Arquitectura de Computadores. Taller. Curso 24/25.

1. En un procesador se pretende ejecutar el siguiente segmento de código:

```
I0 : lw $r4, 0( $r1 )
I1 : lw $r5, 0( $r2 )
I2 : add $r4, $r4, $r5
I3 : sw $r4, 0( $r3 )
I4 : addi $r1, $r1, 4
I5 : addi $r2, $r2, 4
I6 : addi $r3, $r3, 4
I7 : bne $r3, $r0, i0
```

Asumimos que el procesador tiene una arquitectura segmentada de 5 etapas como la del MIPS estándar sin envío adelantado (sin *forwarding*). Además en un mismo ciclo se puede escribir en un registro y leer de él. Todas las operaciones se ejecutan en un ciclo por etapa, excepto:

- Las operaciones de carga y almacenamiento, que requieren un total de dos ciclos para la etapa de memoria (un ciclo adicional).
- Las instrucciones de salto, que requieren un ciclo adicional en la etapa de ejecución. Considerar además que estas instrucciones no cuentan con ningún tipo de predicción de saltos.

Contestar razonadamente a las siguientes cuestiones:

a) Determinar los riesgos de datos que presenta el código e indicar cuales tienen impacto en su ejecución.

b) Mostrar un diagrama de tiempos con las fases de ejecución de cada instrucción para una iteración del código. Indicar en un diagrama de tiempos qué paradas hay y explicar porqué se producen. Determinar el número de ciclos requeridos para ejecutar una iteración del código que no sea la primera. Considerar dos casos posibles: que la etapa ID tenga hardware para realizar la comparación de registros para bne y que no.

c) Consideramos que el bucle se ejecuta 1000 veces. Proponer un desenrollamiento de lazos del bucle con un factor de cuatro iteraciones mostrando la ejecución en un diagrama de tiempos e indicando si hay paradas y a qué se deben. Indicar el número de ciclos por cada iteración y totales. Realizar el desenrollamientos considerando dos casos posibles: que la etapa ID tenga hardware para realizar la comparación de registros para bne y que no.

d) Indicar la aceleración o speedup obtenido mediante el desenrollamiento.

1.-

Tipo RAW (Read-After-Reading) Afectan al código

I0 → I2, sobre $r4. (lw escribe $r4 y add lee el registro; add $r4, $r4, $r5)

I1 → I2, sobre $r5. (Similar al caso anterior)

I2 → I3, sobre $r4. (la instrucción addi escribe el registro y sw lo lee para almacenar su contenido).

I6 → I7, sobre $r3. (addi escribe y bne lee para comprobar)

Tipo WAR (Write-After-Read) ∅

Tipo WAW (Write-After-Write)

I0 → I2, sobre $r4. (lw escribe el registro y, luego, addi también)

2.-

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
I0	IF	ID	EX	M1	M2	WB																	
I1		IF	ID	EX	- *1	M1	M2	WB															
I2			IF	- *2	- *2	- *2	- *2	ID	EX	M	WB												
I3								IF	- *3	*3	ID	EX	M1	M2	WB								
I4											IF	ID	EX	- *4	M	WB							
I5												IF	ID	- *5	EX	M	WB						
I6													IF	- *6	ID	EX	M	WB					
I7															IF	- *7	*7	ID	EX1	EX2	M	WB	Ciclos por Iteración
I0 Hardware																- *8	*8	- *8	IF	ID	EX	M1	18
I0 Normal																-	-	-	-	-	IF	ID	20

*9

*1 I1 no puede acceder a memoria si no terminó de hacerlo I0

*2 I2 no puede comenzar a decodificarse hasta que I1 no hace WB (I0 tmb lo debe hacer, pero I1 es posterior a I0).

*3 I3 debe esperar el WB de I2

*4 I4 no puede acceder a memoria si I3 no terminó de hacerlo

*5 I5 no puede ejecutarse hasta que I4 no libera la etapa de ejecución

*6 Como I5 no puede avanzar a EX, permanece en ID, lo que ocasiona que I6 no pueda decodificarse

*7 I7 debe esperar el WB de I6

*8 I7 debe ejecutarse para saber su próxima instrucción. Con hardware podemos saber qué instrucción será la próxima cuando finaliza la decodificación. Hasta ese momento no se hace nada.

*9 A diferencia del caso anterior, sin hardware no se sabe la próxima instrucción hasta que se ejecute la comparación en la ALU, es decir, termine de ejecutarse.

Como se ha dicho, sin el hardware en ID, se debe ejecutar por completo en la ALU la comparación de registros. Esto nos lleva a 20 ciclos en una iteración. Con el hardware logramos 18 ciclos. (Los ciclos de una iteración van desde el 1er IF, hasta el ciclo anterior al 1er IF de la siguiente iteración)

3:

```
I0 : lw  $r4, 0($r1)
I1 : lw  $r5, 0($r2)
I2 : lw  $r6, 4($r1)
I3 : lw  $r7, 4($r2)
I4 : lw  $r8, 8($r1)
I5 : lw  $r9, 8($r2)
I6 : lw  $r10, 12($r1)
I7 : lw  $r11, 12($r2)
I8 : add $r4, $r4, $r5
I9 : add $r6, $r6, $r7
I10: add $r8, $r8, $r9
I11: add $r10, $r10, $r11
I12: sw  $r4, 0($r3)
I13: sw  $r6, 4($r3)
I14: sw  $r8, 8($r3)
I15: sw  $r10, 12($r3)
I16: addi $r1, $r1, 16
I17: addi $r2, $r2, 16
I18: addi $r3, $r3, 16
I19: bne  $r3, $r0, $r0
```

Almacenas, en parejas de 2 registros para almacenar $r1 y $r2, las 4 iteraciones (factor 4 de desarrollamiento)

4 cts x offset de 4 = 16 bytes en 1 it

Se realizan correspondientes en la iteración

Se almacenan los valores teniendo en cuenta el offset

Se suma al valor almacenado el nº de bytes procesado en 1 it;

* Nota

Se le suma al valor del registro por la forma en que funciona lw;

lw $r4, 0($r1) -> $r4 = MEM[0+$r1]

Si le sumamos 4 al valor almacenado;

$r4 = MEM[0+($r1+4)]

* Tabla + explicaciones sig página

H:

1 iteración	Desarrollado	Simple
Hardware	8'25 ciclos	18 ciclos
No Hardware *	8'75 ciclos	20 ciclos

1000 cts

825 1800

875 2000

$$Speedup = \frac{TejecBase}{Tejec\ Y}$$

$$speedup_{Coutt.} = \frac{1800}{825}; \quad \boxed{speedup_{Coutt} = 2'18}$$

$$speedup_{Siutt} = \frac{2000}{875}; \quad \boxed{speedup_{Siutt} = 2'28}$$

* Nota

Se debe tener en cuenta que desarrollamos el bucle de forma que en 1 iteración hacemos el trabajo de 4; factor 4 de desarrollamiento

$\frac{33}{4} = 8'25$, $\frac{35}{4} = 8'75$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	
I0	IF	ID	EX	M1	M2	WB																																
I1		IF	ID	EX	-	M1	M2	WB																															
I2			IF	ID	-	EX		M1	M2	WB																												
I3				IF	-	ID		EX		M1	M2	WB																										
I4						IF	-	ID	-	EX		M1	M2	WB																								
I5								IF	-	ID		EX	-	M1	M2	WB																						
I6										IF		ID	-	EX	-	M1	M2	WB																			
I7												IF	-	ID	-	EX	-	M1	M2	WB																	
I8														IF	-	ID	-	EX	-	M	WB																
I9																IF	-	ID	-	EX	M	WB															
I10																		IF	-	ID	EX	M	WB														
I11																				IF	ID	EX	M	WB													
I12																					IF	ID	EX	M1	M2	WB											
I13																						IF	ID	EX	-	M1	M2	WB									
I14																							IF	ID	-	EX	-	M1	M2	WB							
I15																								IF	-	ID	-	EX	-	M1	M2	WB					
I16																										IF	-	ID	-	EX	-	M	WB				
I17																												IF	-	ID	-	EX	M	WB			
I18																														IF	-	ID	EX	M	WB		
I19																																IF	ID	EX1	EX2	M	WB	Total ciclos
Con hardware																																		IF	ID	EX	M	33
Sin hardware																																				IF	ID	35

* Se produce un patrón repetitivo en las paradas. Supongamos que, de arriba hacia abajo, las 3 paradas son i, i+1 e i+2. Con esto podemos decir que la parada i se produce porque la instrucción anterior aún no terminó con la memoria, por lo que, la instrucción con parada i, no puede entrar y se mantiene en la fase EX. Es por esto que se produce i+1, no puede ejecutarse y debe mantenerse en ID. De igual forma, i+2 es causada debido a que la instrucción anterior no liberó la decodificación, teniéndose que mantener en IF.

Como pudimos ver en el apartado anterior, gracias al hardware de la fase ID en el bus, ahorramos 2 ciclos.