

**PROGRAMADOR UNIVERSITARIO  
LICENCIATURA EN INFORMÁTICA  
INGENIERÍA EN INFORMÁTICA**

Facultad de Ciencias Exactas y Tecnología  
Universidad Nacional de Tucumán



**ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II  
TRABAJO PRÁCTICO N° 6  
PRADO, MATIAS SANTIAGO**

**1) Rendimiento de Ciclo-Único versus Pipeline**

a) Complete la tabla y compare el tiempo promedio entre las siguientes instrucciones en una implementación de ciclo-único (todas las instrucciones toman un ciclo de reloj), a una implementación con pipeline; sabiendo que los tiempos de operación de las principales funciones son:

300 ps para acceso a memoria

250 ps para operación con la ALU

150 ps para lectura de registro

200 ps para escritura de registro.

lw \$t1, 50(\$t0)

add \$s0, \$t2, \$t3

sw \$t3; 0(\$t0)

Tipo instrucción	Buscar instrucción (fetch)	Lectura registro	Operación ALU	Acceso a datos	Escritura de registro	Tiempo total
Cargar palabra	300	150	250	300	200	1200
Guardar palabra	300	150	250	300		1000
suma	300	150	250		200	900

b)

**PONER FOTO**

Por lo tanto el tiempo total de ciclo en una implementación de ciclo-único es: 1200 ps, puesto que este es el mayor tiempo entre las instrucciones

Por otro lado el tiempo total de ciclo en una implementación pipelining es de 300 ps, puesto que esta es la operación que mas tiempo lleva y mientras esta se este ejecutando ninguna otra podrá ejecutarse y además abarcara el tiempo de ejecución de todas las demás.

Las instrucciones en el caso de pipelining la primera arrancara en 0 y le tomara 1500ps porque usa las 5 operaciones, la segunda arranca en 300 y demora 1200ps porque usa 4 operaciones y por ultimo la 3ra arranca en 600 y le toma 1200ps, por lo tanto el tiempo total empleado en pipelining es de 1800ps

$$Promedio_{ciclo\ unico} = \frac{(1200 + 1200 + 1200)}{3} = 1200 [ps]$$

$$Promedio_{pipeling} = \frac{1800}{3} = 600 [ps]$$

c)

No, no hay ninguna posibilidad de pipelining stall. Suponiendo que en nuestro caso el mas probable es un **riesgo de datos** analizamos las instrucciones:

Analicemos las instrucciones dadas:

lw \$t1, 50(\$t0) (Escribe en el registro \$t1)

add \$s0, \$t2, \$t3 (Lee de \$t2 y \$t3. Escribe en \$s0)

sw \$t3, 0(\$t0) (Lee de \$t3 y \$t0)



**ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II**  
**TRABAJO PRÁCTICO N° 6**  
PRADO, MATIAS SANTIAGO

La instrucción add (la 2<sup>a</sup>) necesita los registros \$t2 y \$t3. La instrucción lw (la 1<sup>a</sup>) no modifica ninguno de esos dos; escribe en \$t1.

La instrucción sw (la 3<sup>a</sup>) necesita los registros \$t3 y \$t0. Ni la instrucción lw (1<sup>a</sup>) ni la add (2<sup>a</sup>) modifican esos registros (\$t0 o \$t3).

Dado que no hay dependencias de datos entre estas tres instrucciones consecutivas, el pipeline puede ejecutarse sin necesidad de *stalls* (burbujas)

2) Procesos con etapas, responda a las siguientes preguntas:

a) Si un proceso tiene 5 etapas de 1 hora de duración cada una y permite utilizar el concepto de pipelining. ¿Cuántas horas tomará realizar el proceso 4 veces?

Entendiendo bien que cada etapa es de 1hs y ni bien termine la 1<sup>era</sup> etapa puede comenzar la primera etapa nuevamente de forma paralela. Por lo tanto si solo se realizará el proceso 1 vez tendríamos 5 hs en total, pero como lo queremos hacer 4 veces, la ultima etapa en la 2<sup>da</sup> vez terminará en la 6<sup>ta</sup> hora y la 3<sup>ra</sup> vez la ultima etapa terminará en la 7<sup>ma</sup> hora y así sucesivamente tendremos que el proceso demorara 8hs en ejecutarse 4 veces. Ejemplo en el grafico siguiente

PONER FOTO

b) Si un proceso tiene 5 etapas de 1 hora de duración cada una y permite utilizar el concepto de pipelining. ¿Cuántas horas tomará realizar el proceso 5 veces?

Tomará exactamente 1 hs mas que el caso anterior

c) ¿Cuánto es la mejora de un proceso de 4 etapas de 1 hora cada una si se lo aplica en un pipelining ideal para hacer 7 veces el mismo proceso?

$$T_{sinpipe} = 4 * 1[hs] * 7 = 28[hs]$$

$$T_{pipelining} = (4 + 7 - 1) * 1[hs] = 10[hs]$$

$$mejora_{pipe} = \frac{T_{sinpipe}}{T_{pipelining}} = \frac{28}{10} = 2,8$$

menor que la mejora ideal que sería 4 (correspondiente a la cantidad de etapas) debido al bajo numero de procesos (instrucciones)



**ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II**  
**TRABAJO PRÁCTICO N° 6**  
PRADO, MATIAS SANTIAGO

d) ¿Pipelining permite reducir el tiempo de ejecución de una única instrucción o reduce el tiempo total de un conjunto de instrucciones?

Pipelining lo que permite es reducir el tiempo total de un conjunto de instrucciones, a una instrucción en particular, incluso, puede empeorarla, lo que mejora es las instrucciones por segundo ejecutadas (throughput) no el tiempo total que lleva una intstruccion (latency)

e) ¿Cuáles podrían ser los posibles inconvenientes de tener un pipeline con demasiadas etapas?

- Mayor penalización por riesgos de control: Una instrucción de salto (branch) que se predice incorrectamente vaciará un pipeline más largo, lo que significa que se descartan más instrucciones "en ejecucion" y se pierden más ciclos de reloj.
- Mayor penalización por riesgos de datos: En riesgos como el "load-use", es posible que el dato tarde más etapas en estar disponible desde la memoria, forzando más ciclos de stall (burbujas).
- Aumento de la latencia: Cada etapa introduce un pequeño retardo. Muchas etapas suman muchos retardos, lo que puede aumentar ligeramente la latencia total de una instrucción individual.

f) ¿Qué sucede cuando el forwarding no es suficiente para resolver un riesgo de datos?

Cuando esto sucede la ejecución debe deternse y este es un caso de stall o burbuja. Cuando se detiene se pierde un ciclo de reloj, el procesador no hace nada hasta que el dato esta disponible.

g) ¿Los riesgos de control ocurren cuando el pipeline debe pararse porque un paso debe esperar para que otro se complete?

No, esa afirmación es falsa.

La pregunta describe un **Riesgo de Dato (Data Hazard)**.

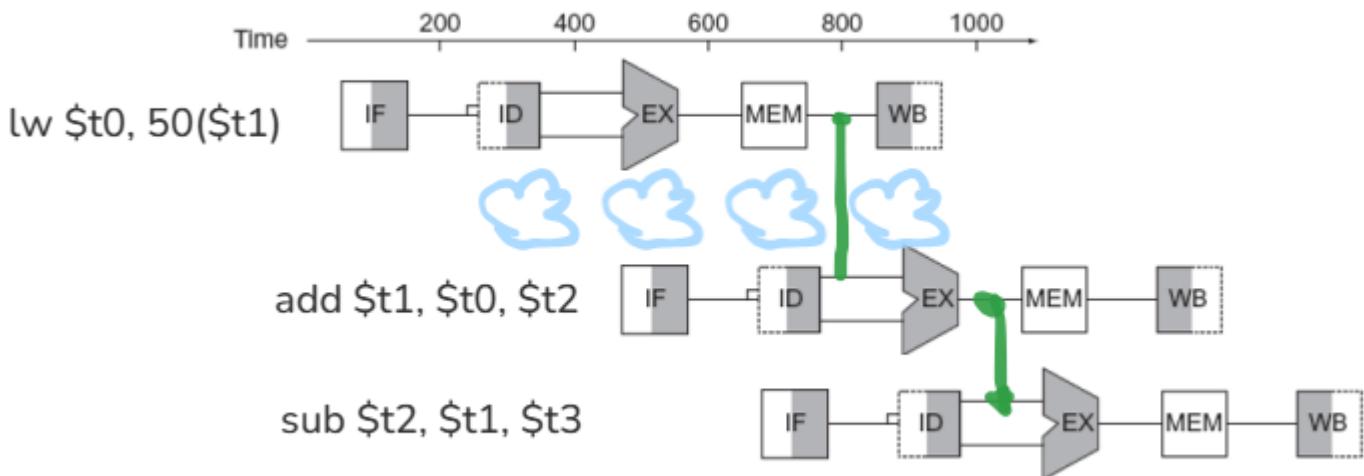
Un **Riesgo de Control (Control Hazard)** ocurre específicamente por la necesidad de tomar una decisión (como en una instrucción de *branch* o salto) mientras otras instrucciones ya están en ejecución.



**ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II**  
**TRABAJO PRÁCTICO N° 6**  
 PRADO, MATIAS SANTIAGO

3) Represente en qué etapas del pipeline se puede aplicar forwarding para el siguiente código de instrucciones:

lw \$t0, 50(\$t1)  
 add \$t1, \$t0, \$t2  
 sub \$t2, \$t1, \$t3



4) Reordenar las instrucciones o agregar otras para el siguiente segmento de código para evitar paradas en el pipeline.

lw \$t1, 0(\$t0)  
 sub \$t2,\$t1,\$t3  
 lw \$t2, 4(\$t0)  
 add \$t3, \$t1,\$t2  
 sw \$t3, 12(\$t0)  
 sw \$t2, 24(\$t0)  
 lw \$t2, 16(\$t0)  
 lw \$t4, 8(\$t0)  
 add \$t5, \$t1,\$t4  
 sw \$t5, 16(\$t0)



**ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II**  
**TRABAJO PRÁCTICO N° 6**  
 PRADO, MATIAS SANTIAGO

1. lw \$t1, 0(\$t0)
2. sub \$t2,\$t1,\$t3 ; <-- STALL (necesita \$t1)
3. lw \$t2, 4(\$t0)
4. add \$t3, \$t1,\$t2 ; <-- STALL (necesita \$t1 de 1 y \$t2 de 3)
5. sw \$t3, 12(\$t0)
6. sw \$t2, 24(\$t0) ; <-- STALL (necesita \$t2 de 3)
7. lw \$t2, 16(\$t0)
8. lw \$t4, 8(\$t0)
9. add \$t5, \$t1,\$t4 ; <-- STALL (necesita \$t1 de 1 y \$t4 de 8)
10. sw \$t5, 16(\$t0) ; (Esta es la 10)

evitando el stall

1. lw \$t1, 0(\$t0) ; Inicia la carga de \$t1
8. lw \$t4, 8(\$t0) ; Inicia carga de \$t4 (LLENA EL STALL de \$t1)
2. sub \$t2, \$t1, \$t3 ; \$t1 está listo, NO HAY STALL
3. lw \$t2, 4(\$t0) ; Sobrescribe \$t2
9. add \$t5, \$t1, \$t4 ; \$t1 está listo. \$t4 está listo (LLENA EL STALL de \$t2)
10. sw \$t5, 16(\$t0) ; El resultado de 'add' (9) es 'forwarded'. NO HAY STALL
4. add \$t3, \$t1, \$t2 ; \$t1 y \$t2 (de 3) están listos. NO HAY STALL
6. sw \$t2, 24(\$t0) ; \$t2 (de 3) está listo. NO HAY STALL
5. sw \$t3, 12(\$t0) ; El resultado de 'add' (4) es 'forwarded'. NO HAY STALL
7. lw \$t2, 16(\$t0) ; Se ejecuta al final, como en el original

5) Determine si existe algún tipo de riesgo

- a) add \$s0, \$t0, \$t1  
 sub \$s0, \$t0, \$t2

Tipo: Riesgo de Dato (Data Hazard).

Justificación: Este es un riesgo de tipo WAW (Write After Write). Ambas instrucciones intentan escribir un resultado en el mismo registro, \$s0. Aunque en un pipeline simple de 5 etapas esto se resuelve porque las escrituras (WB) ocurren en orden, sigue siendo una dependencia de datos que debe ser gestionada.

- b) add \$t3, \$t0, \$t1  
 add \$s0, \$t3, \$t2

Tipo: Riesgo de Dato (Data Hazard).

Justificación: Este es el clásico riesgo RAW (Read After Write). La segunda instrucción (add \$s0,...) necesita leer el valor del registro \$t3 , el cual es el resultado que la primera instrucción (add \$t3,...) está calculando. Esto crea una dependencia directa que, si no se maneja (con forwarding o un stall), causará un error.

**PROGRAMADOR UNIVERSITARIO  
LICENCIATURA EN INFORMÁTICA  
INGENIERÍA EN INFORMÁTICA**

Facultad de Ciencias Exactas y Tecnología  
Universidad Nacional de Tucumán

**ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II  
TRABAJO PRÁCTICO N° 6  
PRADO, MATIAS SANTIAGO**

