Guía para el Trabajo Práctico Número 5

Arquitectura de Computadoras 6/05/2026

Problemas Propuestos

4) *[Modificado]* Se tiene un procesador que implementa el ISA RV32I del RISC-V con un diseño en pipeline de cinco etapas. Sobre este procesador se ejecuta el siguiente lazo:

```
    .data
vector:
    .word   72,57,39,48,46,88,2,91,0

    .text
    .globl  main

    la a0, vector       # Carga la direccion base del vector
    li a1, 10           # Carga la cantidad de elementos del vector
    li a2, 0            # Inicializa el contador de elementos impares
lazo:
    addi a1, a1, -1     # Decrementa el contador de elementos
    beqz a1, fin        # Termina ciclo si elemento es cero
    lw   t0, 0(a0)      # Carga el elemento a utilizar
    addi a0, a0, 4      # Incrementa puntero a siguiente elemento
    andi t0, t0, 1      # Enmascara ultimo bit para determinar paridad
    beqz t0, par        # Salta si el numero es par
    addi a2, a2, 1      # Incrementa contador de impares
par:
    j lazo              # Repite lazo para siguiente elemento
fin:
```

a) Si el procesador sigue el diseño del primer procesador segmentado visto en clase, pero incorpora todos los adelantamientos necesarios para resolver los riegos de datos, ¿cuántos ciclos demora en este procesador la ejecución del lazo propuesto?
c) Suponga ahora que se adiciona una unidad de predicción de saltos que posee un 100 % de acierto, de manera tal que cuando un salto está en la etapa de Decode entonces la instrucción que ingresa al Fetch es siempre la correcta. En este caso, ¿cuántos ciclos demora en este procesador la ejecución del lazo propuesto?
d) Suponga ahora que se utiliza una predicción estática de saltos, en la cual los saltos hacia atrás se predicen como tomados y los saltos hacia adelante se predicen como que no serán tomados. En este caso, ¿cuántos ciclos demora en este procesador la ejecución del lazo propuesto?

6) *[Modificado]* A fin de incrementar la performance del procesador en pipeline visto en clase, se va a diseñar un nuevo procesador consistente en dos pipes en paralelo. Este procesador será capaz de procesar dos instrucciones RV32-I en paralelo.
a) ¿Qué riesgos estructurales podría haber entre dos instrucciones en dos pipelines en paralelo y cómo los resolvería? Considere Memoria de Instrucciones, Memoria de Datos, Banco de Registros, ALU, etc.
b) ¿Es necesario que los circuitos de adelantamiento pasen información de un pipe al otro? No se pide diseñarlos. Justifique su respuesta.
c) ¿Entre dos instrucciones que se ejecuten en paralelo podrían existir riesgos WAR? En caso positivo dé un ejemplo, caso contrario explique su respuesta.
d) ¿Entre dos instrucciones que se ejecuten en paralelo podrían existir riesgos WAW? En caso positivo dé un ejemplo, caso contrario explique su respuesta.
e) A fin de analizar los riesgos RAW, suponga que cada pipe contiene todos los circuitos de adelantamiento necesarios, ¿Existen riesgos RAW?. En caso positivo dé un ejemplo, caso contrario explique su respuesta.

8) Considere el siguiente segmento de programa en lenguaje ensamblador de RISC-V:

```
(1)  add  x1, x2, x3
(2)  sub  x5, x4, x5
(3)  lw   x4, 0(x13)
```

```
(4)  mul  x4, x4, x4
(5)  sw   x4, 0(x13)
(6)  lw   x9, 0(x10)
(7)  lw  x11, 0(x12)
(8)  add x11, x11, x12
(9)  mul x11, x11, x11
(10) sw  x11, 0(x12)
```

a) Realice el diagrama de precedencia para el segmento de programa mostrado.
b) Si se ejecuta el código en el procesador segmentado visto en clase, ¿cuántos ciclos tarda en ejecutarse?
c) Si se ejecuta el código en un procesador superescalar que puede ejecutar hasta cuatro instrucciones en paralelo, pero que finaliza las instrucciones *en el mismo orden* en que llegan, ¿cuántos ciclos tarda en ejecutarse?
d) Si se ejecuta el código en un procesador superescalar que puede ejecutar hasta cuatro instrucciones en paralelo, y que puede finalizar las instrucciones *en cualquier orden*, ¿cuál es el mínimo tiempo que se necesita para ejecutar el código?

10) *[Modificado]* En el trabajo de Laboratorio 1, se vio un ejercicio en el cual se buscaba una implementación del algoritmo GEMM para multiplicar matrices. A continuación, se muestra una versión optimizada del lazo más interno de dicho algortimo:

```
L3:   lw   s11, 0(t2)     # s11 = B[k][j]
      lw   s10, 0(t1)     # s10 = A[i][k]
      mul  s11, s10, s11  # s11 = A[i][k] * B[k][j]
      add  s0, s0, s11    # s0 = C[i][j] + A[i][k] * B[k][j]

      add  t2, t2, s3     # t2 += N (siguiente elemento de B)
      addi t1, t1, 4      # t1++ (siguiente elemento de A)
      bltu t2, s5, L3     # si k < N, saltamos a L3
```

Suponga que el lazo anterior se ejecuta un gran número de veces, y por lo tanto se desea optimizarlo utilizando la técnica de desenrollado de lazos.
a) ¿Cuántos ciclos demora cada iteración del lazo y cuántos bytes ocupa en memoria?
b) Desenrollar 1 sola vez y reordenar el código. ¿Cuántos ciclos demora ahora cada iteración y cuántos bytes ocupa en memoria?

Soluciones Completas

4)
a) El primer procesador estudiado en clase define la dirección del salto en la etapa MEM, y por lo tanto requiere insertar tres burbujas después de un salto para conocer cual es la instrucción que debe buscarse en memoria.
Para determinar el tiempo total de ejecución del lazo debemos tener en cuenta que cada iteración del lazo ejecuta ocho instrucciones cuando el numero es impar y siete cuando es par. De las instrucciones del cuerpo del lazo una es un salto incondicional, otra es un salto condicional que se toma solo en la ultima iteración, otra es un salto condicional que se toma con los números pares.
Dado que el vector tiene cinco elementos pares y tres elementos impares entonces el lazo ejecuta 7 * 5 + 3 * 8 = 59 instrucciones a las cuales se suman las cuatro instrucciones previas al lazo, **la** es una pseudo-instrucción que se traduce en dos instrucciones, y las dos que terminan la ejecución del mismo. De este análisis resulta entonces que para completar el programa del ejemplo se ejecutan 59 + 4 + 2 = 65 instrucciones, de las cuales ocho son saltos incondicionales, once son saltos condicionales no tomados (ocho iteraciones mas tres números impares), seis son saltos condicionales tomados (ultima iteración más tres números pares).
El primer procesador estudiado en clase define la dirección del salto en la etapa MEM, y por lo tanto requiere insertar tres burbujas después de cada salto para conocer cual es la instrucción que debe buscarse en memoria. Sabiendo entonces que los 8 + 11 + 6 = 25 saltos demoran 4 T en ejecutarse, mientras que las 40 instrucciones restantes requieren 1 T, entonces podemos calcular el tiempo total de ejecución del ejemplo como 4 * 25 + 40 = 140 T.
b) Si se agrega una unidad de predicción de saltos que nunca falla entonces siempre se busca la instrucción correcta después de un salto, y por lo tanto todas las instrucciones van a

demorar 1 T. En estas condiciones el tiempo total de ejecución sería simplemente 65 T.

c) Revisando el código propuesto se observa que ambos saltos condicionales son para adelante y el salto incondicional es para atrás. Del análisis efectuado al inicio se sabe que once de los saltos condicionales no son tomados y en estos casos la predicción será correcta y el tiempo de ejecución de los mismos será de 1 T. Por otra parte seis de los saltos condicionales si serán tomados y en estos casos la predicción será incorrecta y el tiempo de ejecución de los mismos será de 2 T. Finalmente los ocho saltos incondicionales tendrán una predicción correcta y demorarán 1 T en completar la ejecución. Por lo tanto, el tiempo total de ejecución será de 11 * 1 + 6 * 2 + 8 * 1 + 40 * 1 = 71 T.

6)

a) La memoria de instrucciones debe ser la misma para los dos pipelines, ya que no tiene mucho sentido tener dos copias del mismo programa. La memoria de datos también debe ser única, porque duplicar memoria no es recomendable, ya que se podrían generar inconsistencias en los datos entre las dos memorias. Por las mismas razones, el banco de registros también debe ser único. Sin embargo, debemos considerar que el banco de registros debe ser capaz de devolver los contenidos de cuatro registros simultáneamente y escribir dos, para no disminuir la performance del pipe, y que la memoria debe ser capaz de escribir o leer dos datos al mismo tiempo, por las mismas razones. Se deben tener dos ALUs independientes que trabajen simultáneamente para no tener riesgos estructurales.

b) Es necesario, porque una instrucción que esté en un pipeline puede tener una dependencia con otra instrucción que está en el otro pipeline, de manera que adelantar los valores del otro pipe permite minimizar los riesgos de tipo RAW.

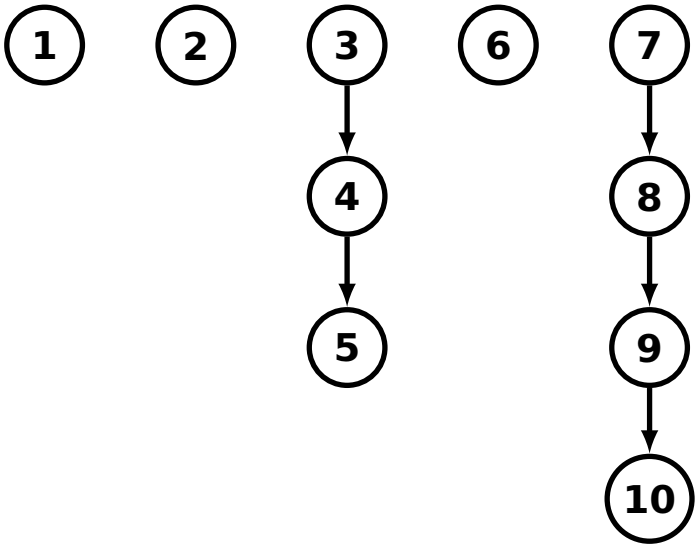
c) No, no es posible que ocurran riesgos WAR, ya que se siguen los criterios de diseño del procesado visto en clase: se escribe siempre en la última etapa, todas las instrucciones pasan por todas las etapas y todas las etapas tienen igual duración.

d) Sí podrían ocurrir riesgos WAW, ya que los dos pipelines realizan las escrituras al mismo tiempo en la misma etapa (sean registros o memoria). Si se utilizará emisión estática el encargado de evitar estos riesgos debería ser el compilador. En los procesador más modernos se utiliza emisión dinámica y se transfiere la responsabilidad al control.

e) Al ser cada pipe idéntico al del procesador visto en clase, sigue existiendo el riesgo RAW del load retardado., Pero además, se suman nuevos riesgos entre instrucciones que se ejecutan simultáneamente en ambos pipelines, ya que es imposible de adelantar resultados al vuelo, los adelantamientos siempre se hacen entre instrucciones siguientes.

8)

a) El diagrama de precedencia sería como se muestra a continuación:



b) Cada instrucción demora un ciclo, y además hay una parada por la dependencia en **x4** entre las instrucciones 4 y 5, y otra parada por la dependencia en **x11** entre las instrucciones 7 y 8, por lo que en total la ejecución demora 12 ciclos.

c) Al colocar 4 unidades de ejecución en paralelo, puedo ejecutar más de una instrucción en cada ciclo, pero como la finalización debe ser en orden, la secuencia de ejecución sería

como la siguiente:

T0: ejecuto 1, 2 y 3
T1: tengo que esperar por la dependencia RAW de **x4** entre 3 y 4.
T2: ejecuto 4.
T3: ejecuto 5, 6 y 7.
T4: tengo que esperar por la dependencia RAW de **x11** entre 7 y 8.
T5: ejecuto 8.
T6: ejecuto 9.
T7: ejecuto 10 y termino.

En un procesador superescalar que puede ejecutar hasta 4 instrucciones en paralelo, demoro 8 ciclos T en ejecutar la secuencia dada en orden. Vale la pena destacar que en ninguno de los ciclos se ejecutan 4 instucciones al mismo tiempo, y que en dos de los ciclos no se ejecuta ninguna instrucción. Podría también ejecutar 9 y 10 en paralelo usando adelantamiento. Lo cual haría que demore 7T.

d) Sin limitaciones de ningún tipo, se intentaría ejecutar las instrucciones 1, 2, 3, 6 y 7 en paralelo, ya que son independientes. En cambio, la instrucción 4 depende de la 3, la 5 de la 4, la 8 de la 7, la 9 de la 8 y la 10 de la 9.
Podemos observar dos cosas: no se pueden ejecutar cinco instrucciones en paralelo, y el camino de dependencias más largo empieza por la instrucción 7, así que es esta instrucción la que debemos adelantar lo máximo posible.

T0: ejecuto 1, 2, 3 y 7
T1: ejecuto 6.
T2: ejecuto 4 y 8.
T3: ejecuto 5 y 9.
T4: ejecuto 10 y termino.

En un procesador superescalar que puede ejecutar hasta 4 instrucciones en paralelo, el menor tiempo en el que se puede ejecutar la secuencia dada fuera de orden es de *5 ciclos T*.

10)

a) El programa original demora 8 T por iteración (se incorpora una burbuja antes del **mul** debido a la penalización lw). El código ocupa 7 x 4 bytes = 28 bytes en memoria.

b) Para realizar un desenrollado de lazo es importante tener especial atención en los desplazamientos de loads y stores y en los desplazamientos de los índices para no alterar el funcionamiento normal del código original. En un primer paso sólo desenrollamos el lazo pero no realizamos ningún reordenamiento de las instrucciones:
Debe tenerse en cuenta que al duplicar la iteración original, el control de lazo no se duplica, pero deben cambiarse los valores para que salte el doble de elementos.

```
L3:   lw   s11, 0(t2)     # s11 = B[k][j]
      lw   s10, 0(t1)     # s10 = A[i][k]
      mul  s11, s10, s11  # s11 = A[i][k] * B[k][j]
      add  s0, s0, s11    # s0 = C[i][j] + A[i][k] * B[k][j]

      lw   s9, 4(t2)      # s9 = B[k+1][j]
      lw   s8, 4(t1)      # s8 = A[i][k+1]
      mul  s9, s8, s9     # s11 = A[i][k+1] * B[k+1][j]
      add  s0, s0, s9     # s0 = C[i][j] + A[i][k+1] * B[k+2][j]

      slli s4, s3, 1      # s4 permite saltar dos elementos
      add  t2, t2, s4     # t2 += N (salto dos elementos de B)
      addi t1, t1, 8      # t1++ (salto dos elementos de A)
      bltu t2, s5, L3     # si k < N, saltamos a L3
```

Una vez desenrollado el lazo, procedemos a ordenar el código procurando evitar las paradas (burbujas) ocasionadas por riesgos de datos entre las instrucciones **mul** y los **lw** que las preceden.
En este caso reordenamos el código manteniendo “la estructura original”, es decir, primero todas las cargas, luego todas las operaciones aritméticas y finalmente todos los almacenamientos. El código quedaría de la siguiente manera:

```
L3:   lw   s11, 0(t2)     # s11 = B[k][j]
      lw   s10, 0(t1)     # s10 = A[i][k]
      lw   s9, 4(t2)      # s9 = B[k+1][j]
      lw   s8, 4(t1)      # s8 = A[i][k+1]

      mul  s11, s10, s11  # s11 = A[i][k] * B[k][j]
      add  s0, s0, s11    # s0 = C[i][j] + A[i][k] * B[k][j]
      mul  s9, s8, s9     # s11 = A[i][k+1] * B[k+1][j]
      add  s0, s0, s9     # s0 = C[i][j] + A[i][k+1] * B[k+1][j]

      slli s4, s3, 1      # s4 permite saltar dos elementos
      add  t2, t2, s4     # t2 += N (salto dos elementos de B)
      addi t1, t1, 8      # t1++ (salto dos elementos de A)
      bltu t2, s5, L3     # si k < N, saltamos a L3
```

De este modo desenrollando el lazo una vez obtenemos una iteración cada 6T y el programa en este caso ocupa 12 x 4 bytes = 48 bytes.
Podemos calcular la aceleración lograda teniendo en cuenta que en el programa principal obteníamos una iteración cada 8T. De esta manera se logra una aceleración de 33 % (8T/6T = 1,33) desenrollando una vez el lazo pero esto conlleva un aumento en la memoria utilizada.