Ejercicio 4.1 y 4.2

Miguel Angel Dorado Maldonado

Ejercicio 1

IF	ID	EX	MEM	WB
200ps	150ps	120ps	190ps	140ps

a. Da el tiempo del ciclo de reloj para un procesador segmentado y para un procesador monociclo..
b. Da la latencia de una instrucción lw en un procesador segmentado y un procesador monociclo.
c. Supongamos que se puede dividir una etapa del camino de datos segmentado en dos nuevas etapas, cada una con una latencia mitad de la etapa original. ¿Qué etapa se debería dividir y cuál sería el nuevo ciclo de reloj del procesador?

a) En un procesador monociclo la instrucción completa corre en un ciclo de reloj. En este caso el ciclo de reloj dura lo mismo que la instrucción más larga. Por lo tanto un ciclo dura 800ps.
(lw -> 1ciclo 800ps)

En el caso del procesador segmentado, los ciclos son más cortos y solo representan una etapa de la instrucción. Cada ciclo dura lo mismo que la etapa más lenta. 200ps.
(lw -> 5ciclos 200ps/c)

b) El tiempo para completar una intrucción en un procesador monociclo es la suma de las 5 etapas de la instrucción. 200 + 150 + 120 + 190 + 140 = 800.
El tiempo para completar una intrucción en un procesador segmentado sigue siendo el mismo. Estos procesadores permiten aumentar el número de instrucciones por unidad de tiempo llevando varias a la vez, pero el tiempo dedicado a cada instrucción sigue siendo el mismo.

c) Lo más óptimo será dividir la etapa más lenta. En este caso if -> if1 = 100ps, if2 = 100ps. De esta forma la etapa más lenta ahora será MEM con 190ps.
Tendremos 6 ciclos de 190ps cada uno

Ejercicio 2

(H.P. 4.16) Para cada una de las siguientes instrucciones:

```
a)      lw $1, 40($6)
b)      add $5, $5, $5
```

a. Indica que valores se almacena en cada uno de los registros situados entre dos etapas del pipeline: IF/ID, ID/EX, EX/MEM and MEM/WB.
b. Qué registros necesitan leerse y cuales se leen realmente?
c. Indica que hace la instrucción en las etapas EX y MEM.

a)

`lw $1, 40($6)`

IF/ID -> PC+4, codigo Inst.
ID/EX -> PC+4, PC+4-shift, RA, RB, RC, desplazamiento.
EX/MEM -> PC+4+shift, aluRes, RB[], RC.
MEM/WB -> MemRes, aluRes.

`add $5, $5, $5`

IF/ID -> PC+4, codigo Inst.
ID/EX -> PC+4, PC+4-shift, RA, RB, RC, desplazamiento.
EX/MEM -> PC+4+shift, aluRes, RB[], RC.
MEM/WB -> MemRes, aluRes.

b)

Para las dos instrucciones el IF/ID va a ser el mismo ya que se necesita leer de memoria de instrucciones y aumentar el PC.

Los registros necesarios a utilizar son:

`lw $1, 40($6)`

ID/EX -> RA, RC, desplazamiento.
EX/MEM -> aluRes, RC
MEM/WB -> MemRes, RC

`add $5, $5, $5`

ID/EX -> RA, RB, RC.
EX/MEM -> aluRes, RC
MEM/WB -> aluRes, RC

c)

`lw $1, 40($6)`

EX
Suma el valor de RA y desplazamiento que será la dirección accedida en MEM.
Tambien pasa el valor de RC donde se escribirá posteriormente.

MEM

Accede a la direccion pasada en EX y la guarda en WB al igual que RC.

```
add $5, $5, $5
```

EX

Suma los valores de RA y RB y pasa el valor del resultado al igual que RC donde se escribirá más tarde.

MEM

Recoge y envía aluRes a la siguiente etapa al igual que RC.