Laboratorio de Arquitectura de computadores GITT Entrega 6

Fecha de entrega: Viernes 21/12/2012 (Grupo: Viernes)

Autor: Manuel Montoya Catalá

**Descripción de la práctica:**

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

- Al principio de la práctica usaremos un programa con operaciones en coma flotante cuya latencia es de 2 ciclos por lo que tenemos las etapas:

- ALU: IF ID EX EX ME WB

- MEMORIA: IF ID EX ME WB

- SALTO: IF ID

- Partimos de las siguientes premisas:

1- En el cauce no pueden haber 2 instrucciones en la etapa IF ó ID a la vez

2- Dos instrucciones ALU no pueden estar en la etapa EX a la vez.

3- Dos instrucciones de MEMORIA no pueden estar en la etapa EX a la vez.

4 – Una instrucción ALU y otra de MEMORIA pueden estar en la etapa EX a la vez

5 – La arquitectura segmentada utiliza "Adelantamiento de Operandos" pero solo

entre las etapas EX a EX ó MEM a EX

6 – Una instrucción de tipo ALU necesita sus operandos en la Etapa EX

7 – Una instrucción de tipo MEM necesita sus operandos en la Etapa MEM

8 – Una instrucción ALU podrá adelantar su resultado válido sólo a partir de la última

de sus etapas EX

9 – Una instrucción MEMORIA podrá adelantar su resultado válido sólo a partir de su

etapa MEM

**Ejercicio 1:**

# Código para el diagrama del cauce:

.data 0

c: .double 3

a: .double 1,2,3,4,5,6,7,8,9,10,11,12

b: .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

.text

ini: ;este array empieza en la posicion 8

addd f4,f2,f0 ; ALU - ALU f4

addd f6,f2,f4

addd f8,f10,f12

addd f8,f16,f0; ALU - Store f8

sd 16(r1),f8 ; (El desplazamiento 16 tiene que ser multiplo de 8)

ld f12,c(r1); LOAD - ALU f12

addd f2,f0,f12

ld f14,c(r1) ; LOAD - STORE f14

sd 80(r1),f14; (El desplazamiento 80 tiene que ser multiplo de 8)

trap #6

#Los diagramas de cauce son:

1) ALU – ALU: Hay 1 ciclo de parada

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Addd f4,f2,f0 | IF | ID | EX | EX | ME | WB |  |  |
| Addd f6,f2,f4 |  | IF | ID | ***RAW*** | EX | EX | ME | WB |

2) ALU – STORE: Hay 1 ciclo de parada

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Addd f4,f2,f0 | IF | ID | EX | EX | ME | WB |  |  |
| sd 16(r1),f8 |  | IF | ID | ***RAW*** | EX | ME | WB |  |

3) LOAD – ALU: Hay 1 ciclo de parada

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ld f12,c(r1) | IF | ID | EX | ME | WB |  |  |  |
| addd f2,f0,f12 |  | IF | ID | ***RAW*** | EX | EX | ME | WB |

4) LOAD – STORE: Hay 1 ciclo de parada

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| ld f14,c(r1) | IF | ID | EX | ME | WB |  |  |  |
| sd 80(r1),f14 |  | IF | ID | ***RAW*** | EX | ME | WB |  |

- Aquí hay in ciclo de parada ya que la arquitectura no tiene adelantamiento ME a ME, si la tuviera no lo habría

# Dado el siguiente código:

.data 0

c: .double 3

a: .double 1,2,3,4,5,6,7,8,9,10,11,12

b: .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

.text

ini: ;El array "a" empieza en la posicion 8

ld f2,c(r0) ; carga en f2 lo que haya en la direccion de c

addi r1,r0,b-8 ; aqui la "b" es el valor de su direccion en memoria, no el dato que almacena

loop:

ld f0,0(r1) ; carga en r0 el dato que haya en la direccion cuyo valor tiene el registro r1

addd f4,f0,f2 ;suma f0 y f2 y lo guarda en f4 !!!!!!!PARON de un ciclo (f0) !!!!!

sd 96(r1),f4 ;carga en la direccion (96+r1) lo que haya en f4 (el primero guarda en la posicion 12 de b)

;PARON de un ciclo (f4)

subi r1,r1,#8 ;resta 8 a r1

bnez r1,loop ; hasta que r1 no sea 0 no pasará !!!!!!!PARON de un ciclo r1 !!!!!!

nop

trap #6

a)

El código va sumando cada una de los 12 datos del array "a" con la constante de la dirección "c" y los guarda en las posiciones de b desde 12 a 1.

- El programa tarda 111 cilos:

- 24 parones escalares

- 12 parones por coma flotante

- 12 parones del nop

b)

Para recorrer el vector "a" valor por valor desde las posiciones más altas a las más bajas, y como son datos double de 64 bits se necesitan 8 bytes de dirección por cada dato el vector

- El elemento apuntado por la dirección 96(r1) es la dirección "96+r1".

En la primera ejecución r1 = 96 por lo que apunta al elemento número 12 de "b"

c)

Los parones están en:

- addd f4,f0,f2 : Parón de tipo LOAD – ALU causado por dependencia RAW de f0

- sd 96(r1),f4 : Parón de tipo ALU – STORE causado por dependencia RAW de f4

- bnez r1,loop : Parón de tipo ALU – SALTO causado por dependencia RAW de r1

Esta instrucción de tipo salto necesita el valor de R1 en su etapa ID

pero no le llegará hasta que termine la etapa EX de la instrucción subi

**Ejercicio 2:**

El codigo reordenado es:

.data 0

c: .double 3

a: .double 1,2,3,4,5,6,7,8,9,10,11,12

b: .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

.text

ini: ;El array "a" empieza en la posicion 8

ld f2,c(r0) ; carga en f2 lo que haya en la direccion de c

addi r1,r0,b-8 ; aqui la "b" es el valor de su direccion en memoria, no el dato que almacena

loop:

ld f0,0(r1) ; carga en r0 el dato que haya en la direccion cuyo valor tiene el registro r1

subi r1,r1,#8 ;resta 8 a r1

addd f4,f0,f2 ;suma f0 y f2 y lo guarda en f4

bnez r1,loop ; hasta que r1 no sea 0 no pasará

sd 104(r1),f4 ;carga en la direccion (104+r1) lo que haya en f4 (el primero guarda en la posicion 12 de b)

trap #6

- Hemos aprovechado el hueco del retardo con la instrucción sd a la que hemos tenido que cambiar el desplazamiento inicial de 96 a 104 ya que ahora el subi se ejecuta antes que ella.

- Hemos cambiado el subi por el addd para eliminar los 2 parones de un solo cambio.

- Tarda 63 ciclos y no tiene paradas

**Ejercicio 3:**

# El código para crear la tabla es el mismo que en el ejercicio 1 y los diagramas de cauce son lo siguientes:

1) ALU – ALU: Hay 7 ciclos de parada

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Addd f4,f2,f0 | IF | ID | EX | EX | EX | EX | EX | EX | EX | EX | ME | WB |  |  |  |  |  |  |
| Addd f6,f2,f4 |  | IF | ID | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | EX | EX | EX | EX | EX | EX | EX | EX |

2) ALU – STORE: Hay 7 ciclos de parada

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Addd f4,f2,f0 | IF | ID | EX | EX | EX | EX | EX | EX | EX | EX | ME | WB |  |  |  |  |  |  |
| sd 16(r1),f8 |  | IF | ID | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | EX | ME | WB |  |  |  |  |  |

3) LOAD – ALU: Hay 1 ciclo de parada

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| ld f12,c(r1) | IF | ID | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |  |  |  |
| addd f2,f0,f12 |  | IF | ID | ***ID*** | EX | EX | EX | EX | EX | EX | EX | EX | ME | WB |  |  |  |  |

4) LOAD – STORE: Hay 1 ciclo de parada

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| ld f14,c(r1) | IF | ID | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |  |  |  |
| sd 80(r1),f14 |  | IF | ID | ***RAW*** | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

- Aquí hay in ciclo de parada ya que la arquitectura no tiene adelantamiento ME a ME, si la tuviera no lo habría

# El hecho de que la latencia de EX sea de 8, hace que en esos 8 ciclos que está la instrucción en esa etapa, ninguna otra instrucción pueda estarlo por lo que 2 instrucciones ALU seguidas, aunque no tengan dependencia RAW seguirán teniendo un retraso de 7 ciclos:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Addd f4,f2,f0 | IF | ID | EX | EX | EX | EX | EX | EX | EX | EX | ME | WB |  |  |  |  |  |  |
| Addd f6,f2,f4 |  | IF | ID | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | ***ID*** | EX | EX | EX | EX | EX | EX | EX | EX |

|  |  |  |  |
| --- | --- | --- | --- |
| Productor | Consumidor | Ciclos Parada | Ejemplo |
| FP ALU | FP ALU | 7 | Add f4,f2,f0 / Addd f6,f2,f4 |
| FP ALU | Store | 7 | Addd f4,f2,f0 / s d 16(r1),f8 |
| Load | FP ALU | 1 | ld f12,c(r1) / addd f2,f0,f12 |
| Load | Store | 1 | ld f14,c(r1) / sd 80(r1),f14 |

**Ejercicio 4:**

.data 0

cte: .double 3

a: .double 1,2,3,4,5,6,7,8,9,10,11,12

b: .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

.text 0x1000

ini:

ld f2,cte(r0) ;

addi r1,r0,b-8; Apunta al último elemento del vector a

loop:

ld f0,0(r1);

addd f4,f0,f2; PARON LOAD - ALU (Se retrasa 1) X3

sd 12\*8(r1),f4; PARON ALU - STORE (Se retrasa 1) X3

ld f6,-8\*1(r1);

addd f8,f6,f2; PARON LOAD - ALU (Se retrasa 1) X3

sd 11\*8(r1),f8 PARON ALU - STORE (Se retrasa 1) X3

ld f10,-8\*2(r1);

addd f12,f10,f2; PARON LOAD - ALU (Se retrasa 1) X3

sd 10\*8(r1),f12 PARON ALU - STORE (Se retrasa 1) X3

ld f14,-8\*3(r1);

addd f16,f14,f2; PARON LOAD - ALU (Se retrasa 1) X3

sd 9\*8(r1),f16 PARON ALU - STORE (Se retrasa 1) X3

subi r1,r1,4\*8

bnez r1,loop; PARON ALU - SALTO (Se retrasa 1 ya que es suma normal) X3

nop

trap #6

a) El código se ejecuta en 75 ciclos

b) Tenemos 3 tipos de paradas:

- LOAD – ALU: Parón de un ciclo. Hay 12 de estas (ALU de 64 bits)

- ALU – STORE: Parón de un cilo. Hay 12 de estas (ALU de 64 bits)

- ALU – SALTO : Parón de un cilo. Hay 3 de estas (ALU de 32 bits)

**Ejercicio 5:**

# El código a reordenar es:

.data 0

cte: .double 3

a: .double 1,2,3,4,5,6,7,8,9,10,11,12

b: .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

.text 0x1000

ini:

ld f2,cte(r0) ;

addi r1,r0,b-8

loop:

ld f0,0(r1); Apunta al último elemento del vector a

ld f6,-8\*1(r1);

ld f10,-8\*2(r1);

ld f14,-8\*3(r1);

addd f4,f0,f2;

addd f8,f6,f2; PARON ALU - ALU (Se retrasa 1) X3

addd f12,f10,f2; PARON ALU - ALU (Se retrasa 1) X3

addd f16,f14,f2; PARON ALU - ALU (Se retrasa 1) X3

sd 12\*8(r1),f4;

sd 11\*8(r1),f8

sd 10\*8(r1),f12

subi r1,r1,4\*8

bnez r1,loop; PARON ALU - SALTO (Se retrasa 1 ya que es suma normal) X3

sd 13\*8(r1),f16

trap #6

-Para quitar los parones tendremos que intercalar entre los addd, operaciones de memoria, teniendo cuidado en que no se produzcan dependencias RAW.

Así pues intercalamos 2 ld y un sd (No podemos intarcalar los 3 ld porque habría un RAW con el primer addd).

- Para quitar el paron ALU-SALTO intercambiamos las posiciones de subi y el último sd

Esto desajustará el valor de r1 con relación al bucle por lo que lo que tenemos que hacer es restarle 4\*8 al registro r1 desde el principio y sumar 4\*8 de desplazamiento a todas las operaciones de memoria

- El código resultante que se ejecuta en 45 está en la siguiente hoja

- La ganancia respecto al código sin desenrollar es:

.data 0

cte: .double 3

a: .double 1,2,3,4,5,6,7,8,9,10,11,12

b: .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

.text 0x1000

ini:

ld f2,cte(r0) ;

addi r1,r0,b-8-4\*8; Restamos inicialmente los 4\*8 del subi al r1, tendremos qu sumarle eso a las operaciones de memoria

loop:

ld f0,4\*8(r1); Apunta al último elemento del vector a

ld f6,-8\*1+4\*8(r1);

addd f4,f0,f2;

ld f10,-8\*2+4\*8(r1);

addd f8,f6,f2;

ld f14,-8\*3+4\*8(r1);

addd f12,f10,f2;

sd 12\*8+4\*8(r1),f4;

addd f16,f14,f2;

sd 11\*8+4\*8(r1),f8

sd 10\*8+4\*8(r1),f12

sd 9\*8+4\*8(r1),f16

bnez r1,loop;

subi r1,r1,4\*8

trap #6