

**EL PROGRAMA UNIVERSITARIO DE LA IMAGINACIÓN**

**Laboratorio RVfpga 16**

## **Peligros de control: Instrucciones para ramificación**

# INTRODUCCIÓN

En este laboratorio, completamos nuestro análisis de peligros. En los últimos dos laboratorios, estudiamos los riesgos *estructurales* y *de datos* en el procesador SweRV EH1, y ahora nos enfocamos en ***los riesgos de control*** . Como explicaron S. Harris y D. Harris en " *Diseño digital y arquitectura informática: edición RISC-V* " (que llamamos DDCARV) , se produce un *riesgo de control* cuando la decisión de qué instrucción buscar a continuación no se ha tomado en el momento hay que buscarlo.

**NOTA:** Antes de analizar la lógica de riesgo de control de SweRV EH1, recomendamos leer cómo se ejecutan las instrucciones de beq y cómo se resuelven los riesgos de control en el procesador canalizado que se describe en la Sección 7.5 de DDCARV. Específicamente, los riesgos de control se discuten en la Sección 7.5.3 . También recomendamos leer la Sección 7.7.3 sobre Predicción de bifurcación antes de completar la Sección 3 de esta práctica de laboratorio.

Los peligros de control son causados por instrucciones de bifurcación y salto, porque estas instrucciones deben calcular qué instrucción buscar a continuación. Y, para las instrucciones de bifurcación, también deben calcular si la bifurcación se toma o no. Por el contrario, para todas las demás instrucciones, la siguiente instrucción para buscar está en PC + 4.

En algunos procesadores, los riesgos de control nunca ocurren. Por ejemplo, los riesgos de control no ocurren en los procesadores donde una instrucción determinada se ejecuta completamente antes de que se obtenga la siguiente instrucción. Esto es cierto tanto para los procesadores de uno o varios ciclos en DDCARV. Específicamente, debido a que una instrucción de bifurcación se ejecuta completamente, las decisiones sobre si se toma la bifurcación y qué instrucción buscar a continuación se resuelven antes de que se obtenga la siguiente instrucción. Por el contrario, los procesadores de tubería obtienen la siguiente instrucción antes de que se resuelvan esas decisiones.

Un mecanismo para hacer frente a los riesgos de control es detener la tubería hasta que se haya tomado la decisión de qué instrucción buscar después de la bifurcación. Debido a que esta decisión se toma en la etapa EX1 en SweRV EH1 (como veremos en la Sección 2), la tubería tendría que detenerse durante cuatro ciclos en cada ramal (consulte la Figura 1 en el Laboratorio 11, que muestra la tubería). Esto degradaría gravemente el rendimiento del sistema si se producen bifurcaciones con frecuencia, que suele ser el caso en los programas reales, por lo que esta solución no está implementada en SweRV EH1.

Una alternativa es predecir si se tomará la bifurcación o no y comenzar a obtener instrucciones de la ruta predicha. Una vez que la decisión de la bifurcación está disponible, el procesador puede vaciar las instrucciones obtenidas si la predicción fue incorrecta (en cuyo caso se debe pagar una multa por predicción errónea de la bifurcación), o puede continuar con la ejecución de las instrucciones obtenidas cuando la predicción fue correcta (en cuyo caso no hay pérdida de rendimiento). En SweRV EH1 hay disponibles dos predictores de rama (BP), que analizamos en este laboratorio: un **predictor de rama ingenuo** , que siempre predice ramas como no tomadas y, por lo tanto, ofrece un rendimiento deficiente sin costo de hardware, y un **predictor de rama Gshare** , que ofrece mayor rendimiento a costa de hardware adicional.

En la Sección 2, describimos la ejecución de una instrucción beq en SweRV y luego realizamos algunas simulaciones de ejemplo utilizando el BP ingenuo (este es el escenario típico asumido en libros de texto como *DDCARV* ). Luego, en la Sección 3, explicamos cómo los riesgos de control se pueden manejar de manera más eficiente utilizando el Gshare Branch Predictor que implementa SweRV EH1.

# EJECUCIÓN DE LA INSTRUCCIÓN beq Y CÁLCULO POR PC

En esta sección analizamos la ejecución de una instrucción beq en SweRV EH1. Primero, en la Sección 2.A, explicamos cómo se ejecutan las instrucciones beq en la etapa EX1 y cómo se calculan la Dirección de búsqueda y la Dirección de búsqueda siguiente en la etapa FC1 (esto completa la explicación de la etapa FC1 que comenzamos en la Sección 2 .Bi de Lab 11) . Aunque la figura incluida ( Figura 1 ) y la mayoría de las descripciones son válidas para cualquier instrucción, nos enfocamos en la ejecución de una instrucción beq en una configuración de procesador que usa el BP ingenuo donde las bifurcaciones siempre se predicen como *no tomadas* (como se hace en DDCARV o en PaHe). Luego, en la Sección 2.B, realizamos algunos experimentos para ejemplificar estos conceptos. Una vez más, para estos experimentos, desactivamos el uso del predictor de rama y, en su lugar, usamos una predicción *no tomada* para todas las ramas condicionales (es decir, lo que hemos llamado BP ingenuo ).

1. **explicación teórica**

La Figura 1 muestra las estructuras principales en la etapa FC1 que se utilizan para determinar la **Dirección de captura** (que es el valor en el Contador de programa (PC), definido en DDCARV como un registro que contiene la dirección de memoria de la instrucción actual) y la **Siguiente Fetch Address** (que es el valor utilizado para actualizar la PC al final de cada ciclo). La figura también muestra las estructuras necesarias para ejecutar una instrucción beq en la etapa EX1 (la mayor parte del hardware que se muestra también se usa en la ejecución de otras instrucciones de bifurcación). Como en otros laboratorios, los nombres de las señales que se usan en la figura son los nombres reales que se usan en los módulos Verilog del procesador SweRV EH1.

1. **Obtener cómputo de dirección**

Como se muestra en la Figura 1 , la etapa FC1 incluye dos multiplexores: un multiplexor 2:1 que produce la dirección de captura en ifc\_fetch\_addr\_f1[31:1] y un multiplexor 5:1 que calcula la siguiente dirección de captura y la coloca en la señal fetch\_addr\_bf[ 31:1] .

* **Multiplexor 2:1** : produce la señal ifc\_fetch\_address\_f1[31:1] , la dirección de memoria de la instrucción obtenida en el ciclo actual que, como analizamos en la Figura 3 de la práctica de laboratorio 11, se proporciona al controlador de memoria para leer los 128 bits. paquete de instrucciones de la caché de instrucciones. Las dos entradas a este multiplexor son:
  + La dirección de destino de la rama ( exu\_flush\_path\_final[31:1] ) calculada en la etapa EX1, como analizaremos a continuación.
  + La siguiente dirección de búsqueda ( ifc\_fetch\_addr\_f1\_raw[31:1] ), calculada y registrada en el ciclo anterior como la salida del multiplexor 5:1 incluida en esta etapa y analizada a continuación ( fetch\_addr\_bf[31:1] ).

La señal de control de este multiplexor se llama exu\_flush\_final y se proporciona desde la etapa Execute. Si la recuperación debe ocurrir desde la dirección de destino de la bifurcación, exu\_flush\_final = 1 y exu\_flush\_path\_final[31:1] se usa como la dirección de recuperación; de lo contrario , exu\_flush\_final = 0 y ifc\_fetch\_addr\_f1\_raw[31:1] se utiliza como dirección de obtención.

Tenga en cuenta que en los procesadores explicados en DDCARV se utiliza un multiplexor 2:1 análogo para actualizar la PC en cada ciclo.

* **Multiplexor 5:1** : produce la señal fetch\_addr\_bf[31:1] , la dirección proviene de una de las siguientes cinco fuentes:
  + La dirección de búsqueda ( ifc\_fetch\_addr\_f1 ), que se usa en algunos casos cuando la PC permanece igual de un ciclo al siguiente.
  + La siguiente dirección secuencial ( fetch\_addr\_next ), que se calcula como la dirección de obtención ( ifc\_fetch\_addr\_f1 ) + 16, y que apunta al siguiente paquete de 128 bits.
  + La dirección pronosticada por el búfer de destino de rama ( ifc\_bp\_btb\_target\_f2 ), que es una de las estructuras principales del predictor de rama, y que se utiliza como dirección de obtención cuando se predice que se tomará una rama.
  + Dos señales de entrada más ( miss\_addr y exu\_flush\_path\_final ) que corresponden al *miss path* y al *flush path* respectivamente, pero que no analizamos en este laboratorio.

La señal proporcionada por este multiplexor ( fetch\_addr\_bf[31:1] ) se registra y se utiliza en el siguiente ciclo como entrada al multiplexor 2:1 analizado anteriormente.

Tenga en cuenta que este multiplexor 5:1 no existe en los procesadores de DDCARV, que tienen diseños más simples.

1. **Ejecución de la Instrucción beq**

Una bifurcación condicional debe calcular la dirección de destino de la bifurcación y probar si se cumple la condición. Específicamente, en el caso de SweRV EH1 (ver Figura 1 ) :

* **de destino de la bifurcación** : se utiliza un nuevo sumador en EX1 para calcular la dirección de destino de la bifurcación y colocarla en la señal flush\_path[31:1] . Esta señal se proporciona como entrada al multiplexor 2:1 en FC1 ( exu\_flush\_path\_final[31:1] ) a través de alguna lógica y registros.
* **Resolución de condiciones** : se utiliza un nuevo módulo en EX1, dentro del módulo **exu\_alu\_ctl** , para verificar si los dos operandos son iguales ( eq = 1) o no ( eq = 0). Con base en la señal eq (y algunas otras señales, como ap.beq , que analizará en una tarea propuesta), las señales flush\_upper y exu\_flush\_final se calculan y se envían a la etapa FC1, donde esta última se usa como señal de control de el multiplexor 2:1. Esta señal de control ( exu\_flush\_final ) es 1 cuando la bifurcación se pronosticó incorrectamente y 0 en caso contrario.

Específicamente, en el caso de una instrucción beq y suponiendo el uso del BP ingenuo explicado anteriormente donde todas las ramas se predicen como no tomadas, si los dos operandos de la rama no son iguales, entonces la rama no debe tomarse y la predicción es correcta. : exu\_flush\_final = flush\_upper = eq = 0. En este caso, el procesador puede continuar obteniendo y ejecutando instrucciones secuencialmente y no hay pérdida de rendimiento. Analizaremos esta situación en la Sección 2.Bi

En cambio, si los dos operandos son iguales, se debe tomar la bifurcación y, en el caso del BP ingenuo que predice no tomado, se produjo un error de predicción: exu\_flush\_final = flush\_upper = eq = 1. En este caso, como explicaremos en Sección 2.B.ii, las siguientes acciones se activan en la canalización SweRV EH1 (consulte la Figura 1 ).

* Cuando exu\_flush\_final = 1, la búsqueda de instrucción se redirige a la dirección de destino de la rama, seleccionando la entrada 1 del multiplexor 2:1 en FC1 ( ifc\_fetch\_addr\_f1[31:1] = exu\_flush\_path\_final[31:1] ), que contiene la rama dirección de destino calculada en la etapa EX1 como se explicó anteriormente.
* Las etapas de canalización que preceden a EX1 se vacían. Para ello, se proporcionan varias señales ( exu\_flush\_final , exu\_flush\_upper\_e2 , exu\_i0\_flush\_final y exu\_i1\_flush\_final ) a etapas anteriores (el uso de estas señales no se especifica en la Figura 1 ).

**TAREA (OPCIONAL) :** Examine los elementos del procesador incluidos en la Figura 1 en el código Verilog y explique cómo funcionan.

* Los elementos que se muestran en la etapa Decode (Archivo de Registro, Registro de Instrucciones y Unidad de Control) se encuentran en los módulos **dec** , **dec\_decode\_ctl** y **dec\_gpr\_ctl** .
* Los elementos que se muestran en la etapa EX1 se pueden encontrar en los módulos **exu** y **exu\_alu\_ctl** .
* Los elementos que se muestran en la etapa FC1 se pueden encontrar en los módulos **ifu** e **ifu\_ifc\_ctl** .

**TAREA (OPCIONAL):** Explicar cómo la señal flush\_upper se genera en el módulo **exu\_alu\_ctl** a partir de la señal eq , las señales de control ap.beq , ap.predict\_t y ap.predict\_nt , y algunas otras señales.

**TAREA (OPCIONAL):** Analizar en código Verilog el efecto de las señales exu\_flush\_final , exu\_flush\_upper\_e2 , exu\_i0\_flush\_final y exu\_i1\_flush\_final en EX1 y en las etapas que la preceden: FC1, FC2, Align y Decode. Para este análisis, puede ser útil usar las simulaciones de la Sección 2.B, donde puede incluir las señales que necesita.



Figura 1 . Vista de alto nivel de la instrucción beq ejecutándose a través de SweRV EH1

1. **Experimentos**

Ahora que hemos descrito los conceptos principales en la ejecución de una instrucción beq en EX1 y el cálculo de la Dirección de búsqueda y la Dirección de búsqueda siguiente en FC1, ahora mostramos algunas simulaciones para solidificar estos conceptos.

A lo largo de esta sección trabajamos con el ejemplo que se muestra en la Figura 2 , que ejecuta un bucle que se repite para iteraciones 0xFFFF (es decir, 65.535 en decimal) y que contiene dos instrucciones beq : la primera beq siempre *no se tomará* (excepto en la última iteración de el bucle) y siempre se *tomará la segunda* . Como es habitual, las instrucciones que queremos analizar (en este caso las instrucciones beq , resaltadas en rojo) están rodeadas de varios nops con el fin de aislarlas de las instrucciones anteriores y posteriores. La carpeta *[RVfpgaPath]/RVfpga/Labs /Lab16/BEQ\_Instruction* proporciona el proyecto PlatformIO que puede analizar, simular y modificar según lo desee.

|  |
| --- |
| Test\_Assembly:  li t2, 0x008 # Deshabilitar el predictor de bifurcación  csrrs t1, 0x7F9, t2  li t3, 0xFFFF li t4, 0x1 li t5, 0x0 li t6, 0x0  LOOP: add t5, t5, 1  INSERT\_NOPS\_7  **beq t3, t4, OUT** INSERT\_NOPS\_7  add t4, t4, 1  INSERT\_NOPS\_7  **beq t3, t3, LOOP** INSERT\_NOPS\_7  OUT: INSERT\_NOPS\_8  .end |

Figura 2 . Programa que incluye instrucciones beq

En nuestros experimentos deshabilitamos el uso de instrucciones comprimidas. Además, como mencionamos anteriormente, en esta sección, el Gshare Branch Predictor disponible en SweRV EH1 está deshabilitado y siempre se predice que las bifurcaciónes *no se tomarán* (naïve BP). Esto se hace incluyendo dos instrucciones que permiten al usuario configurar el procesador durante la ejecución. Como se describe en el Apéndice B del Laboratorio 11, debe incluir las siguientes dos instrucciones en su código para deshabilitar el Predictor de rama y, en su lugar, usar una predicción *no tomada para cada rama.*

li t2, 0x008

csrrs t1, 0x7F9, t2

En esta configuración, la primera rama del programa ( Figura 2 ) siempre será predicha correctamente (excepto en la última iteración del ciclo, que no analizaremos aquí) y la segunda rama siempre será mal predicha, lo que provocará un vaciado. de las cuatro etapas precedentes y un redireccionamiento de la ejecución. A continuación analizaremos la ejecución de las dos instrucciones beq .

1. **Ejecución de la primera rama: beq t3, t4, OUT**

En esta sección analizamos la ejecución de la primera instrucción de bifurcación de la Figura 2 , que siempre se predice correctamente (excepto en la última iteración del bucle, que no analizamos aquí). Abra el proyecto en PlatformIO, constrúyalo y abra el archivo de desensamblado (disponible en *[ RVfpgaPath ] /RVfpga/Labs/Lab16/BEQ\_Instruction/.pio/build/swervolf\_nexys/firmware.dis* ). Observe que la primera instrucción beq se coloca en la dirección 0x000001a8:

**0x000001a8: 07de0063 beqt3,t4,208 <SALIDA>**

A continuación, simulamos el programa de la Figura 2 en Verilator como se explica en el GSG y luego abrimos el archivo de seguimiento generado por el simulador en GTKWave. La figura 3 se acerca a una iteración aleatoria del ciclo (debe evitarse la primera iteración, ya que contiene fallas I$ que hacen que sea más difícil de analizar, así como la última iteración, que falla la predicción) y se enfoca en la ejecución de la primera instrucción beq .

La mayoría de las señales incluidas en la figura son las que mostramos en el diagrama de la Figura 1 . Sin embargo, debe tener en cuenta que aquellas señales que contienen direcciones de instrucciones (marcadas con un sufijo \_ext ) se han extendido para la simulación con 1 bit a la derecha igual a 0 en aras de la claridad (tenga en cuenta que las señales originales no extendidas en el código Verilog no incluye el bit menos significativo ya que siempre es 0); específicamente:

Código Verilog: exu\_flush\_path\_final[31:1]  Simulación: exu\_flush\_path\_final\_ext[31:0]

Código Verilog: ifc\_fetch\_addr\_f1\_raw[31:1]  Simulación: ifc\_fetch\_addr\_f1\_raw\_ext[31:0]

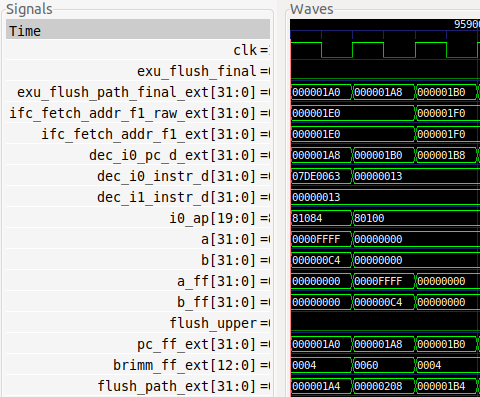
Código Verilog: ifc\_fetch\_addr\_f1[31:1]  Simulación: ifc\_fetch\_addr\_f1\_ext[31:0]

Código Verilog: pc\_ff[31:1]  Simulación: pc\_ff\_ext[31:0]

Código Verilog: brim\_ff[12:1]  Simulación: brim\_ff\_ext[12:0]

Código Verilog: flush\_path[31:1]  Simulación: flush\_path\_ext[31:0]

El archivo *test\_1.tcl* se proporciona con el proyecto. Para usarlo en GTKWave, haga clic en *Archivo → Leer archivo de script Tcl* y abra el *archivo [RVfpgaPath]/RVfpga/Labs /Lab13/BEQ\_Instruction/test\_1.tcl* . Luego, haga clic en *Acercar* ( ) varias veces y muévase a cualquier iteración del bucle, excepto la primera o la última. Verá la ejecución de las dos instrucciones beq ; La Figura 3 muestra lo que debe observar para la primera instrucción de bifurcación.



**DECODE**

**FC1**

**2-1 MUX**

**beq t3,t4,OUT**

**i+2**

**i+1**

**i**

**EX1**

Figura 3 . Simulación Verilator para la ejecución del primer beq de la Figura 2

Analice la forma de onda de la Figura 3 y el diagrama de la Figura 1 al mismo tiempo. La Figura 3 muestra tres ciclos consecutivos: Decodificación del beq (ciclo *i* ), EX1 del beq (ciclo *i+1* ), y selección de la siguiente PC en FC1 después de resolver el beq (ciclo *i+2* ).

* **Ciclo *i* - Etapa de decodificación para la instrucción beq** : la señal dec\_i0\_pc\_d\_ext contiene la dirección de la instrucción en la etapa de decodificación (en la Vía 0), que para la primera beq es 0x000001A8, y la señal dec\_i0\_instr\_d (generalmente llamada Registro de instrucción (IR) en los libros de texto ) contiene la instrucción máquina de 32 bits, que para la primera beq es 0x07DE0063 (en binario: 0000 0111 1101 1110 0000 0000 0110 0011).

En RISC-V, el código de operación para la instrucción beq es (consulte el Apéndice B de [DDCARV]):

imm 12,10:5 | rs2 | rs1 | 000 | im 4:1,11 | 1100011

para que pueda verificar que 0x07DE0063 corresponde a: beq t3,t4,OUT (imm 12:0 = 0x060) . Recuerde que el inmediato da el desplazamiento desde la PC actual de la dirección de destino. La dirección de destino (indicada por la etiqueta "OUT:") es de 24 instrucciones (es decir, 7 nops + 1 add + 7 nops + 1 beq + 7 nops + 1 nop = 24 instrucciones) más allá de la PC actual (es decir, beq t3,t4 ,FUERA ). Esto es 24\*4 = 96 (0x60) bytes más allá de la PC actual.

Durante esta etapa, el **pipeline** **se generan señales de control** . Para la primera instrucción beq , se establecen los siguientes bits de i0\_ap (que para esta instrucción es igual a 0x81084; consulte SweRVref.docx):

* + valid : indica que es una instrucción válida que utiliza la ALU.
  + beq : indica que es una *rama si es igual* a la instrucción.
  + sub : indica que la ALU debe realizar la resta. alguna rama

Las instrucciones usan el resultado de la resta para calcular la comparación ( sin embargo, este no es el caso para beq , como mostraremos).

* + predict\_nt : indica que la rama se predice como *no tomada* .

Además, **se lee el archivo de registro** y **la instrucción de bifurcación se enruta a la tubería I0** . Las señales a y b (0xFFFF y 0xC4, respectivamente, en este ejemplo) contienen las entradas al comparador utilizado en la siguiente etapa, que en este caso coinciden con los valores leídos del Archivo de Registro (en otros casos, los operandos podrían proporcionarse a través de reenvío, como se analiza en el laboratorio 15).

* **Ciclo *i+1* - Etapa EX1 para la instrucción beq** : En el ciclo siguiente se **ejecuta la** instrucción beq . Se comparan las señales a\_ff y b\_ff . Dado que los dos números (0xFFFF y 0xC4) son diferentes, no se toma la bifurcación. Como se describió anteriormente, en esta configuración todas las ramas se predicen y *no se toman* ( i0\_ap.predict\_nt = 1). Por lo tanto, la bifurcación no ha sido mal predicha ( flush\_upper = 0) y la ejecución puede continuar como está.
* **Ciclo *i+2* - Etapa FC1** : en el siguiente ciclo, dado que la bifurcación se predijo y resolvió como no tomada, la búsqueda simplemente continúa secuencialmente. En la Figura 3 , observe que exu\_flush\_final = 0 e ifc\_fetch\_addr\_f1\_ext[31:0] = ifc\_fetch\_addr\_f1\_raw\_ext[31:0] = 0x000001F0. Esta dirección apunta al siguiente paquete secuencial de instrucciones de 128 bits. Puede ver que en los dos ciclos anteriores se obtuvo el paquete de instrucciones de 128 bits anterior ( ifc\_fetch\_addr\_f1\_ext[31:0] = 0x000001E0).

**TAREA :** **(OPCIONAL)** Modificar Figura 1 para incluir los valores de cada señal que se muestra en la Figura 3 en los ciclos *i* , *i+1* e *i+2* .

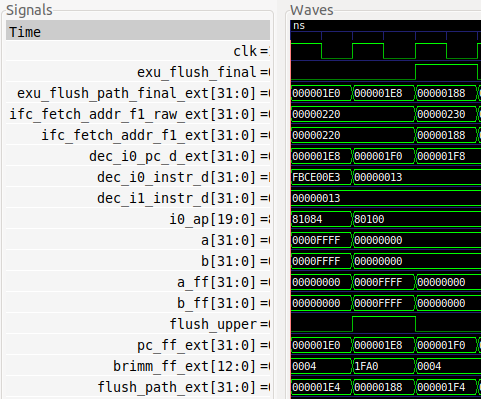
**TAREA :** Modifique el programa de la Figura 2 para hacer que la primera instrucción de bifurcación recupere sus operandos de entrada a través del reenvío (forwarding).

1. **Ejecución de la segunda rama: beq t3, t3, OUT (OPCIONAL)**

Ahora analizamos la segunda rama, que siempre se toma pero se predice erróneamente como si no se tomara. Abra el archivo de desmontaje (disponible en *[ RVfpgaPath] /RVfpga/Labs/Lab16/BEQ\_Instruction/.pio/build/swervolf\_nexys/firmware.dis )* . Observe que la segunda instrucción beq se coloca en la dirección 0x000001E8:

**0x000001e8: fbce00e3 beqt3,t3,188 <BUCLE>**

La Figura 4 muestra las señales durante una iteración aleatoria del ciclo (pero no la primera iteración, que evitamos debido a las fallas de la caché de instrucciones (I$)).



**DECODE**

**FC1**

**2-1 MUX**

**beq t3,t3,LOOP**

**i+2**

**i+1**

**i**

**EX1**

Figura 4 . Simulación de Verilator para la segunda rama en el ejemplo de la Figura 2

Analice la forma de onda de la Figura 4 y el diagrama de la Figura 1 al mismo tiempo. Los valores resaltados en rojo muestran tres ciclos consecutivos durante la ejecución de la segunda instrucción beq : decodificación de beq (ciclo *i* ), EX1 de beq (ciclo *i+1* ) y selección de la siguiente PC en FC1 después de resolver beq ( ciclo *i+2* ).

* **Ciclo *i* - Etapa de decodificación para la instrucción beq** : La PC (señal dec\_i0\_pc\_d\_ext ) es 0x000001E8, y la instrucción (señal dec\_i0\_instr\_d ) es 0xFBCE00E3 (en binario: 1111 1011 1100 1110 0000 0000 1110 0011).

En RISC-V, el código de operación para la instrucción beq es (consulte el Apéndice B de [DDCARV]):

imm 12,10:5 | rs2 | rs1 | 000 | im 4:1,11 | 1100011

Entonces puedes verificar que 0xFBCE00E3 corresponde a: beq t3,t3, LOOP (Inmediato 12:0 = 0x1FA0) . Recuerde que el inmediato da el desplazamiento desde la PC actual de la dirección de destino. La dirección de destino (indicada por la etiqueta "LOOP:") es de 24 instrucciones (es decir, 7 nops + 1 add + 7 nops + 1 beq + 7 nops + 1 add = 24 instrucciones) *antes* de la PC actual (es decir, beq t3,t3 , BUCLE ). Esto es 24\*4 = 96 bytes antes de la PC actual. Entonces, el inmediato codifica -96, que es 0x1FA0, escrito en representación de complemento a dos de 13 bits.

Durante esta etapa el **cauce** **se generan señales de control** . Para esta instrucción beq , las señales de control son las mismas que para la primera beq (consulte la sección anterior).

Además, **se lee el archivo de registro** y **la instrucción de bifurcación se enruta a la tubería I0** . Las señales ayb ( 0xFFFF para ambas) contienen las entradas al comparador utilizado en la siguiente etapa, que en este caso son los valores leídos del Archivo de Registro.

* **Ciclo *i+1* - Etapa EX1 para la instrucción beq** : En el ciclo siguiente se **ejecuta la** instrucción beq . Por un lado, se comparan las señales a\_ff y b\_ff . Dado que los dos valores son iguales, se debe tomar la rama. Sin embargo, como se explicó antes, en nuestra configuración todas las ramas se predicen y *no se toman* ( i0\_ap.predict\_nt = 1). Por lo tanto, la bifurcación ha sido mal predicha ( flush\_upper = 1). Por lo tanto, las instrucciones deben obtenerse de la dirección de destino de la bifurcación y las etapas iniciales de la canalización deben vaciarse.

En esta etapa, la dirección de destino se calcula como la suma de pc\_ff\_ext (0x1E8) y brim\_ff\_ext (0x1FA0). El resultado se coloca en la señal flush\_path\_ext (0x00000188).

* **Ciclo *i+2* - Etapa FC1** : En el siguiente ciclo, la ejecución debe continuar en la dirección de destino de la bifurcación. En la Figura 4 puede ver que exu\_flush\_final = 1 e ifc\_fetch\_addr\_f1\_ext = exu\_flush\_path\_final\_ext = 0x00000188. Esta dirección corresponde a la dirección de destino de la bifurcación, que es la dirección de la primera instrucción del bucle (tenga en cuenta que se trata de una bifurcación hacia atrás).

**TAREA** (OPCIONAL)**:** Modifique la Figura 1 para incluir los valores de cada señal que se muestra en la Figura 4 en los ciclos *i* , *i+1* e *i+2* .

**TAREA** (OPCIONAL) **:** Analice el funcionamiento de los dos multiplexores de FC1 con el ejemplo de la Figura 2 , examinando las señales en diferentes circunstancias.

Por ejemplo, analice cómo se logra la búsqueda para la ejecución secuencial (es decir, un grupo de instrucciones sin bifurcaciones). Verás que, en el procesador SweRV EH1, el funcionamiento en este caso es el siguiente:

- En los ciclos pares, fetch\_addr\_next se selecciona usando el multiplexor 5:1, que contiene la dirección de búsqueda actual ( ifc\_fetch\_addr\_f1 ) más 16, por lo que se lee el siguiente paquete secuencial de instrucciones de 128 bits (recuerde que una lectura I$ proporciona 128 bits ).

- En los ciclos impares, el ifc\_fetch\_addr\_f1 se selecciona mediante el multiplexor 5:1, por lo que no se obtienen nuevas instrucciones.

De esta forma, se obtienen cuatro instrucciones de 32 bits cada 2 ciclos, que es la misma tasa de instrucciones que necesita la etapa de decodificación (2 instrucciones por ciclo).

Tenga en cuenta que en los procesadores de DDCARV, la PC simplemente se incrementa en cuatro en cada ciclo (para ejecución secuencial) para obtener una instrucción por ciclo.

Modifique también el programa de la Figura 2 para crear nuevos escenarios. Por ejemplo, puede agregar algunas instrucciones AL después de la rama tomada y ver cómo se vacían después de la redirección.

**TAREA :** En el laboratorio 15, analizamos cómo se resuelven los riesgos de datos RAW en la etapa de compromiso por medio de las ALU secundarias. De manera similar a las instrucciones AL que estudiamos en ese laboratorio, una instrucción de bifurcación condicional puede tener un riesgo de datos RAW con una operación anterior de varios ciclos que debe resolverse en el momento de la confirmación. Si se determina que la bifurcación se pronosticó erróneamente, la canalización debe vaciarse y redirigirse desde la etapa de confirmación. Analice esta situación utilizando una versión ligeramente modificada del programa de la Figura 2 , proporcionada en *[RVfpgaPath]/RVfpga/Labs /Lab16/BEQ\_Instruction\_HazardCommit* , y el archivo *.tcl* proporcionado en esa misma carpeta.

# El Gshare Branch Predictor utilizado por SweRV EH1(OPCIONAL)

En la Sección 2, discutimos la configuración de SweRV EH1 que incluye solo un predictor de rama ingenuo que siempre predice que no se toma, pero en esta sección analizamos el funcionamiento del Gshare Branch Predictor disponible en SweRV EH1. Gshare BP realiza una predicción más inteligente para cada instrucción de bifurcación, lo que mejora el rendimiento pero requiere hardware adicional. Antes de describir cómo funciona Gshare BP en SweRV EH1, comparamos el rendimiento de los dos BP.

**TAREA :** En el ejemplo de la Figura 2 , elimine todas las instrucciones nop y analice la simulación. Luego calcule el IPC con los contadores de rendimiento ejecutando el programa en la placa.

Habilite el predictor de rama utilizado en SweRV EH1 (comentando las dos instrucciones iniciales en la Figura 2 ) y analice la simulación y la ejecución en el tablero.

Compare los dos experimentos y explique los resultados.

**NOTA:** Un artículo clásico publicado por Scott McFarling en 1993 se llama “Combining Branch Predictors” ( <https://www.hpl.hp.com/techreports/Compaq-DEC/WRL-TN-36.pdf>). Describe, en la Sección 7, el funcionamiento del predictor de rama Gshare. También puede buscar otros documentos, como <https://people.engr.ncsu.edu/efg/521/f02/common/lectures/notes/lec16.pdf>. Recomendamos leerlos para comprender cómo funciona Gshare BP antes de comenzar esta sección.

La Figura 5 muestra una vista simplificada del Gshare BP disponible en SweRV EH1. Todas las estructuras de BP se implementan dentro del módulo **ifu\_bp\_ctl** (en el archivo *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/ifu/ ifu\_bp\_ctl.sv* ). Las estructuras relacionadas con Gshare BP están rodeadas por un cuadrado azul en la figura.

Este BP está compuesto por la Branch History Table (BHT), que predice la dirección de la bifurcación ( *tomada* o *no tomada* ), y el Branch Target Buffer (BTB), que predice la dirección de destino en el caso de bifurcaciónes tomadas. En nuestra configuración predeterminada, el BHT contiene 128 entradas de 2 bits. Puede encontrarlo en las líneas 1615-1705 del módulo **ifu\_bp\_ctl** . En nuestra configuración predeterminada, el BTB contiene 32 entradas de 13 bits. Puede encontrarlo en las líneas 1439-1613 del módulo **ifu\_bp\_ctl** .

Para hacer una predicción de bifurcación, ocurre lo siguiente en cada ciclo (ver Figura 5 ):

1. La dirección de búsqueda ( ifc\_fetch\_addr\_f1 [31:1] ) y algunas otras señales se pasan a través de varios módulos hash dentro del módulo **ifu\_bp\_ctl** : f1hash , rdtagf1 , fghrhs ... Todos estos módulos hash se implementan en el archivo *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex /beh/beh\_lib.sv* , usando algunas de las macros definidas en *[RVfpgaPath]/RVfpga/src/SweRVolfSoC/SweRVEh1CoreComplex/include/common\_defines.vh* .

Como ejemplo, puede ver que el módulo fghrhs recibe la señal btb\_rd\_addr\_f1 , que proviene de un hashing de la dirección de obtención ( f1hash(.pc(ifc\_fetch\_addr\_f1[31:1]), .hash(btb\_rd\_addr\_f1[`RV\_BTB\_ADDR\_HI:`RV\_BTB\_ADDR\_LO] )) ) y fghr\_ns (que es el registro de historial global) y emite la señal bht\_rd\_addr\_hashed\_f1 .



Esta señal se utiliza para acceder a la tabla BHT del Gshare Branch Predictor.

**TAREA** : Analice todos estos módulos hash e intente hacerse una idea de cómo funcionan y cómo se utilizan en las estructuras Gshare BP.

1. Todas estas señales hash ( btb\_rd\_addr\_f1 , bht\_rd\_addr\_hashed\_f1 , fetch\_rd\_tag\_f1 , etc.) se utilizan para acceder a las dos estructuras principales que componen el Gshare BP: el BHT y el BTB.

**TAREA** : Analizar cómo se realiza el acceso a estas dos estructuras .

1. Como resultado del acceso al BHT, se obtiene una predicción de dirección en la señal ifu\_bp\_kill\_next\_f2 , que es 0 si se predice que *no se toma el ramal* y 1 si se predice que se *toma* . Esta señal se usa, además de otras señales que no describimos aquí, para calcular la señal de control del multiplexor 5:1 de FC1.

**TAREA** : Analizar cómo se calcula la señal seleccionada del multiplexor 5:1 .

1. Como resultado del acceso al BTB, la dirección de destino prevista para las bifurcaciones tomadas se obtiene de un sumador en la señal ifu\_bp\_btb\_target\_f2 [31:1] . (Tenga en cuenta que la dirección predicha también puede provenir de la pila de direcciones de retorno (RAS) en caso de que se prediga una instrucción ret ). Esta señal es una de las entradas del multiplexor 5:1 de FC1.

**TAREA** : analice cómo se obtiene la dirección de destino predicha ( ifu\_bp\_btb\_target\_f2 ) a partir del valor leído en BTB ( btb\_rd\_tgt\_f2[11:0]) y la dirección de obtención en FC2 ( ifc\_fetch\_addr\_f2[31:4] ) .

**TAREA** : Analizar el RAS implementado en el procesador SweRV EH1 . Una búsqueda en Internet también brindará información adicional sobre el funcionamiento de esta estructura (por ejemplo , <http://www-classes.usc.edu/engr/ee-s/457/EE457_Classnotes/ee457_Branch_Prediction/EE560_05_Ras_Just_FYI.pdf>).

1. En el multiplexor 5:1 de FC1, si ifu\_bp\_kill\_next\_f2 = 1, la dirección de destino pronosticada se usa como la siguiente dirección de obtención: fetch\_addr\_bf [31:1] = ifu\_bp\_btb\_target\_f2 [31:1] (a menos que se vacíe la canalización). En cambio, si ifu\_bp\_kill\_next\_f2 = 0, una de las otras cuatro entradas se usa como la dirección de búsqueda siguiente.



Figura 5 . Estructuras principales (rodeadas por un cuadrado azul) que componen el Gshare Branch Predictor disponible en SweRV EH1

A lo largo de esta sección , continuamos trabajando con el código de ejemplo de la Figura 2 . La única diferencia en esta sección es que habilitamos el Gshare Branch Predictor reemplazando las dos instrucciones que deshabilitan el Gshare BP con dos instrucciones nop (la razón para insertar dos nops es mantener las mismas direcciones de instrucción que en la sección anterior).

Ahora analizamos la ejecución de la segunda instrucción de bifurcación en el programa, como se hizo en la Sección 2.B.ii. Recuerde que la segunda instrucción beq está ubicada en la dirección 0x000001E8 en nuestro programa, lo que significa que está contenida dentro del paquete de 128 bits asignado en el rango de direcciones 0x1E0-0x1EF:

**0x000001e8: fbce00e3 beqt3,t3,188 <BUCLE>**

La Figura 6 hace zoom en una iteración aleatoria del ciclo. Como de costumbre, se evita la primera iteración, ya que contiene fallos de I$; además, la predicción de bifurcación falla para esta instrucción de bifurcación en su primera iteración. La mayoría de las señales incluidas en la figura son las que mostramos en la Figura 5 . El archivo *test\_1\_BP.tcl* se proporciona con el proyecto. Para usarlo en GTKWave, haga clic en *Archivo → Leer archivo de script Tcl* y abra el *archivo [RVfpgaPath]/RVfpga/Labs /Lab16/BEQ\_Instruction/test\_1\_BP.tcl* . Luego, haga clic en *Acercar* ( ) varias veces y muévase a cualquier iteración del bucle, excepto a la primera. Verá la ejecución de las dos instrucciones beq ; La Figura 6 muestra lo que debe observar para la segunda instrucción de bifurcación.

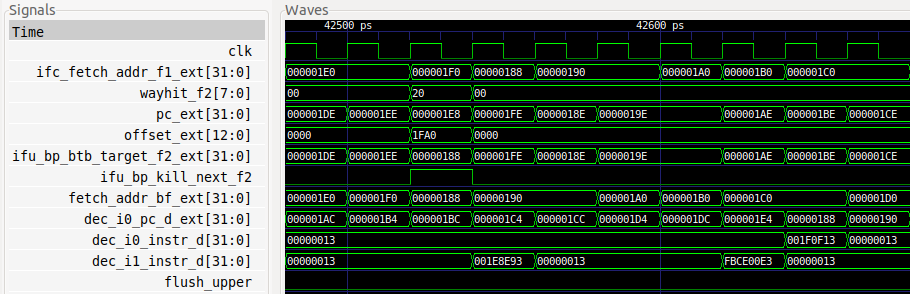
**i+8**

**i+7**

**i+9**

**i+3**

**i+2**



**i**

Figura 6 . Simulación de Verilator para el ejemplo de la Figura 2

Analice la forma de onda de la Figura 6 y el diagrama de la Figura 5 al mismo tiempo. Los valores resaltados en rojo corresponden a la segunda instrucción beq a medida que atraviesa las etapas de la canalización.

* **Ciclo *i*** : la dirección del paquete que contiene la segunda rama se proporciona a la caché de instrucciones: ifc\_fetch\_addr\_f1\_ext = 0x000001E0. El búfer de destino de rama (BTB) se lee utilizando esta dirección.
* **Ciclo *i+2*** : Se produce un acierto en el BTB: wayhit\_f2 = 0x20 (esta señal, que no está incluida en la Figura 5 , indica un acierto cuando es distinta de cero). La dirección de la bifurcación ( pc\_ext = 0x000001E8) se agrega al desplazamiento proporcionado por BTB ( offset\_ext = 0x1FA0, que es un valor negativo), lo que da como resultado la dirección de destino prevista ( ifu\_bp\_btb\_target\_f2\_ext = 0x00000188). Dado que el BHT predice que la rama será tomada ( ifu\_bp\_kill\_next\_f2 = 1), se utiliza como PC de búsqueda siguiente ( fetch\_addr\_bf\_ext = 0x00000188).
* **Ciclo *i+3*** : la dirección de obtención es la dirección de destino prevista de la bifurcación, que se calculó en el ciclo anterior: ifc\_fetch\_addr\_f1\_ext = 0x00000188.
* **Ciclo *i+7*** : La rama se decodifica en la Vía 1 ( dec\_i1\_instr\_d = 0xFBCE00E3).
* **Ciclo *i+8*** : La bifurcación se ejecuta. La predicción fue correcta, por lo que no es necesario activar la descarga ( flush\_upper = 0).
* **Ciclo *i+9*** : La ejecución continúa normalmente a través de la dirección de destino de la bifurcación dado que la predicción fue correcta.

**TAREA** : Explicar cómo se actualiza el Registro de Historial Global en el módulo **ifu\_bp\_ctl** .

# EJERCICIOS

1. Implemente un predictor de rama bimodal y compare su rendimiento con el Gshare BP. **(OPCIONAL)**
2. ( *El siguiente ejercicio se basa en el ejercicio 4.25 del libro “Computer Organization and Design – RISC-V Edition”, de Patterson & Hennessy ([HePa]).* )

Considere el siguiente bucle:

LOOP: lw x10, 0(x13) lw x11, 4(x13) add x12, x10, x11 add x13, x13, -8 bnez x12, LOOP Supongamos que se usa la predicción de bifurcación perfecta (en el caso de SweRV EH1, podemos emular este comportamiento simplemente evitando la primera iteración), que la tubería tiene soporte de reenvío completo (nuevamente, este es el caso en SweRV EH1) y que las ramas se resuelven en la etapa EX1.

* 1. Muestre una simulación para la segunda y tercera iteraciones de este bucle. Explique el comportamiento obtenido. Puede usar el programa provisto en *[RVfpgaPath]/RVfpga/Labs /Lab16/HePa\_Exercise-4-25* .