los parones ya que su resultado está listo en la etapa de ejecución EX,

pudiendo adelantar a tiempo el valor del mismo a la siguiente etapa EX

- Caso de que la instrucción generadora del resultado sea una **Operación de Memoria**

=> Elimina 1 de los 2 parones que se generarían sin esta técnica.

Esto es debido a que el nuevo dato a cargar en la memoria correspondiente sólo

está disponible a partir de la etapa MEM, por lo que si otra instrucción lo necesita

en la etapa EX, no se lo podrá adelantar a tiempo

- Caso de que la segunda instrucción sea una **Salto**

=> Los saltos necesitan la dirección de salto en su etapa ID.

a) Si la instrucción anterior es de tipo ALU y opera sobre el registro donde está la posición de memoria del salto, su resultado no estará listo hasta la etapa EX, por lo que habrá un parón aún con adelantamiento

b) Si la instrucción anterior es de tipo memoria y opera sobre el registro donde está la posición de memoria del salto, su resultado no estará listo hasta la etapa MEM, por lo que habrá dos parones aún con adelantamiento

**Ejercicio 1:**

- La ejecución de este código dura 15 ciclos, teniendo solamente 13 instrucciones.

- Se producen 2 ciclos de parada:

1º) En la instrucción "add r3,r2,r1", ya que su operando r1 es generado en la instrucción inmediata anterior mediante una instrucción de memoria

2º) En la instrucción "add r7,r6,r1", ya que su operando r6 es generado en la instrucción inmediata anterior mediante una instrucción de memoria

- No hay paradas en las últimas instrucciones "sw" ya que, aunque uno de sus operandos es generado en la instrucción imediata anterior, dicha intruccion es de tipo ALU por lo que su resultado estará listo en su etapa EX, y gracias al adelantamiento de operandos.

.data

a: .word 5

b: .word 17,18

c: .word 0,0,0,0,127

.text

ini: lw r1,a ;carga lo que haya en la direccion "a" en el registro r1

lw r1,a(r0) ;carga lo que haya en la direccion "a + r0" en registro r1

add r7,r0,r0 ;suma "r0 + r0" y lo guarda en r7

lw r1,a(r7) ; carga lo que haya en la direccion"a + r7" en el registro r1

add r3,r2,r1 ; -- PARÓN ; suma "r2 + r1" y lo guarda en r3

lw r4,b ;carga lo que haya en la direccion "b" en el registro r4

lw r5,a+4 ; carga lo que haya en la direccion "a + 4" en el registro r5

lw r6,a+4(r0); carga lo que haya en la direccion "a + 4 + r0" en el registro r6

add r7,r6,r1 ; -- PARÓN suma "r6 + r1" y lo guarda en r7

sw c,r7 ; guarda lo que haya en el registro r7 en la direccion "c"

addi r7,r7,4 ;suma "r7 + 4" y lo guarda en r7

sw c-4(r0),r7 ; guarda lo que haya en el registro r7 en la direccion "c-4"

trap #6 ;terminar la simulacion

**Ejercicio 2:**

# Para que el reordenamiento sea efectivo lo que tenemos que hacer es introducir otra instrucción del programa entre las 2 instrucciones consecutivas que generan el parón.

La única condición de dicha instrucción intermedia es que no altere el resultado del programa y por supuesto que no use como operando el mismo resultado.

- En nuestro caso intercambiamos:

1) add r3,r2,r1 con lw r4,b

2) lw r6,a+4(r0) con add r7,r6,r1

.data

a: .word 5

b: .word 17,18

c: .word 0,0,0,0,127

.text

ini: lw r1,a ;carga lo que haya en la direccion "a" en el registro r1

lw r1,a(r0) ;carga lo que haya en la direccion "a + r0" en registro r1

add r7,r0,r0 ;suma "r0 + r0" y lo guarda en r7

lw r1,a(r7) ; carga lo que haya en la direccion"a + r7" en el registro r1

lw r4,b ;carga lo que haya en la direccion "b" en el registro r4

add r3,r2,r1 ; -- NO PARÓN ; suma "r2 + r1" y lo guarda en r3

lw r6,a+4(r0); carga lo que haya en la direccion "a + 4 + r0" en el registro r6

lw r5,a+4 ; carga lo que haya en la direccion "a + 4" en el registro r5

add r7,r6,r1 ; -- NO PARÓN suma "r6 + r1" y lo guarda en r7

sw c,r7 ;guarda lo que haya en el registro r7 en la direccion "c"

addi r7,r7,4 ;suma "r7 + 4" y lo guarda en r7

sw c-4(r0),r7 ; guarda lo que haya en el registro r7 en la direccion "c-4"

trap #6 ;terminar la simulacion

**Ejercicio 3:**

# El código del programa es el siguiente:

.data

a: .word 1,2,3,-4,5,6,7,8,-9,10

b: .word 10,20,-30,-40,50,60,-70,-80,90,100

c: .word 40

s: .word 0,0,0,0,0,0,0,0,0,0

.text

ini:

lw r5,c(r0) ;cargamos el valor 10 en la direccion r5 Usado como contador

loop:

lw r1,a-4(r5) ;carga en r1 el sumando de a PARON !!!

lw r2,b-4(r5) ;carga r2 el sumando de b

add r3,r2,r1 ;Realiza la suma PARON !!!

sw s-4(r5),r3

subi r5,r5,4 ;restamos 1 al registro r0

bnez r5,loop ;si el contador no ha acabado seguimos sumando PARON !!!

nop ; instruccion de relleno para que no ejecute el trap PARON !!

trap #6 ;terminar la simulación

# Este código tarda en ejecutarse 93 ciclos y generan 31 parones:

- 1º Tipo de Parón: lw r1,a-4(r5)

Sólo se da la primera vez que el programa pasa por esa línea y es debido a que en la instrucción anterior, le estamos dando el valor a r5 mediante una operación de memoria

- 2º Tipo de Parón: add r3,r2,r1

Sucede en cada ciclo del bucle y es debido a que en la instrucción anterior, le estamos dando el valor a r2 mediante una operación de memoria.

- 3º Tipo de Parón: add r3,r2,r1

Sucede en cada ciclo del bucle y es debido a que el bucle bnez r5,loop necesita el valor del operando r5 en su etapa ID, pero este no estára disponible hasta la etapa EX de la instrucción anterior, produciéndose por tanto un hueco.

- 4º Tipo de Parón: nop

El nop intruce un parón en cada bucle

**Ejercicio 4:**

# El código optimizado es el siguiente:

.data

a: .word 1,2,3,-4,5,6,7,8,-9,10

b: .word 10,20,-30,-40,50,60,-70,-80,90,100

c: .word 40

s: .word 0,0,0,0,0,0,0,0,0,0

.text

ini:

lw r5,c(r0) ;cargamos el valor 10 en la direccion r5 Usado como contador

loop:

lw r1,a-4(r5) ;carga en r1 el sumando de a

lw r2,b-4(r5) ;carga r2 el sumando de a

subi r5,r5,4 ;restamos 1 al registro r0

add r3,r2,r1 ;Realiza la suma

bnez r5,loop ;si el contador no ha acabado seguimos sumando

sw s(r5),r3

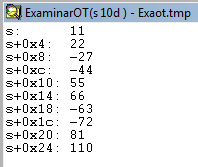
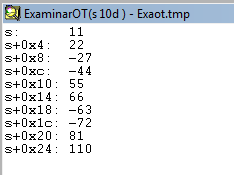
trap #6 ;terminar la simulación

- Para optimizar el programa, hemos aprovechado la ejecución de después del bucle para guardar los resultados y hemos intercambiado el subi con el add

- Este programa tarda 63 ciclos y posee un único parón en la segunda instrucción debido al uso de r5, pero no hemos visto forma de quitarlo sin perder generalidad en el diseño o añadir otras instrucciones por lo que ahí se queda.

- Para comprobar el resultado, vemos las posiciones en memoria de las

salidas de los 2 códigos:



- Vemos que los resultados coinciden y son los válidos

- La diferencia entre ellos es que uno tarda 93 ciclos y el otro 63.

- La ganancia es 93/63 = 1,4762

**Ejercicio 5:**

# El código sin optimizar es el siguiente:

.data 100

n: .word 5

suma: .space 4

.text 1000

ini: lw r7,n(r0)

sll r7,r7,#1 ;PARON

xor r1,r1,r1

loop: add r1,r1,r7

subi r7,r7,2

bnez r7,loop ;PARON

nop ;PARON

sw suma(r0),r1

trap #6

- Este código se ejecuta en 31 ciclos y tiene 11 parones

- Intentamos reordenar el código para eliminar los parones:

.data 100

n: .word 5

suma: .space 4

.text 1000

ini: lw r7,n(r0)

xor r1,r1,r1

sll r7,r7,#1

loop: add r1,r1,r7

bnez r7,loop

subi r7,r7,2

addi r7,r7,2

sw suma(r0),r1

trap #6

# En esta reordenación, hemos intercambiado xor y sll para eliminar ese primer parón y hemos aprovechado el hueco del bucle con la instrucción subi, pero esto a su vez crea el inconveniente de que el bucle se ejecutará una vez, con el retardo pertinente y tenemos que añadir una última instrucción addi r7,r7,2, para que el registro r7 quede como en el código original.

- Este código se ejecuta en 24 ciclos