Práctico 5: Segmentación

Objetivo: Comprender los conceptos de segmentación (pipelining), dependencias de datos, riesgos, y técnicas de optimización como adelantamiento (forwarding) y predicción de saltos en procesadores RISC.

Nota:

- Todos los tiempos se expresan en nanosegundos (ns).
- Las etapas del pipeline son: IF (Instruction Fetch), ID (Instruction Decode), EX (Execute), MEM (Memory Access), WB (Write Back).

Parte 1: Segmentación Básica

1. Comparar el rendimiento de un procesador uniciclo (sin segmentación) con uno segmentado (pipeline de 5 etapas) utilizando los tiempos por etapa de la siguiente tabla: a) Diagramas de ejecución:

Clase de instrucción	Búsqueda de instrucción (IF)	Lectura de registros (ID)	Operación de la ALU (EX)	Acceso al dato (MEM)	Escritura en registros (WB)	Tiempo total
Cargar palabra (lw)	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
Almacenar palabra (sw)	2 ns	1 ns	2 ns	2 ns		7 ns
Formato R (add,and,slt..)	2 ns	1 ns	2 ns		1 ns	6 ns
Salto condicional (beq)	2 ns	1 ns	2 ns			5 ns

a) Grafique la ejecución de las instrucciones lw, sw, add en:
   1. Un procesador uniciclo (sin segmentación).
   2. Un procesador segmentado de 5 etapas.

Sugerencia: Use un diagrama de tiempos donde cada columna represente un ciclo de 2 ns (el tiempo máximo por etapa) y cada fila una instrucción. Indique claramente el inicio y fin de cada instrucción o etapa.

b) Cálculo del speed-up:
   - Calcule el tiempo total de ejecución de las 3 instrucciones en ambos procesadores.
   - Determine la mejora (speed-up) de la segmentación con la fórmula:

   $Speed - up = \frac{Tiempo\ Uniciclo}{Tiempo\ Segmentado}$

c) Escalabilidad:
   - Si se ejecutan 1003 instrucciones en lugar de 3, ¿cómo cambian los tiempos totales en ambos procesadores?
   - Generalice para $n \to \infty$ y explique cómo la segmentación mejora el rendimiento a largo plazo.

d) Impacto de la etapa EX:
   - Si el tiempo de la etapa EX aumenta de 2 ns a 4 ns:
     1. ¿Cómo se modifica el ciclo de reloj del pipeline?
     2. Recalcule el speed-up para las 3 instrucciones y compare con el caso original.

Parte 2: Dependencias de Datos y Riesgos

Las dependencias de datos generan riesgos (hazards) que afectan el pipeline. Analizar cómo identificarlos y resolverlos.

2. Dada la siguiente secuencia de instrucciones:

```
add x2 , x5 , x4
add x4 , x2 , x5
sw x5 , 100 (x2)
add x3 , x2 , x4
```

a. Identifique todas las dependencias de datos en el código (RAW, WAR, WAW). Especifique los registros involucrados y las instrucciones afectadas.
b. De las dependencias RAW (riesgo de datos), ¿cuáles se pueden resolver con adelantamiento (forwarding)? Justifique

3. Diagrama de Tiempos sin Adelantamiento. Para la secuencia:

```
sub x4 , x1 , x2
sw x4 , 4(x2)
sub x5 , x4 , x2
lw x3 , 8(x5)
```

a. Dibuje los diagramas de tiempos sin adelantamiento, indicando las etapas IF, ID, EX, MEM, WB por ciclo.
b. Identifique los atascos (stalls), su causa (dependencia específica) y cuántos ciclos se pierden.

4. Variación con Dependencias. Para la secuencia modificada:

```
sub x4 , x1 , x2
sw x4 , 4(x4)
sub x5 , x4 , x2
lw x3 , 8(x5)
```

a. Dibuje el diagrama de tiempos sin adelantamiento.
b. Compare con el Ejercicio 3 y explique cómo la modificación en sw afecta los riesgos.

5. Adelantamiento de Datos

a. Repita los diagramas de los Ejercicios 3 y 4 con adelantamiento habilitado.
b. Explique cómo el adelantamiento reduce los atascos y cuántos ciclos se ahorran en cada caso.

Parte 3: Saltos y Predicción

Introducción: Los saltos condicionales generan riesgos de control. Analizar su impacto sin predicción.

6. Saltos Condicionales. Para la secuencia:

```
lw $2 , 0 ($4)
beq $1 , $3 , loop
sub $4 , $4 , $5
```

a. Dibuje el diagrama de tiempos sin adelantamiento ni predicción de saltos.
b. Analice los dos casos del beq:
   - Salto tomado.
   - Salto no tomado.
c. ¿Cuántos ciclos se pierden por la falta de predicción en cada caso? Explique.

Parte 4: Optimización de Código

Introducción: Reordenar instrucciones puede minimizar atascos sin cambiar el resultado.

7. Reordenamiento. Dado el código en alto nivel:

   A = B + C
   D = E - F

   Y su traducción a RISC:

```
lw X2, B
lw X3, C
add X1, X2, X3
sw X1, A
lw X5, E
lw X6, F
sub X4, X5, X6
sw X4, D
```

   a. Reordene las instrucciones para reducir atascos sin alterar el resultado.
   b. Dibuje el diagrama de tiempos antes y después del reordenamiento.
   c. Cuantifique la mejora en ciclos ahorrados.

Ejecución con Dependencias

8. Dado el siguiente fragmento de código y suponiendo que el valor inicial de los registros **t1** y **t2** es 11 y 22 respectivamente,

```
addi t1, t2, 5
add t3, t1, t2
addi t4, t1, 15
```

   a. Calcule los valores finales de los registros en un procesador multiciclo.
   b. Calcule los valores finales en un procesador segmentado sin gestión de conflictos.
   c. Inserte nop para corregir el resultado en el segmentado sin adelantamiento.
   d. Verifique los valores finales con adelantamiento habilitado.

Parte 5: Análisis de Pipeline

9. Etapas por Ciclo. Para la secuencia:

```
xor s1, s2, s3
addi s0, s3, -4
lw s3, 16(s7)
sw s4, 20(s1)
or t2, s0, s1
```

   a. Indique la etapa del pipeline de cada instrucción en los 5 primeros ciclos.
   b. Liste los registros leídos y escritos en cada ciclo.
   c. Dibuje el diagrama de ejecución.

10. Comparación con y sin Adelantamiento. Para la secuencia:

```
addi s1, zero, 11
lw s2, 25(s0)
add s3, s3, s4
or s4, s1, s2
lw s5, 16(s2)
```

a. Dibuje el diagrama de ejecución en un pipeline sin adelantamiento para los primeros 7 ciclos.
b. Repita con adelantamiento optimizado (datos de WB a MEM).
c. Compare los ciclos totales y explique cómo el adelantamiento resuelve dependencias.

Simulación de un Bucle con Branch History Table (BHT) en RARS

11. Introducción: Analizar cómo un predictor de saltos (BHT) afecta el rendimiento de un programa con saltos condicionales usando el simulador RARS.

Este programa suma los primeros 10 números naturales utilizando un bucle y luego imprime el resultado. Se utilizan instrucciones de salto condicional (blt) para controlar el flujo del programa.

```
# Este programa suma los primeros 10 números naturales y luego imprime el
resultado

# Inicialización de variables
.data
sumatoria: .word 0  # Variable para almacenar la sumatoria
i: .word 0          # Variable para almacenar el índice del bucle

# Código
.text
main:
    li t0, 0        # Inicializar la variable sumatoria
    li t1, 1        # Inicializar el índice del bucle
    li t2, 10       # Establecer el límite del bucle

bucle:
    add t0, t0, t1  # Sumar el índice actual a la sumatoria
    addi t1, t1, 1  # Incrementar el índice del bucle

    blt t1, t2, bucle # Saltar al bucle si t1 < t2
    # Si t1 >= t2, se ha completado el bucle
    # Imprimir el resultado
    mv a0, t0       # Colocar el resultado en el registro a0
    li a7, 1        # Cargar el código de la llamada al sistema para
    imprimir entero
    ecall           # Realizar la llamada al sistema

    # Terminar la ejecución del programa
    li a7, 10       # Cargar el código de la llamada al sistema para salir
    ecall           # Realizar la llamada al sistema
```

a. Preparación en RARS:
   - Abra RARS y copia el código en el editor.

- Vaya al menú "Tools" y selecciona "Branch History Table Simulator".
- Conecte el BHT al programa marcando "Connect to Program".
- Configure el BHT con un tamaño inicial de 4 entradas y predictor de 1 bit (ajusta si se indica).

b. Ejecución:
- Ensamble el código.
- Ejecute el programa con "Run" (o paso a paso con "Step" para mayor detalle).
- Observe la ventana del BHT durante la ejecución del bucle.

c. Análisis:
- Estado del BHT: Registre cómo cambia el estado del predictor (e.g., "Taken" o "Not Taken") para la instrucción blt en cada iteración del bucle (10 iteraciones).
- Precisión: Calcule la tasa de aciertos (predicciones correctas / total de saltos).
- Impacto: Explique cómo los aciertos y fallos afectan el número de ciclos totales.

d. Modificación:
- Cambie el límite del bucle a 5 (i.e., li t2, 5) y repite la simulación.
- Compare la tasa de aciertos y los ciclos totales con el caso original.
- Sugiere una razón por la que el tamaño del bucle influye en la precisión del BHT.

12. Análisis Avanzado del Pipeline (Revisada con RARS)

Objetivo: Diseñar un programa en RISC-V con tres tipos de saltos condicionales (beq, bne, blt) y analizar su comportamiento en RARS con el BHT Simulator.

Escribe un programa que:
- Use un contador (t0) inicializado en 0.
- Incluya un bucle que itere 6 veces con blt.
- Dentro del bucle, use beq para saltar a una etiqueta si t0 == 3.
- Use bne para saltar a otra etiqueta si t0 != 5.
- Imprima t0 al final.

Ejemplo sugerido:

```
.text
main:
   li t0, 0    # Contador
   li t1, 6    # Límite
 loop:
   addi t0, t0, 1 # Incrementar contador
   beq t0, t1, end # Si t0 == 6, salir
   beq t0, t3, skip # Si t0 == 3, saltar (t3 = 3)
   bne t0, t4, loop # Si t0 != 5, repetir (t4 = 5)
 skip:
   j loop
 end:
   mv a0, t0
   li a7, 1
   ecall
   li a7, 10
   ecall
```

a. Simulación en RARS:

- Cargue el código en RARS y configura el BHT (tamaño = 8 entradas, predictor 1-bit).
- Ejecute el programa y observa el BHT para cada instrucción de salto (beq, bne, blt).

b. Análisis:
- Patrones: Describe el patrón de ejecución de cada salto (e.g., blt se toma 5 veces, no se toma 1 vez).
- Estados del BHT: Registre los estados iniciales y finales del predictor para cada instrucción.
- Tasa de aciertos: Calcule la precisión del BHT (aciertos / total de predicciones).
- Dificultades: Identifique qué salto tuvo más fallos y explica por qué (e.g., patrón irregular).

c. Optimización:
- Modifique el código para hacer los saltos más predecibles (e.g., reordena condiciones).
- Pruebe con un predictor de 2 bits y compara la tasa de aciertos.

13. **Bucle Anidado y Configuraciones del BHT**

Objetivo: Evaluar el impacto de diferentes configuraciones del BHT en un programa con bucles anidados y saltos condicionales.

Cree un programa que sume números pares del 0 al 100 usando bucles anidados:

- Bucle externo: itera 100 veces (blt).
- Bucle interno: verifica si el número es par (beq) y lo suma.
- Imprime la suma final.

Ejemplo sugerido:

```
.data
suma: .word 0
.text
main:
   li t0, 0    # Contador
   li t1, 100  # Límite
   li t2, 0    # Suma

outer_loop:
   li t3, 2    # Divisor para paridad
   beq t0, t1, end # Si t0 == 100, salir
   and t4, t0, 1  # t4 = 1 si impar, 0 si par
   beq t4, zero, add_num # Si par, sumar
   j next
 add_num:
   add t2, t2, t0 # Sumar número par

next:
   addi t0, t0, 1
   blt t0, t1, outer_loop # Repetir

end:
   sw t2, suma, t3
   mv a0, t2
   li a7, 1
```

```
ecall
li a7, 10
ecall
```

a. Simulación en RARS:
   - Cargue el código y activa el BHT Simulator.
   - Pruebe tres configuraciones:
     - Config 1: 8 entradas, predictor 1-bit.
     - Config 2: 8 entradas, predictor 2-bit.
       Ejecuta el programa para cada configuración.

b. Análisis:
   - Tasa de aciertos: Compare la precisión del BHT para blt y beq en cada configuración.
   - Ciclos totales: Registre el número de ciclos (usa "Settings" > "Cycle Count" en RARS).
   - Impacto: Explique cómo el tamaño y el tipo de predictor afectan el rendimiento.

14. Supongamos que sobre el procesador RISC, se ejecuta un programa de 500 instrucciones distribuidas de la siguiente manera:
    - El 20% corresponden a instrucciones de **lw**, y de éstas, en la mitad de los casos vienen seguidas de una instrucción aritmética que lee el registro sobre el que escribe la instrucción **lw**.
    - El 15% son instrucciones de **sw**.
    - El 25% son instrucciones **beq**, en las que la condición evaluada se cumple en el 70% de los casos.
    - El 5% se corresponden con instrucciones **jal**.
    - El 35% restante son instrucciones aritmético-lógicas.

    Suponiendo que el procesador funciona a una frecuencia de 1.5 GHz.
    a. Calcule el CPI obtenido en la ejecución del programa.
    b. Calcule el tiempo de ejecución de este.

15. En un procesador RISC-V segmentado diseñado con tecnología CMOS de 90 nm, se aplican las siguientes reducciones: 30% en el retardo de la ALU, 10% en el extensor de signo y 20% en los retardos de lectura/escritura del banco de registros. Para un programa de 100 millones de instrucciones con la distribución indicada:

	tipo-I/R	**lw**	**sw**	**beq** saltan	**beq** no saltan	**jal**
frecuencia	60%	15%	10%	10%	4%	1%

    a. Calcule el retardo del camino crítico en cada etapa del pipeline (IF, ID, EX, MEM, WB) tras las optimizaciones.
    b. Determine el speed-up obtenido respecto al diseño original, considerando atascos por dependencias."