METODOLOGÍA PARA LA DETECCIÓN DE RIESGOS DE DATOS APLICADA A UN PROCESADOR RISC-V EN PIPELINE DE 5 ETAPAS

DATA HAZARD DETECTION METHODOLOGY FOR A RISC-V FIVE STAGE PIPELINE PROCESSOR

Edgar Alberto Maestro Ruiz
Universidad de Guadalajara, México
edgarmae_90@hotmail.com

Josué Vladimir Quiroga Esparza
CUCEI – Universidad de Guadalajara, México
josue.quiroga4742@alumnos.udg.mx

Marco Antonio Gurrola Navarro
Universidad de Guadalajara, México
marco.gurrola@academicos.udg.mx

Recepción: 28/octubre/2022 **Aceptación:** 14/marzo/2023

Resumen

Uno de los principales retos en el diseño de un microprocesador (uP) con microarquitectura segmentada o *pipeline*, es la detección de riesgos de datos, ya que, al dividir la ruta de datos o *datapath*, se generan posibles inconsistencias durante la ejecución secuencial de las instrucciones, que deben ser solucionadas vía software en tiempo de compilación o implementando hardware que solucione estos problemas en tiempo de ejecución. En este trabajo se plantea una metodología para la identificación de los riesgos de datos que pueden surgir al implementar el conjunto de instrucciones base de RISC-V con microarquitectura *pipeline* de 5 etapas. La validación se realiza mediante la comparación de los resultados obtenidos en pruebas aplicadas a un sistema de referencia disponible en la página de la organización RISC-V International y una implementación de un uP *pipeline* que incluye las técnicas *bypassing* y *stall insertion* para la solución de riesgos de datos.

Palabras Clave: adelantamiento de instrucciones, inserción de paro, riesgos de datos, RISC-V, segmentación

Abstract

One of the main challenges about pipeline processors development is data hazard detection, since, dividing data path, possible inconsistencies are generated during sequential execution of instructions, which must be solved via software during compilation time or implementing hardware that solves these problems during execution time. In this paper a methodology is proposed to detect all data hazard that could be generated implementing the base instruction set architecture of RISC-V in five stages pipeline segmentation. Validation is achieved by comparing the result of tests applied to a reference system available in RISC-V international website and an implementation of a pipeline processor that solves data hazard through stall insertions and bypassing technics.

Keywords: *data hazard, forwarding, pipelining, RISC-V, stall insertion*

1. Introducción

Antecedentes

El *pipelining* es una de las técnicas más utilizadas en los procesadores actuales, con la cual se obtiene un aumento de la frecuencia de operación del uP y un mejor aprovechamiento de los recursos de hardware.

No obstante, las implementaciones *pipeline* deben considerar los problemas que pueden surgir al dividir el proceso de ejecución de las instrucciones, uno de estos son los riesgos de datos (*data hazard*). La aparición de estas condiciones requiere un análisis detallado para su detección y solución ya sea a través de recursos de hardware o software que resuelvan estos problemas en tiempo de ejecución o en tiempo de compilación.

En el texto [1] se presenta un análisis de los riesgos de datos para una microarquitectura *pipeline* de 5 etapas, donde se considera una pequeña porción de las instrucciones del ISA base de RISC-V. De ahí que el diseño completo de una microarquitectura *pipeline* de este tipo no puede ser logrado sólo con la información de esta fuente. En [2] se presenta un método para la solución y detección de riesgos de datos a través de un acuerdo entre software y hardware, por lo que algunas dependencias son resultas en tiempo de compilación y otras en tiempo de ejecución.

En el trabajo [3] se presenta el diseño de una unidad de adelantamiento para un uP *pipeline* de 5 etapas con arquitectura MIPS, que implementa todas las instrucciones básicas del ISA, sin embargo, existen algunas diferencias entre el set de instrucciones de MIPS y RISC-V que no permiten tener un panorama completo para la detección de todos los riesgos de datos incluidos en esta última arquitectura. En [4] presentan un método llamado *supply-matching* que facilita la detección de riesgos de datos usando una microarquitectura basada en la ISA PowerPC. El método determina cuándo las dependencias pueden ser solucionadas mediante *forwarding* o si debe ser corregida mediante le técnica de *stall insertion*. En [5] se presenta un análisis de los riesgos de datos mediante la categorización de instrucciones en una arquitectura *pipeline* RISC de 5 etapas, se listan 8 diferentes casos en donde se incluyen instrucciones de tipo aritméticas, lógicas, carga y bifurcación.

La motivación del presente trabajo es obtener un método sencillo y práctico para la detección de los riesgos de datos que pueden ser incluidos en una microarquitectura en *pipeline* de 5 etapas que implementa un conjunto de instrucciones base de RISC-V incluyendo arquitectura del modo de privilegio M, de tal manera que el análisis no sea solo enfocado a los riesgos de datos relacionados con los *general purpose registers* (GPR) si no, con todos los elementos de hardware que puedan ocasionarlos, tales como los CSR (*control and status registers*) incluidos en el ISA de RISC-V.

Arquitectura y microarquitectura

Un uP está determinado por 2 aspectos: su arquitectura y su microarquitectura. La arquitectura son todas las características que definen a un uP de manera funcional, estas son: su conjunto de instrucciones, sus registros disponibles, modos de direccionamiento, etc. De esta forma cualquier tarea realizada por diferentes microprocesadores con la misma arquitectura debe generar el mismo resultado.

Por otro lado, la microarquitectura define al uP de manera física, esto es la forma en que sus componentes de hardware están interconectados para realizar las funciones determinadas por la arquitectura, así, procesadores con diferentes

microarquitecturas pueden diferir en varios aspectos tales como sus recursos de hardware, rendimiento, consumo de potencia, etc.

A continuación, se describirán de manera breve estos 2 aspectos del uP, arquitectura y microarquitectura, que se ha implementado en este proyecto. Cabe aclarar que solo se han incluido en la descripción las características básicas que permiten el análisis de los riesgos de datos, el cual es el objetivo de este trabajo.

Arquitectura seleccionada

El conjunto de instrucciones de la arquitectura del uP implementado incluye todas las instrucciones del ISA base RV32I, excluyendo de este solo la instrucción FENCE, ya que esta no implica ninguna funcionalidad en un sistema con un solo hilo de hardware (*hart*), como el que se presenta en este trabajo, además se han incluido las 6 instrucciones de la extensión Zicsr y 2 instrucciones privilegiadas: MRET y WFI, dichas instrucciones permiten la implementación del modo de privilegio M, siendo este el modo el único requerido para cualquier implementación de RISC-V. La descripción de todas estas instrucciones se puede encontrar en la especificación de RISC-V en [6].

Propuesta de microarquitectura

La figura 1 muestra el diagrama simplificado de la microarquitectura propuesta por los autores de este trabajo para un uP RISC-V segmentado en 5 etapas [2]:

- Obtención de la instrucción (IF).
- Decodificación (ID).
- Ejecución (EXE).
- Acceso a memoria (MEM).
- Escritura a registros (WB).

En la etapa IF se integra una memoria de instrucciones (IMEM), el registro contador de programa (PC), un multiplexor 4 a 1 (PC_MUX) y un sumador de 32 bits (ADD_PC). Mediante estos elementos se realiza la extracción de las instrucciones desde IMEM, cuya dirección es establecida por el registro PC, la entrada de este

registro es seleccionada a través de PC_MUX, la cual puede ser cualquiera de los siguientes datos: a) la dirección de la siguiente instrucción (calculada por el sumador ADD_PC), b) una dirección de salto proveniente de instrucciones B y J, c) la dirección del vector de interrupciones previamente configurada en el registro MTVEC y d) la dirección de retorno de excepciones e interrupciones (que debe estar almacenada en el registro MEPC).

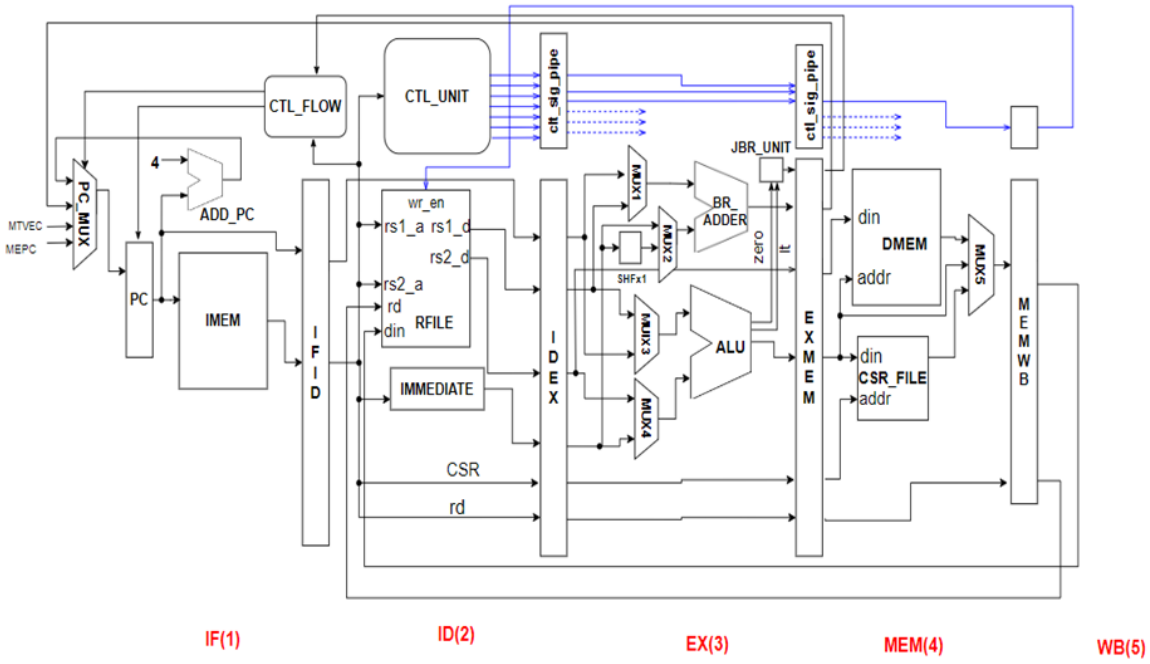

Figura 1 Microarquitectura propuesta para un uP RISC-V segmentada en 5 etapas.

La etapa ID se constituye por un banco de 32 registros (RFILE), que cumple con las especificaciones de [6], una unidad de reconstrucción de dato inmediato (IMMEDIATE) y la unidad de control, compuesta a su vez por los módulos (CTL_FLOW y CTL_UNIT). Por medio de estos elementos se realiza la decodificación de la instrucción y se obtienen los datos que estas requieren para su procesamiento. El módulo RFILE entrega los operandos establecidos por las instrucciones como "**rs1"** y "**rs2"** en flanco de bajada mientras las escrituras son realizadas en flanco de subida; la unidad IMMEDIATE decodifica los campos "**imm**", "**zimm**", "**offset**" y "**shamt**" incluidos en las instrucciones, para convertirlo en un dato de 32 bits que pueda ser utilizado por las unidades en la etapa EX. Por su parte la

unidad CTL_UNIT decodifica la instrucción para establecer las señales de control que requieren todas las unidades y multiplexores de la ruta de datos, mientras la unidad CTL_FLOW evalúa las condiciones de control de flujo para realizar paros o saltos en el pipeline cuando sea necesario

En la etapa EX se integra la *Unidad Aritmética Lógica* (ALU), un sumador para el cálculo de direcciones de salto (BR_ADDER), una unidad para evaluación de condiciones de salto (JBR_UNIT) y 4 multiplexores que eligen entre las diferentes posibles entradas para los módulos ALU y BR_ADDER. Por medio de la ALU es posible obtener el dato que debe ser cargado al registro de RFILE establecido por el campo "**rd**" en instrucciones tipo U, I R y J; también por medio de esta unidad se obtienen la dirección de memoria para instrucciones tipo L y S, así como los resultados de las comparaciones requeridas en instrucciones tipo B, los cuales son reflejados en las banderas "**zero**" y "**lt**" incluidas en su interfaz. Por su parte el sumador BR_ADDER permite la obtención de las direcciones de salto en instrucciones tipo J y B, mientras la unidad JBR_UNIT establece una bandera de condición de salto a través de la evaluación de las señales "**zero**" y "**lt**" producidas por la ALU y algunas señales de control que le indican el tipo de instrucción que se está ejecutando.

La etapa de acceso a memoria incluye además de la memoria de datos (DMEM), un banco de registros (CSR_FILE) que integra los CSR incluidos en sistema y un multiplexor de selección de salida (MUX5). DMEM es utilizada para el almacenamiento de datos en instrucciones S y para lectura de datos en instrucciones L, esta es implementada como una memoria provisional que puede ser reemplazada por algún tipo de memoria especifico, por ejemplo: RAM o cache. Por su parte el módulo CSR_FILE contiene 17 CSR, estos son: MSTATUS, MTVEC, MEPC, MCAUSE, MTVAL, MIE, MIP, MSCRATCH, MISA, MINSRET, MCYCLE, MCOUNTINHIBIT y 5 registros de protección de memoria. Todos estos registros son implementados de acuerdo con las especificaciones de RISC-V [7], incluyendo solo los campos requeridos para un sistema con solo el modo de privilegio M; la lecturas y escrituras a estos registros se realiza de manera similar a RFILE, donde las lecturas se ejecutan en flanco de subida y las escrituras en flaco de bajada, cabe

aclarar que las operaciones que implican las instrucciones tipo CSRR y CSRI se realizan mediante un hardware integrado en CSR_FILE. Por su parte MUX5 selecciona el dato que debe escribirse en el registro destino establecido por el campo "**rd**", cuyas opciones pueden provenir de la ALU, la memoria de datos o el CSR_FILE. Por último, la etapa WB está compuesta solo por el hardware de escritura de RFILE, se puede observar que la dirección de registro destino "**rd**" y la señal de escritura al RFILE recorren por el pipeline desde la etapa ID hasta la etapa WB donde estas son utilizadas junto con el dato obtenido por multiplexor MUX5.

Pipelining **(segmentación)**

La segmentación o *pipelining* es una técnica que se desarrolló con el objetivo de mejorar el rendimiento de los microprocesadores, estableciendo periodos de reloj más cortos y un mejor aprovechamiento de los recursos de hardware. Esto es logrado a través de registros que dividen la ruta de datos en segmentos más cortos. La microarquitectura planteada en la figura 1 presenta una segmentación de 5 etapas, comúnmente utilizada en sistemas RISC simples. La longitud de estos registros está definida por la cantidad de bits de los datos que son propagados de una etapa a otra. En la figura 2 se puede observar una representación del recorrido de las instrucciones a través del pipeline, donde la etapa liberada por una instrucción es ocupadas por la instrucción subsiguiente, de esta manera, idealmente una vez que el pipeline se llene, se obtiene el resultado de una instrucción en cada ciclo de reloj, sin embargo, al dividir la ruta de datos en segmentos es posible ajustar el periodo de reloj mínimo del uP al retardo del segmento más largo en el pipeline, reduciendo así el tiempo de ejecución por instrucción.

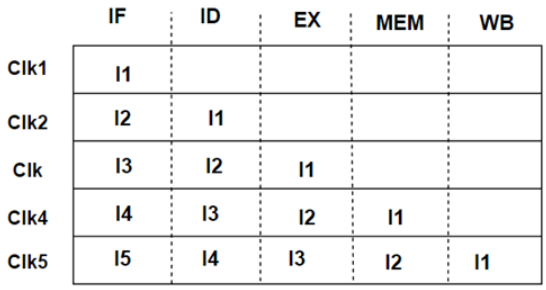

Figura 2 Ejecución de instrucciones en un *pipeline* de 5 etapas.

No obstante, esta mejora es limitada a causa de las condiciones de riesgos que son ocasionadas al dividir el ciclo de ejecución de las instrucciones, los cuales son descritos a continuación.

Riesgos en las implementaciones segmentadas

- *Riesgos estructurales:* Este tipo de riesgos se presentan cuando una o varias instrucciones requieren un determinado elemento de hardware al mismo tiempo. Un ejemplo de esto pudiera ocurrir durante la ejecución de las instrucciones tipo B si en la etapa de ejecución solo se contara con un sumador, ya que estas requieren de 2 operaciones de suma para poder llevar a cabo su ejecución
- *Riesgos por control:* Este tipo de riesgos son ocasionados al interrumpir el flujo secuencial de las instrucciones al ejecutar saltos, bifurcaciones, excepciones o interrupciones, ya que cuando estas ocurren algunas instrucciones deben ser invalidas, debido a que estos saltos se definen en las etapas ID o MEM, de esta manera las instrucciones que entraron al pipeline después de la condición que ocasionó el salto no deben realizar su ejecución, puesto que estas ya no son parte del segmento de código que se requiere procesar.

  El análisis para detección y solución de este tipo de riesgos no está contemplado en este trabajo.
- *Riesgos de datos:* Los riesgos de datos son ocasionados en un uP pipeline a causa de las dependencias entre instrucciones, las cuales pueden causar errores al momento de operar con datos de registros o direcciones de memoria que no han sido actualizados debido a la ejecución segmentada de las instrucciones. Estas dependencias pueden ser categorizadas en 4 tipos (Figura 3):
  - ✓ Escribir después de escribir (WAW).
  - ✓ Escribir después de leer (WAR).
  - ✓ Leer después de leer (RAR).
  - ✓ Leer después de escribir (RAW).

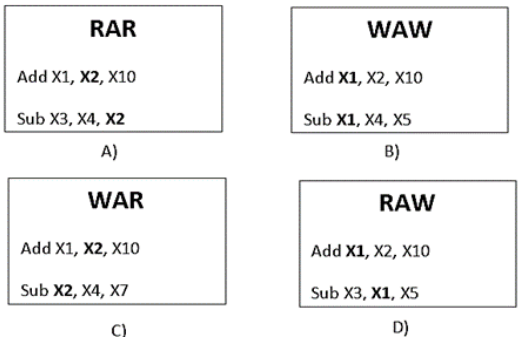

Figura 3 Tipos de dependencias de datos.

Un riesgo de datos ocurre cuando una de las instrucciones implicadas con una condición de dependencia utiliza el dato que otra genera en el transcurso de su ejecución, por lo tanto, deben de existir operaciones tanto de lectura como de escritura realizadas por las instrucciones. De esta forma las dependencias RAR y WAW no ocasionan riesgos de datos, ya que en ellas solo se implican un tipo de operación relacionadas con un dato especifico. De esta forma solo las dependencias RAW y WAR son objeto de análisis.

Las dependencias RAW se refieren a casos donde la primera instrucción modifica (escribe) el dato que la segunda instrucción requerirá (lee) para operar, lo cual coincide con la condición de riesgo descrita anteriormente.

El caso de las dependencias WAR la primera instrucción lee a un dato que luego es modificado por una segunda instrucción, estas condiciones también son llamadas antidependencias ya que el dato es leído antes que otra instrucción lo modifique, por lo que no se considera como riego de datos. De esta manera, solo las dependencias RAW pueden ocasionar riesgos de datos, sin embargo, esto también depende de la manera en cómo se ejecutan las instrucciones que implica cada escenario de dependencia.

Técnicas de solución para riesgos de datos

Para el manejo de los riesgos de datos obtenidos en la microarquitectura planteada, se han utilizado 2 tipos de soluciones, estas son:

- Solución por inserción de paro (*stall instertion*).
- Solución por adelantamiento (*forwarding*).

La solución por *forwarding* consiste en hacer uso del dato generado por las instrucciones una vez que este se encuentre disponibles en alguna de las etapas del pipeline. La figura 4 muestra el hardware en azul que se ha añadido a la microarquitectura de la figura 1 para la implementación de las técnicas *forwarding y stall insertion.* En esta figura se observa como los datos obtenidos de las etapas EX y MEM son enviados a la etapa ID para ser utilizados un ciclo después en la etapa EX. Dichos multiplexores son controlados por un hardware capaz de detectar las dependencias, el cual es identificado en la figura 4 como HZD_CTL. Este tipo de solución difiere de la propuesta por [2], en donde, los datos son adelantados desde las etapas MEM y WB hacia EX, sin embargo, realizar este adelantamiento desde una etapa anterior (Figura 4), permite obtener un mejor balance, con respecto al tiempo de retardo entre las etapas del pipeline como se sugiere en [4].

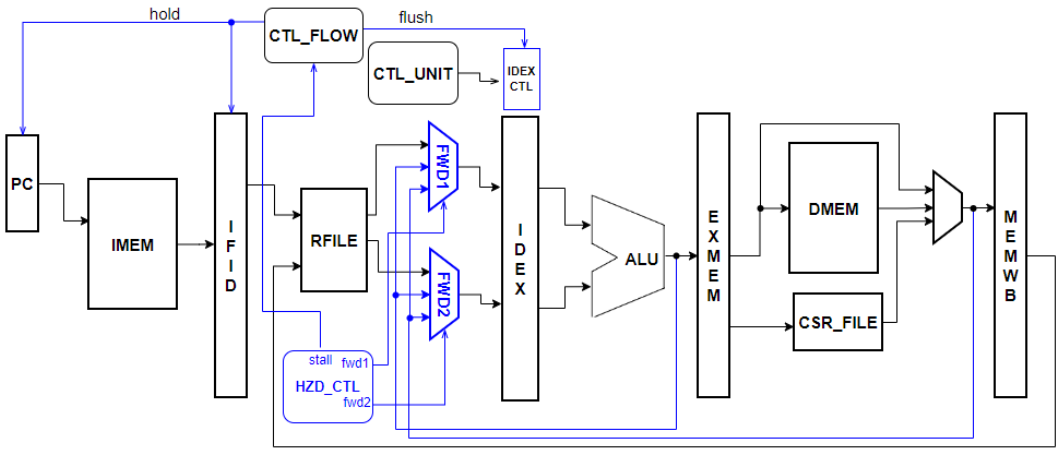

Figura 4 Hardware propuesto para la implementación de *forwarding* y *stall insertion*.

Por otro lado, los riesgos que no pueden ser solucionados por adelantamiento deben ser resueltos a través de la técnica *stall insertion*, la cual implica mantener retenida la instrucción que requiere el dato mientras las instrucciones de las etapas adelante continúan su ejecución. De esta forma, una vez que el dato solicitado haya sido generado, este puede ser adelantado por el hardware de *forwarding* y así evitar más retrasos en el flujo de las instrucciones.

Para la implementación de esta técnica se han añadido 2 señales provenientes de la unidad CTL_FLOW: “hold” y “flush”; y una más proveniente de la unidad

HZD_CTL identificada como “stall”. La señal “stall” indica a la unidad CTL_FLOW cuando una condición de riesgo no puede ser resuelta a través de *forwarding*, de esta forma, la señal “hold” se activa actuando como como señal de deshabitación de reloj para los registros IFID e IDEX, reteniendo así las instrucciones de las etapas IF e ID, mientas que la señal “flush” inhibe las escrituras a las señales de control del registro IDEX, evitando que la instrucción en la etapa ID se propague a EX en el siguiente ciclo.

Durante la sección 2 se plantea una metodología para la identificación de los riesgos de datos a través de tablas de análisis y categorización del conjunto de instrucciones seleccionado, además se establecen los tipos de soluciones disponibles para cada escenario de riesgo. En la sección 3 se realiza un análisis de las tablas resultantes y se plantea la segunda parte de la metodología para la validación del modelo obtenido en la sección presente aplicando los casos de dependencias de datos encontrados en la sección 2. En la sección 4 se discute acerca de puntos importantes que deben ser considerados en algunos escenarios de riesgo específicos, además se establecen puntos de mejora en la microarquitectura que fueron evidenciados a través de los resultados del método. Por último, en la sección 5 se concluye que el método presentado obtuvo los resultados esperados con relación a la detección de los riesgos de datos implicados en una microarquitectura segmentada y se identifican las condiciones en las que este método puede o no ser aplicado.

2. Método

Para la detección de todos los riegos de datos que pueden ser ocasionados en un uP *pipeline* es necesario identificar las fuentes de datos y los destinos que implican las instrucciones en su proceso de ejecución, para esto en la tabla 1 se ha establecido un agrupamiento de las instrucciones con relación a estas características, permitiendo un análisis por combinaciones de tipo de instrucciones y no por combinaciones de instrucciones individuales, reduciendo así el número de escenarios que deben ser analizados. Cabe señalar que no siempre los elementos fuente o destino están incluidos en el formato de la instrucción, ya que algunas

instrucciones hacen uso de recursos de hardware de manera implícita, tal es el caso de las instrucciones tipo ENV y MRET.

Tabla 1 Relación entre instrucciones con sus elementos fuentes y destinos.

Tipo	Instrucciones	Fuentes	Destinos
R	add, sub, and, or, xor, slt, sltu, sra, srl, sll	**rs1**, **rs2**	**rd**
I	addi, andi, ori, xori, slti, sltui, srai, srli, slli, jalr	**rs1**, **(**imm, shamt)	**rd**
B	blt, bltu, bge, bgeu, beq, bne	**rs1**, **rs2**, offset	pc
L	lw, lh, lhu, lb, lbu	**rs1**, offset, **mem**	**rd**
S	sw, sh, sb	**rs1**, **rs2**, offset	**mem**
U	lui, auipc	(pc), imm	**rd**
J	jal	pc, offset	pc, **rd**
CSRR	csrrw, csrrs, csrrc	**rs1, csr**	**rd**, **csr**
CSRI	csrrwi, csrrsi, csrrci	**csr**, zimm	**rd**, **csr**
MRET	mret	**mepc**, **mstatus**	**mstatus**
WFI	wfi	na	na
ENV	ecall, ebreak	**mtvec**	**mepc**, **mcause**, **mtval**

Como se estableció anteriormente, los riegos de datos pueden ser ocasionados solo por dependencias RAW, las cuales conllevan 2 instrucciones: una que realiza una escritura y otra que realiza una lectura al mismo elemento de *hardware* accedido por la primera, de esta manera es posible relacionar los elementos fuente con los elementos destino de acuerdo con el elemento de *hardware* al que pertenecen. La tabla 2 establece esta relación. Por ejemplo, los elementos fuente "**rs1**" y "**rs2**" están relacionados con el elemento destino "**rd**" ya que estos se refieren a registros del RFILE. De esta manera los elementos fuente que no están relacionados con algún elemento destino son descartados del análisis de riesgos de datos. Por otro lado, el elemento PC tampoco es contemplado para este análisis, ya que las escrituras a este registro producen una condición de salto, lo cual, está relacionado a los riesgos por control, que no están incluidos para su análisis en este trabajo.

Tabla 2 Relación destino – fuente.

Destino	Fuente	Hardware
rs1	rd	RFILE
rs2	rd	
mem	mem	DMEM
csr	csr, mstatus, mepc, mtvec, mtval, mcuause	CSR_FILE
mstatus	csr, mstatus	
mtvec	csr	
mepc	csr, mepc	
pc	pc	PC

Identificación de dependencias de datos

Una vez identificados los elementos fuentes y destino y su relación con el elemento de hardware al que pertenecen, es posible establecer las dependencias RAW obtenidas a través de matrices que contengan las combinaciones de los tipos de instrucciones que accedan a los elementos relacionados de la tabla 2. Para esto se les identificará como instrucciones dependientes (denotada como I_d) a la instrucción que realiza la operación de lectura a un elemento fuente en cuestión y como instrucción proveedora (denotado como I_p) a la instrucción que realiza la operación de escritura al elemento destino relacionado. La tabla 3 define las dependencias relativas al elemento "**rs1**", la tabla 4 las dependencias relativas al elemento "**rs2"**, tabla 5 al elemento "**mem"**, tabla 6 al elemento "**mepc**", tabla 7 al elemento "**mstatus**", tabla 8 al elemento "**mtvec"** y tabla 9 al elemento "**csr**"

Tabla 3 Matriz de dependencias referente a "RS1".

RS1 / Rd		INSTRUCCIONES DEPENDIENTE					
		R(2)	I(2)	L(2)	S(2)	B(2)	CSRR(2)
INSTRUCCIONES PROVEEDORAS	R(5, 3)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)
	I(5, 3)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)
	L(5, 4)	(2)(1)	(2)(1))	(2)(1)	(2)(1)	(2)(1)	(2)(1)
	J(5, 3)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)
	U(5, 3)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)	(2)(0)
	CSRR(5, 4)	(2)(1)	(2)(1))	(2)(1)	(2)(1)	(2)(1)	(2)(1)
	CSRI(5, 4)	(2)(1)	(2)(1))	(2)(1)	(2)(1)	(2)(1)	(2)(1)

Tabla 4 Matriz de dependencias referente a "RS2".

RS2 / Rd		INSTRUCCIONES DEPENDIENTE		
		R(2)	S(2)	B(2)
INSTRUCCIONES PROVEEDORAS	R(5, 3)	(2)(0)	(2)(0)	(2)(0)
	I(5, 3)	(2)(0)	(2)(0)	(2)(0)
	L(5, 4)	(2)(1)	(2)(1)	(2)(1)
	J(5, 3)	(2)(0)	(2)(0)	(2)(0)
	U(5, 3)	(2)(0)	(2)(0)	(2)(0)
	CSRR(5, 4)	(2)(1)	(2)(1)	(2)(1)
	CSRI(5, 4)	(2)(1)	(2)(1)	(2)(1)

Tabla 5 Matriz de dependencias referente a "mem".

MEM / MEM		INST DEP
		L (4, 4)
INST PROV	S (4, 4)	(-1) (-1)

Tabla 6 Matriz de dependencias referente a “mepc”.

CSR, MEPC / MEPC		INST DEP
		MRET(2)
INST PROV	CSRR(4,4)	(1)(1)
	CSRI(4,4)	(1)(1)
	ENV(2,2)	(-1)(-1)

Tabla 7 Matriz de dependencias referente a “mstatus”.

CSR, MSTATUS / MSTATUS		INST DEP
		MRET(2)
INST PROV	CSRR(4,4)	(1)(1)
	CSRI(4,4)	(1)(1)

Tabla 8 Matriz de dependencias referente a “mtvec”.

CSR / MTVEC		INST_DEP
		ENV(2)
INST PROV	CSRR(4,4)	(1)(1)
	CSRI(4,4)	(1)(1)

Tabla 9 Matriz de dependencias referente a “csr”.

CSR, MSTATUS, MTVEC, MEPC, MTVAL y MCAUSE / CSR		INSTRUCCIONES DEPENDIENTE	
		CSRR(4)	CSRI(4)
INST PROV	CSRR(4, 4)	(-1)(-1)	(-1)(-1)
	CSRI(4, 4)	(-1)(-1)	(-1)(-1)
	MRET(2, 2)	(-3)(-3)	(-3)(-3)
	ENV(2, 2)	(-3)(-3)	(-3)(-3)

Los números entre paréntesis adjuntos a los tipos de instrucciones tanto proveedoras como dependientes, así como los números entre paréntesis incluidos en las casillas de las tablas son utilizados para identificar si los escenarios de dependencias establecidos en las matrices ocasionan efectivamente un riesgo de datos y si estos pueden ser resueltos a través de adelantamiento o requiere de alguna inserción de paro, a continuación se revisará la manera en cómo se obtienen estos datos y como representarlos.

Identificación de riesgos de datos

La estructura de las tablas obtenidas en la subsección anterior identifica las dependencias implicadas en el ISA a partir de las combinaciones posibles de las instrucciones, sin embargo, para determinar si estas pueden causar algún riesgo de

datos es necesario determinar si el elemento fuente leído por la ***I_d*** es accedido uno o más ciclos de reloj antes que la escritura del elemento destino realizada por la ***I_p*** suceda, esta condición puede ser descrita mediante la ecuación 1.

$$\Delta_c(C) = W_c - R_c - C \geq 1 \quad (1)$$

Donde:

W_c **:** Ciclo de reloj en donde ***I_p*** realiza la operación de escritura.

R_c **:** Ciclo de reloj donde ***I_d*** realiza la operación de lectura.

C **:** Ciclos de diferencia con las que son ejecutadas ***I_p*** e ***I_d***

Δ_c **:** Ciclos de diferencia entre la lectura y la escritura del dato.

Cabe aclarar que W_c y R_c son obtenidos normalizando el ciclo 1 en inicio de cada instrucción, por lo cual se vuelven parámetros constantes e intrínsecos de cada tipo de instrucción, mientras Δ_c es un parámetro que define cada escenario de dependencia y puede variar con respecto a C. Por su parte C no puede ser igual a 0, ya que esto implicaría que ambas instrucciones comienzan su ejecución en el mismo ciclo de reloj, lo cual no puede ocurrir en un sistema de un solo *hart* y un valor negativo implicaría que ***I_d*** fuese ejecutada antes que ***I_p*** lo que equivale a una antidependencia**,** que como ya se mencionó, no son causa de riesgos de datos.

Los números entre paréntesis seguido al tipo de ***I_d*** de las matrices de dependencias indican el parámetro R_c; mientas que el primer número entre paréntesis seguido del tipo de ***I_p*** indica el parámetro W_c de cada tipo de instrucción, de esta manera se obtiene la diferencia Δ_c perteneciente al número del primer paréntesis mostrado en la casilla correspondiente a cada escenario incluido en las matrices de dependencias, donde los casos que producen riesgos de datos son marcados con números rojos, mientras los que no son marcados con números negros. Todos los valores de Δ_c han sido calculados tomando el parámetro $C = 1$, donde ***I_p*** e ***I_d*** han sido ejecutadas de manera consecutiva.

Identificación del tipo de solución para los escenarios de riesgos de datos

Para encontrar el tipo de solución que puede ser aplicado a cada escenario incluido en las tablas de dependencias, es necesario identificar si el dato generado

por la ***Ip*** (que será denotado como ***Dw***) es obtenido en el mismo ciclo o ciclos de reloj anteriores al que la ***Id*** requiera hacer uso de él. Ya que si ***Dw*** no es obtenido al momento que la ***Id*** lo requiere, se necesitará detener la ***Id*** hasta que ***Dw*** esté listo para ser adelantado, o en su defecto, esperar a que este haya terminado de escribirse en su destino. Esto puede ser expresado a través de la ecuación 2, donde se establece que un riesgo de datos puede ser resuelto por *forwarding*.

$$\Delta O_c(C) = O_c - R_c - C \leq 0 \qquad (2)$$

Donde:

ΔO_c **:** Ciclos de diferencia entre la obtención de ***Dw*** y la lectura del elemento fuente indicado por la ***Id***.

O_c **:** ciclo de reloj en donde la ***Ip*** realiza la obtención de ***Dw***

De nuevo O_c y R_c son obtenidos normalizando el ciclo 1 en inicio de cada instrucción, de esta manera si ΔO_c es igual a 0, entonces ***Dw*** es generado en el mismo ciclo de reloj en que se realiza la lectura del elemento fuente, este caso es mostrado en la figura 5a, en donde se observa que ambos eventos son realizados en el mismo ciclo en diferentes etapas del pipeline, de esta forma ***Dw*** puede ser adelantado a la etapa donde este es requerido al final del ciclo (lo cual es indicado por la flecha azul). El caso en que ΔO_c es mayor a 1 mostrado en la figura 5.b, implica que ***Dw*** es obtenido 2 o más ciclos después a la lectura del elemento fuente, por lo que este no puede ser adelantado en el siguiente ciclo y se requerirá realizar las inserciones de paro durante los ciclos de reloj correspondientes a ΔO_c.

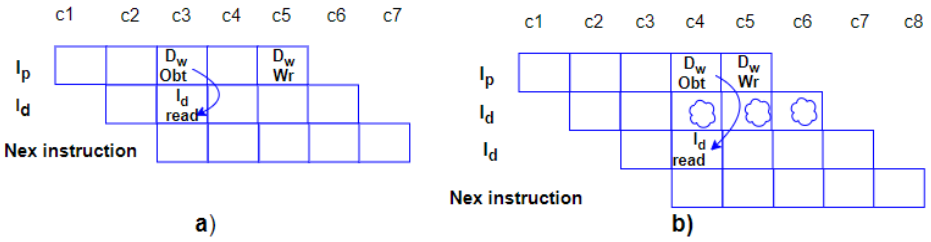

Figura 5 “csr_func” y “wb_func”.

El parámetro O_c es mostrado en el segundo número entre paréntesis adjunto al tipo de ***Ip*** en las matrices de dependencias mientras el parámetro ΔO_c pertenece al

número en el segundo paréntesis incluido en cada casilla de las tablas. Así, los casos donde el riesgo pude solucionarse por *forwarding* son marcados con números negros, mientras que los que requieren de inserciones de paro son marcados con un número en rojo. De nuevo, ΔO_c es calculado tomando el peor caso donde $C = 1$, lo que implica que las instrucciones son ejecutadas consecutivamente.

3. Resultados

Mediante las matrices obtenidas a través del método aquí propuesto se han identificado y analizado un total de 82 escenarios de dependencias que son ocasionados por las combinaciones de los tipos de instrucciones incluidos en el ISA base de RISC V, de los cuales 69 de ellos han sido identificados como posibles causas de riesgos de datos, donde 36 de estos casos pueden ser resueltos por *forwarding*, sin la necesidad de inserciones de paro y 33 que no podrán ser resueltos sin *stall insertion*.

Para comprobar la veracidad del método propuesto se ha desarrollado un plan de pruebas que incluyen todos los escenarios de dependencias establecidos en las matrices obtenidas. Este conjunto de pruebas fue aplicado a una implementación de la microarquitectura planteada. Tanto para el desarrollo del *Device Under Test* (DUT) como para la ejecución de las pruebas se ha utilizado el software Alliance VLSI, el cual es un conjunto de herramientas libres que permite el desarrollo, síntesis y simulación de circuitos *Very Large Scale Integration* (VLSI).

El diseño del DUT se ha implementado en VHDL y se ha sintetizado hasta nivel compuertas, de esta manera fue posible encontrar el tiempo aproximado de retardo en cada etapa y así poder obtener un mejor balance de tiempo de retardo entre las etapas del pipeline.

Por su parte, las pruebas son ejecutadas a través de un ambiente de verificación basado en las herramientas disponibles en Alliance, las cuales son controladas y auxiliadas por scripts en lenguaje Perl. La figura 6 muestra la manera en que las pruebas son ejecutadas por medio del ambiente de verificación. Mediante este método de validación se comprueba que ningún riesgo de datos produjo algún error inesperado durante la ejecución de las pruebas.

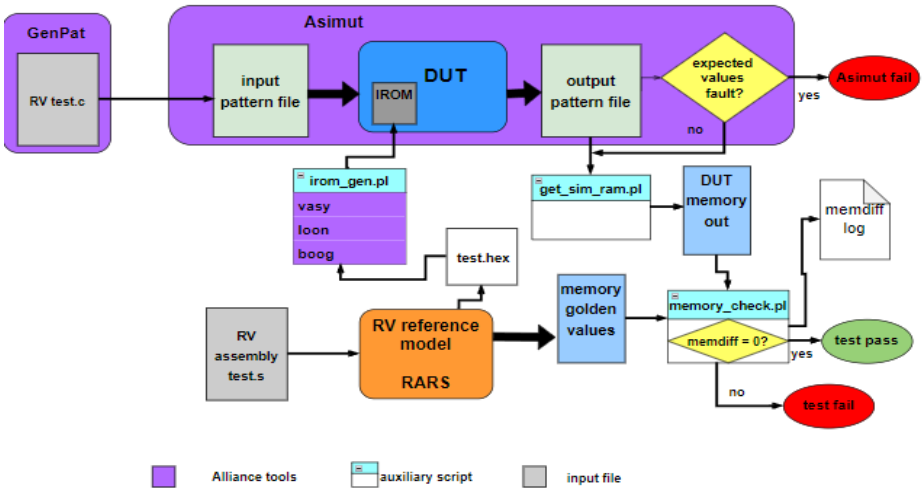

Figura 6 Ambiente de verificación basado en herramientas disponibles en Alliance VLSI.

4. Discusión

A través de las matrices obtenidas en la sección 2 se puede observar que las dependencias relativas a la memoria de la tabla 5, así como las referentes al elemento fuente "**csr**" de la tabla 9, no generan riesgos de datos, ya que tanto las lecturas como las escrituras son realizadas en el mismo ciclo de reloj relativo a cada instrucción, de esta manera se evita la integración de lógica adicional para su resolución. Por otro lado, por medio de estas matrices es posible evidenciar si algún escenario puede ser resuelto a través de reubicación de hardware, por ejemplo, los riesgos relativos al MEPC, MSTATUS, MTVEC de las tablas 6, 7 y 8 respectivamente, pudieran ser eliminados si este tipo de registros se reubicaran en la etapa EX y así evitar *hardware* adicional para su detección.

Otro punto que se debe examinar es con relación a los riegos de datos relativos a los elementos fuente "**rs1**" y "**rs2"** de las tablas 3 y 4, ya que en el caso específico donde la dirección tanto del registro fuente como el destino es igual a cero la técnica *forwarding* no debe ser implementada, puesto que este registro es de solo lectura por lo que no se cumplen las características que lo definen como riesgo de datos.

5. Conclusiones

El método propuesto en este trabajo establece una manera practica y visual para el análisis de los riesgos de datos que pueden ser ocasionados por las

dependencias implícitas en un conjunto de instrucciones RISC V, sin embargo, la implementación de este método no está limitado a solo la detección de estas condiciones, ya que también puede ser utilizado para la obtención de un plan de pruebas que validen todas las condiciones de dependencias de datos y como detección de posibles optimizaciones en el diseño del pipeline. El resultado de las pruebas muestra la veracidad del método no solo para los riesgos relacionados con el banco de registros RFILE, si no para todos los riesgos de datos ocasionados por elementos con memoria integrados a una microarquitectura segmentada.

6. Bibliografía y Referencias

[1] Patterson, D. A., Hennessy, J. L., Computer Organization and De-sign: The Hardware/Software Interface, RISC-V edition, ISBN: 978-0-12-812275-4, 2014

[2] Jer Huang, Alvin M. Despain, Hardware/Software Resolution of Pipeline Hazards in Pipeline Synthesis of Instruction Set Processors, 1993 International Conference on Computer Aided Design, DARPA J-FBI-91-194

[3] W. P. Kiat, K. M. Mok, W. K. Lee, H. G. Goh and I. Andonovic, A Comprehensive Analysis on Data Hazard for RISC32 5-Stage Pipeline Processor, 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA), Taipei, 2017, pp. 154-159, doi: 10.1109/WAINA.2017.20.

[4] He, Yihui & Wan, Han & Jiang, Bo & Gao, Xiaopeng. (2017). A Method to Detect Hazards in Pipeline Processor. MATEC Web of Conferences. 139. 00085. 10.1051/matecconf/201713900085.

[5] A. Pandey, Study of data hazard and control hazard resolution techniques in a simulated five stage pipelined RISC processor, 2016 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, 2016, pp. 1-4, doi: 10.1109/INVENTIVE.2016.7824864.

[6] A. Waterman, K. Asanovich, The RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 20191213, RISC-V Foundation, December 2019.

[7] A. Waterman, K. Asanovich, The RISC-V Instruction Set Manual Volume II: privileged Architecture, Ver 20211203, RISC-V Foundation, December 2021.