

**EL PROGRAMA UNIVERSITARIO DE LA IMAGINACIÓN**

**Laboratorio RVfpga 17**

## **Ejecución superescalar**

# INTRODUCCIÓN

Como ya hemos discutido, el procesador SweRV EH1 de Western Digital es un núcleo **superescalar bidireccional de 32 bits con canalización de 9 etapas** . En las prácticas de laboratorio 11 a 13, analizamos el flujo de las instrucciones básicas a través del procesador SweRV y los detalles de cada una de las etapas de la canalización, y en las prácticas de laboratorio 14 a 16 analizamos cómo se manejan los datos, el control y las dependencias estructurales en este procesador. . Ahora, con esa comprensión fundamental, ¡está listo para analizar la ejecución superescalar!

**NOTA:** Antes de comenzar con este laboratorio, recomendamos leer la Sección 7.7.4 del libro de S. Harris y D. Harris, “ *Diseño digital y arquitectura de computadoras: Edición RISC-V* ”, Morgan Kaufmann [ DDCARV ]. Parte del contenido de este laboratorio está inspirado en ese libro.

La ejecución superescalar es una técnica de microarquitectura que mejora el rendimiento de un procesador. Un procesador superescalar contiene varias copias del hardware de ruta de datos para ejecutar varias instrucciones simultáneamente. La latencia de ejecutar una sola instrucción permanece sin cambios, pero el procesador puede ejecutar y confirmar más instrucciones por ciclo, mejorando así su rendimiento. La figura 1 muestra un diagrama de bloques de ejemplo de un procesador superescalar bidireccional extraído de [DDCARV].

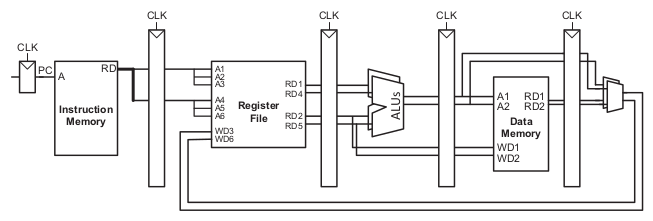


figura 1\_ Figura 7.68 de [DDCARV]: Diagrama de bloques del procesador superescalar bidireccional

SweRV EH1 es un procesador superescalar bidireccional similar al que se muestra en la Figura 1 que puede obtener, ejecutar y confirmar hasta dos instrucciones por ciclo. La ruta de datos alinea dos instrucciones a la vez que provienen de la memoria de instrucciones. El archivo de registro de puertos múltiples puede leer hasta cuatro operandos de origen y escribir dos valores en cada ciclo (más un valor más procedente de una carga sin bloqueo, como analizamos en la práctica de laboratorio 15). El procesador SweRV EH1 contiene dos Integer Pipes, un Multiply Pipe, un Load-Store Pipe y un Divider no segmentado. Todos estos conductos son completamente independientes, por lo que un par de instrucciones aritmético-lógicas (AL) independientes o cualquier par de dos instrucciones diferentes independientes pueden ejecutarse simultáneamente. Sin embargo, como se discutió en la práctica de laboratorio 14, un par de multiplicaciones, divisiones o instrucciones de carga/almacenamiento no se pueden ejecutar en el mismo ciclo, porque solo hay uno de cada uno de estos conductos en el procesador y, por lo tanto, dos secuenciales iguales que no son AL. instrucciones conducirá a un peligro estructural.

SweRV EH1 no incluye soporte para programación dinámica de instrucciones con ejecución fuera de orden, excepto en el caso de cargas sin bloqueo. Sin embargo, es posible reordenar estáticamente el código para aprovechar mejor los recursos, incluidas las dos formas de canalización. Idealmente, en un procesador superescalar bidireccional como SweRV EH1, el rendimiento (IPC) se duplicaría en comparación con un diseño de un solo problema. Desafortunadamente, los programas reales no suelen alcanzar ese ideal: en los programas reales, el rendimiento suele mejorar entre 1,3 y 1,5 veces cuando se pasa de procesadores de 1 a 2 vías ; sin embargo, agregar la segunda forma requiere mucho más hardware.

En la Sección 2, analizamos dos programas simples, comparando el comportamiento cuando se usan configuraciones de un solo problema y de dos problemas de SweRV EH1. Luego, en la Sección 3, proponemos varios ejercicios relacionados con la ejecución superescalar.

# EMISIÓN ÚNICA VS. EMISIÓN DOBLE

En esta sección trabajamos con dos programas simples: el primero (Sección 2.A) contiene un bucle con cuatro instrucciones AL y el segundo (Sección 2.B) contiene un bucle con dos multiplicaciones intercaladas con dos instrucciones AL.

1. **Cuatro instrucciones AL independientes**

En esta sección, comparamos el rendimiento del programa de la Figura 2 que se ejecuta en el núcleo SweRV EH1 de uno o dos problemas. Recuerde que, en el Apéndice B del Laboratorio 11, describimos cómo se pueden configurar las diferentes funciones principales (ejecución canalizada, predicción de bifurcación, superescalar, etc.).

El programa contiene un bucle que realiza 1 000 000 (0xF4240) iteraciones; el cuerpo del ciclo contiene cuatro instrucciones independientes AL ( add , sub , or y xor ), rodeadas por nop instrucciones que nos permiten ver cada iteración aislada de las demás. La carpeta *[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions* proporciona el proyecto PlatformIO que puede analizar, simular y modificar según lo desee.

|  |
| --- |
| .globl Test\_Asamblea  .texto  Asamblea\_de\_prueba:  li t2, 0x400 # Deshabilitar la ejecución de problemas duales csrrs t1, 0x7F9, t2  li t0, 0x0 li t1, 0x1li t2, 0x1li t3, 0x3 li t4, 0x4 li t5, 0x5 li t6, 0x6  lui t2, 0xF4 agregar t2, t2, 0x240  REPETIR: agregar t0, t0, 1 INSERT\_NOPS\_10 INSERT\_NOPS\_4 **add t3, t3, t1 sub t4, t4, t1 o t5, t5, t1 xor t6, t6, t1** INSERT\_NOPS\_10 INSERT\_NOPS\_3 bne t0, t2, REPETIR # Repetir el ciclo  .final |

figura 2\_ Programa con cuatro instrucciones AL

En la Sección 2.Ai, analizamos la simulación en Verilator y la ejecución en la placa Nexys A7 para el programa de la Figura 2 en un procesador SweRV EH1 de un **solo problema .** Para ese propósito, deshabilitamos la capacidad de emisión dual usando las siguientes dos instrucciones al comienzo del programa:

li t2, 0x400

csrrs t1, 0x7F9, t2

En la Sección 2.A.ii, analizamos la simulación en Verilator y la ejecución en la placa Nexys A7 para el programa de la Figura 2 en un procesador SweRV EH1 **de doble emisión .** Para ello, simplemente comentamos las dos instrucciones anteriores.

1. **Ejecución en un procesador SweRV EH1 de un solo problema**

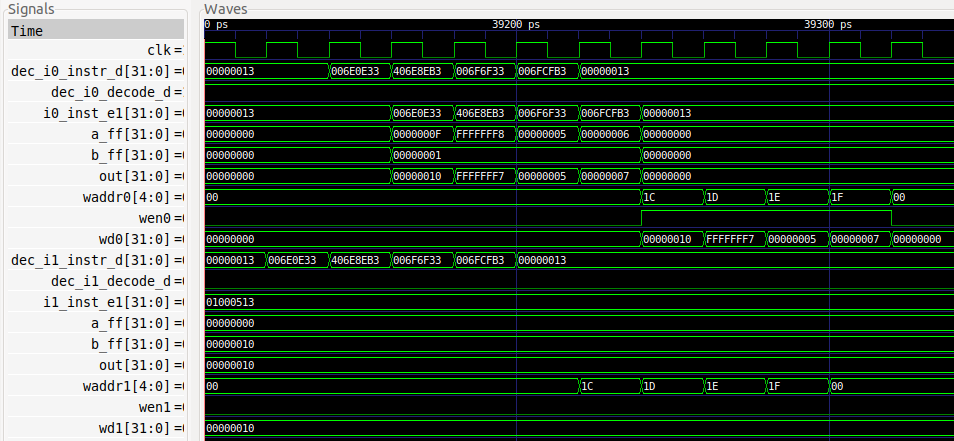
En su configuración de un solo problema, SweRV EH1 simplemente ignora la segunda forma, ejecutando solo una instrucción por ciclo. La Figura 3 ilustra la simulación del programa de la Figura 2 en esta configuración de SweRV EH1. Elegimos una iteración aleatoria del bucle *REPETIR (excepto la primera), dado que todas las iteraciones son iguales.*

**add**

**sub**

**or**

**xor**



**Writeback**

**Writeback**

**Decode**

**Decode**

**EX1 (I0 Pipe)**

**EX1 (I1 Pipe)**

**1 2 3 4 5 6 7 8 9**

**WAY0**

**WAY1**

figura 3\_ Simulación del programa de la Figura 2 en un SweRV EH1 de un solo problema

Las instrucciones se reciben en ambos sentidos en el momento de la decodificación (ver señales dec\_i0\_instr\_d[31:0] y dec\_i1\_instr\_d[31:0] ), pero solo se envían a ejecución en el Camino 0, porque el Camino 1 está deshabilitado. Las señales dec\_i0\_decode\_d y dec\_i1\_decode\_d determinan si la instrucción se propaga desde la etapa de decodificación a la etapa EX1, y las señales i0\_inst\_e1[31:0] e i1\_inst\_e1[31:0] contienen la instrucción en la vía 0 y la vía 1, respectivamente, en la etapa E1 .

* **Vía 0** :
  + La señal dec\_i0\_decode\_d siempre es 1 en nuestro ejemplo; específicamente, es 1 para las cuatro instrucciones AL bajo análisis.
  + La instrucción en la etapa de decodificación ( dec\_i0\_instr\_d[31:0] ) **se propaga** a la tubería I0 ( i0\_inst\_e1[31:0] )
* **Vía 1** :
  + La señal dec\_i1\_decode\_d siempre es 0 en nuestro ejemplo; específicamente, es 0 para las cuatro instrucciones AL bajo análisis.
  + La instrucción en la etapa de decodificación ( dec\_i1\_instr\_d[31:0] ) **NO se propaga** ( i1\_inst\_e1[31:0] ) a la etapa de ejecución.

En consecuencia, solo se usa la ALU de la tubería I0 (ver señales aff , bff y out en ambos sentidos) y solo se usa el puerto de escritura 0 del archivo de registro (ver señales waddr , wen y wd en ambos sentidos).

La figura 4 ilustra el flujo de las cuatro instrucciones AL a través de la tubería I0, desde la etapa de decodificación hasta la de reescritura (EX5). Muestra los nueve ciclos (1 a 9) especificados en la Figura 3 . Los huecos vacíos corresponden a las instrucciones nop que rodean a las cuatro instrucciones AL, que no se muestran en la figura por motivos de simplicidad.



figura 4\_ Flujo de las 4 instrucciones AL a través de la tubería I0

**TAREA :** En la simulación de la Figura 3 , incluya las señales de seguimiento y resalte las instrucciones a medida que pasan por la canalización desde la etapa de decodificación hasta la etapa de reescritura, de manera similar a la Figura 4 . Puede usar el archivo *.tcl* proporcionado en: *[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions/test\_task1.tcl* .

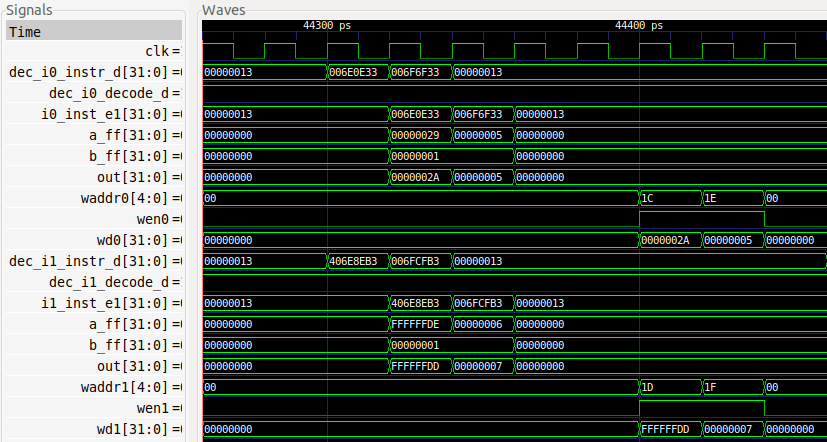
**TAREA :** Quitartodas las instrucciones nop dentro del cuerpo del ciclo de la Figura 2 .

Repita la simulación de la Figura 3 . ¿Cuál es el IPC esperado para este programa?

Ejecuta el programa en la placa y verifica que el IPC obtenido es el que esperabas.

1. **SweRV EH1**

Comente las dos instrucciones de la Figura 2 para habilitar la capacidad de emisión dual de SweRV EH1. Ahora, SweRV EH1 enviará a ejecución dos instrucciones por ciclo siempre que sea posible. La Figura 5 ilustra la simulación del programa. Como antes, elegimos una iteración aleatoria del ciclo *REPETIR* (excepto la primera), dado que todas las iteraciones son iguales.



**Decode**

**Writeback**

**Writeback**

**EX1 (I1 Pipe)**

**Decode**

**or**

**add**

**WAY1**

**EX1 (I0 Pipe)**

**1 2 3 4 5 6 7**

**WAY0**

**xor**

**sub**

figura 5\_ Simulación del programa de la Figura 2 en un SweRV EH1 de doble emisión

Se reciben dos instrucciones, una en cada sentido, en el momento de la decodificación (véanse las señales dec\_i0\_instr\_d[31:0] y dec\_i1\_instr\_d[31:0] ), y se envían dos instrucciones por ciclo a las etapas Execute, una a través del I0 Pipe y la otro a través de la tubería I1.

* **Vía 0** :
  + La señal dec\_i0\_decode\_d siempre es 1, lo que es cierto para dos de las cuatro instrucciones AL de nuestro ejemplo (las otras dos instrucciones AL se decodifican en la Vía 1).
  + La instrucción en la etapa Decode ( dec\_i0\_instr\_d[31:0] ) se propaga a la tubería I0 ( i0\_inst\_e1[31:0] ).
* **Camino 1** :
  + La señal dec\_i1\_decode\_d siempre es 1, lo que es cierto para dos de las cuatro instrucciones AL de nuestro ejemplo (las otras dos instrucciones AL se decodifican en la Vía 0).
  + La instrucción en la etapa Decode ( dec\_i1\_instr\_d[31:0] ) se propaga a la tubería I1 ( i1\_inst\_e1[31:0] ).

Por lo tanto, se utilizan las ALU en ambos conductos (I0 e I1) (vea las señales aff , bff y out en ambos sentidos), y se usan ambos puertos de escritura del archivo de registro (vea las señales waddr , wen y wd en ambos sentidos).

La figura 6 ilustra el flujo de las cuatro instrucciones AL a través de la tubería I0 y la tubería I1, desde la etapa de decodificación hasta la etapa EX5. Muestra los siete ciclos (1 a 7) especificados en la Figura 5 . Los huecos vacíos corresponden a las instrucciones nop que rodean a las cuatro instrucciones AL, que no se muestran en la figura por motivos de simplicidad.



figura 6\_ Flujo de las 4 instrucciones por ambos conductos (I0 e I1)

**TAREA :** en la simulación de la Figura 5 , agregue señales de seguimiento y resalte las instrucciones a medida que pasan por la canalización desde las etapas de Decodificación a Escritura diferida, similar a lo que se muestra en la Figura 6 . Puede usar el archivo *.tcl* proporcionado en: *[RVfpgaPath]/RVfpga/Labs/Lab17/Four\_AL\_Instructions/test\_task2.tcl* .

**TAREA :** Quitartodas las instrucciones nop dentro del cuerpo del ciclo de la Figura 2 .

Repita la simulación de la Figura 5 . ¿Cuál es el IPC esperado para este programa?

Ejecuta el programa en la placa y verifica que el IPC obtenido es el que esperabas.

1. **(OPCIONAL)** **Dos instrucciones mul intercaladas con dos instrucciones AL**

En esta sección analizamos el programa de la Figura 7 en un núcleo SweRV EH1 de doble emisión. El programa ejecuta un bucle que realiza 1.000.000 de iteraciones; el cuerpo del bucle contiene dos instrucciones mul intercaladas con dos instrucciones AL ( mul , add , mul y sub ), rodeadas por nop instrucciones que nos permiten ver cada iteración aislada de las demás. Las cuatro instrucciones son independientes. Analizamos la simulación en Verilator, resaltando las principales señales y brindando una breve explicación del comportamiento del programa en cada caso. La carpeta *[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL\_TwoMUL\_Instructions* proporciona el proyecto PlatformIO que puede analizar, simular y modificar según lo desee.

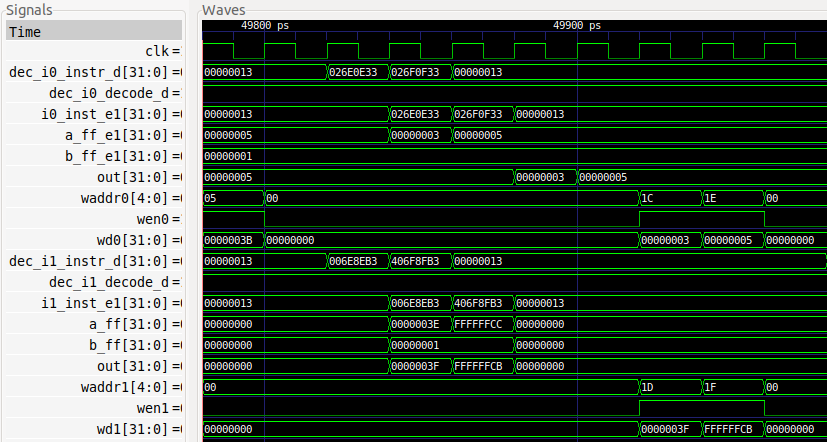
|  |
| --- |
| .globl Test\_Asamblea  .texto  Asamblea\_de\_prueba:  # li t2, 0x400 # Deshabilitar ejecución de problema dual # csrrs t1, 0x7F9, t2 li t3, 0x3 li t4, 0x4 li t5, 0x5 li t6, 0x6 li t0, 0x0lui t1, 0xF4add t1, t1, 0x240  REPETIR: agregar t0, t0, 1 INSERT\_NOPS\_10 INSERT\_NOPS\_4 **mul t3, t3, t1 agregar t4, t4, t1 mul t5, t5, t1 sub t6, t6, t1** INSERT\_NOPS\_10 INSERT\_NOPS\_3 bne t0, t1, REPETIR # Repetir el ciclo  .final |

figura 7\_ Programa con 2 instrucciones mul y 2 instrucciones AL

En este programa, el procesador enviará a ejecución una instrucción mul y una instrucción AL por ciclo, por lo que se utilizan tanto el Multiply Pipe como el I0 o I1 Pipe. La figura 8 ilustra la simulación del programa de la figura 7 en el procesador superescalar de 2 vías SweRV EH1. Elegimos una iteración aleatoria del bucle *REPETIR (excepto la primera), dado que todas las iteraciones son iguales.*

**mul**

**mul**



**Writeback**

**Writeback**

**Decode**

**Decode**

**EX1 (I1 Pipe)**

**EX1 (Multiply Pipe)**

**WAY1**

**WAY0**

**sub**

**add**

figura 8\_ Simulación del programa de la Figura 7

Las instrucciones se reciben en ambos sentidos en el momento de la decodificación (véanse las señales dec\_i0\_instr\_d[31:0] y dec\_i1\_instr\_d[31:0] ) y se envían a las etapas de ejecución en ambos sentidos.

* **Vía 0** :
  + La señal dec\_i0\_decode\_d siempre es 1, para dos de las cuatro instrucciones analizadas en nuestro ejemplo (las otras dos instrucciones se decodifican en la Vía 1).
  + La instrucción en la etapa de decodificación ( dec\_i0\_instr\_d[31:0] ) se propaga a Multiply Pipe ( i0\_inst\_e1[31:0] )
* **Camino 1** :
  + La señal dec\_i1\_decode\_d siempre es 1, para dos de las cuatro instrucciones analizadas en nuestro ejemplo (las otras dos instrucciones se decodifican en la Vía 1).
  + La instrucción en DECO ( dec\_i1\_instr\_d[31:0] ) se propaga a la tubería I1 ( i1\_inst\_e1[31:0] )

Por lo tanto, se utilizan la ALU de la tubería I1 y el multiplicador (consulte las señales a\_ff\_e1 , b\_ff\_e1 y out y las señales a\_ff , b\_ff y out ), y se utilizan ambos puertos de escritura del archivo de registro (consulte las señales waddr , wen y wd en ambos sentidos).

**TAREA :** En la simulación de la Figura 8 , agregue señales de rastreo y resalte las instrucciones a medida que pasan por la canalización desde las etapas de Decodificación a Escritura diferida. Puede usar el archivo *.tcl* proporcionado en: *[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL\_TwoMUL\_Instructions /test\_taskMuls.tcl* .

**TAREA :** Quitartodas las instrucciones nop dentro del cuerpo del ciclo de la Figura 7 .

Repita la simulación de la Figura 8 . ¿Cuál es el IPC esperado para este programa?

Ejecuta el programa en la placa y verifica que el IPC obtenido es el que esperabas.

Repita los mismos experimentos para la configuración de un solo problema y compare los resultados.

# EJERCICIOS

1. Cree programas similares a los de las Figuras 2 y 7 utilizando combinaciones de instrucciones que muestren nuevas situaciones relacionadas con la ejecución de problemas duales.
2. **(OPCIONAL)** Analice las diferencias entre el procesador SweRV EH1 (doble edición) y el procesador superescalar de ejemplo propuesto en la Sección 7.7.4 del libro de texto de S. Harris y D. Harris, " Diseño digital y arquitectura informática: Edición RISC-V " [DDCARV ] (se muestra en la Figura 1 por conveniencia).
3. **(OPCIONAL)** Analice el programa de la Figura 7.70 en la Sección 7.7.4 de DDCARV, que se proporciona en un proyecto de PlatformIO en la carpeta *[RVfpgaPath]/ RVfpga/Labs/Lab17/DDCARV\_SuperscalarExample* . Ejecute el programa en SweRV EH1, tanto en simulación como en la placa (para este último, elimine las instrucciones nop). Explique los resultados. Si es necesario, reordenar el programa tratando de obtener el IPC óptimo.

A continuación, deshabilite la ejecución de dos problemas como se explica en esta práctica de laboratorio y en SweRVref.docx (Sección 2). Compare la simulación y los resultados obtenidos en la pizarra cuando se comparan con cuando la función de emisión dual está habilitada.

1. **(OPCIONAL)** Modifique el programa del ejercicio 3 sustituyendo la instrucción sumar s9 , s8, t1 por la instrucción sumar t2 , s8, t1 . Explique los resultados. Si es necesario, reordene el programa para intentar obtener el IPC óptimo.

Luego deshabilite la ejecución de dos problemas como se explica en esta práctica de laboratorio y en SweRVref.docx (Sección 2). Compare la simulación y los resultados obtenidos en la pizarra cuando se comparan con cuando la función de emisión dual está habilitada.

1. **(OPCIONAL)** ( *El siguiente ejercicio se basa en el ejercicio 4.31 del libro “Computer Organization and Design – RISC-V Edition”, de Patterson & Hennessy ([HePa]).* )

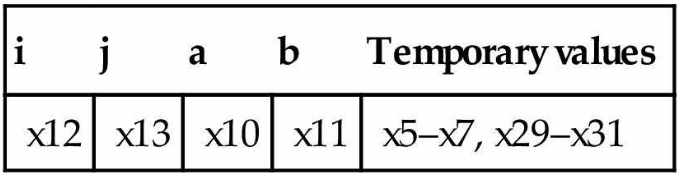
En este ejercicio, comparamos el rendimiento de los procesadores de uno y dos problemas, teniendo en cuenta las transformaciones del programa que se pueden realizar para optimizar la ejecución de dos problemas. Los problemas en este ejercicio se refieren al siguiente bucle (escrito en C):

for(i=0;i!=j;i+=2) b[i]=a[i]–a[i+1];

Un compilador que realiza poca o ninguna optimización podría producir el siguiente código ensamblador RISC-V:

li x12, 0 li x13, 8000 li x14, 0 SUPERIOR: slli x5, x12, 2 agregar x6, x10, x5 lw x7, 0(x6) lw x29, 4(x6) sub x30, x7, x29 agregar x31, x11 , x5 sw x30, 0(x31) addi x12, x12, 2 ENT: bne x12, x13, ARRIBA

Este código utiliza los siguientes registros:



Este código se proporciona en *[RVfpgaPath]/ RVfpga/Labs/Lab17/PaHe\_SuperscalarExample* con algunas modificaciones menores en comparación con el código proporcionado por el ejercicio del libro que no afecta el comportamiento del programa:

* registro x13 se inicializa a 8000, por lo que el bucle realizará 4000 iteraciones.
* Se elimina la instrucción jal .
* Las instrucciones ld y sd se sustituyen por las instrucciones lw y sw . Esto implica cambiar los accesos de 4 a 8 bytes de ancho.

Supongamos un procesador planificado estáticamente de dos problemas que tiene las siguientes propiedades:

1. Una instrucción debe ser una operación de memoria; el otro debe ser una instrucción aritmética/lógica o una rama.

2. El procesador tiene todas las rutas de reenvío posibles entre etapas.

3. El procesador tiene una predicción de bifurcación perfecta.

4. Dos instrucciones no pueden emitirse juntas si una depende de la otra.

5. Si es necesario entrar en pérdida, ambas instrucciones en una etapa deben entrar en pérdida.

1. Compare las propiedades de este procesador de ejemplo y las propiedades del procesador SweRV EH1.
2. Dibuje un diagrama de canalización y una simulación que muestre cómo se ejecuta una iteración aleatoria del bucle (excepto el primero) del código RISC-V anterior en el procesador SweRV EH1 de doble problema. Suponga que el ciclo sale después de cuatro mil iteraciones (este es el caso en el código anterior).
3. ¿Cuál es la aceleración de pasar de un procesador SweRV EH1 de un solo problema a uno de dos problemas? Explique los resultados. Pruebe el programa en la placa y habilite/desactive la ejecución de dos problemas.
4. Reorganice/reescriba el código RISC-V anterior para lograr un mejor rendimiento en el procesador SweRV EH1 de dos problemas. (Sin embargo, no desenrolle el lazo).
5. Ahora, desenrolle el código RISC-V para que cada iteración del bucle desenrollado maneje dos iteraciones del bucle original. Luego, reorganice/reescriba su código desenrollado para lograr un mejor rendimiento en el procesador SweRV EH1 de dos problemas.
6. **(OPCIONAL)** ( *El siguiente ejercicio se basa en los ejercicios 7.30, 7.32 y 7.34 del Capítulo 7 de DDCARV* ) .

Supongamos que el procesador SweRV EH1 está ejecutando el siguiente fragmento de código. Recordemos que SweRV EH1 cuenta con una Unidad de Peligrosidad. Puede suponer un sistema de memoria que devuelve el resultado dentro de un ciclo (para ese propósito usamos el DCCM, e insertamos el fragmento de código en un bucle y evitamos la primera iteración para que no haya errores de I$).

addi s1, t0, 11 # t0 contiene la dirección base del DCCM lw s2, 25(s1) lw s5, 16(s2) add s3, s2, s5 o s4, s3, t4 y s2, s3, s4

1. Simule el programa con Verilator y GTKWave. Analice los resultados y para cada ciclo, especifique:

\* ¿Qué instrucciones se decodifican, se emiten para su ejecución y se comprometen?

\* ¿Qué registros se están escribiendo y cuáles se están leyendo?

\* ¿Qué reenvío y paradas se producen?

1. ¿Cuál es el CPI del procesador en este programa? Primero responda teóricamente y luego confirme su respuesta ejecutando el programa en el tablero.
2. Realice el mismo análisis en el procesador de un solo problema y compare los resultados con los resultados del procesador de dos problemas.

Se proporciona un proyecto de PlatformIO en: *[RVfpgaPath]/ RVfpga/Labs/Lab17/DDCARV\_Exercises-30-32-34* . El programa bajo análisis se inserta en un bucle para que sea más fácil de entender en la simulación (cualquier iteración, excepto la primera, se puede usar para el análisis) y se puede medir usando contadores de rendimiento.

1. **(OPCIONAL)** ( *El siguiente ejercicio se basa en los ejercicios 7.31, 7.33 y 7.35 del Capítulo 7 de DDCARV* ) .

Repita el ejercicio 7 para el siguiente fragmento de código.

addi s1, t0, 52 addi s0, s1, -4 lw s3, 16(s0) sw s3, 20(s0) xor s2, s0, s3 o s2, s2, s3

Se proporciona un proyecto de PlatformIO en: *[RVfpgaPath]/ RVfpga/Labs/Lab17/DDCARV\_Exercises-31-33-35* .