**GUÍA DEL ALUMNO**

**SESIÓN 2: Riesgos RAW en DLX**

# El Cauce en DLX y nuestras herramientas

El cauce del DLX consta de cinco etapas:

|  |  |
| --- | --- |
| **IF** | ***Instruction Fetch*: búsqueda de la instrucción. Incremento del PC** |
| **ID** | ***Instruction Decoding:* decodificación de instrucción, búsqueda de registros, evaluación de condición de salto y carga del PC** |
| **EX** | ***Execution*: Ejecución de operación(ALU), cálculo de direcciones efectivas(L/S)** |
| **ME** | ***Memory*: Acceso a memoria** |
| **WB** | ***Write Bank*: post-escritura en registro** |

La siguiente tabla muestra las operaciones que se hacen en cada etapa del cauce para los distintos tipos de instrucciones.

**Etapa ALU Load /Store Salto**

**Rd Reg1 op Reg2← Rd M(Reg1+Inm)← PC PC+ Inm←**

**M(Reg1+Inm) Reg2←**

**IF RI Mem(← PC) PC PC+4 ← Rbr PC← +dato Inm**

**ID A ← Reg1**

1. **Reg1← B ← Inm if(A.op.0)**
2. **Reg2 / Dato Inm← Rtmp ← Reg2 PC ← Rbr (Salto)**

**EX C ← A op B Rdir ← A + B**

**SRdat ← Rtmp**

**ME D ← C LRdat ← Mem[Rdir] (Load)**

**Mem[Rdir]← SRdat (Store)**

**WB Rd ← D Rd ← LRdat (Load)**

**TABLA de operaciones en el cauce** Donde:

* Inm es el dato inmediato y Reg1, Reg2 y Rd son los registros nombrados en la instrucción.
* El resto son registros de segmentación y auxiliares.
* En la fase ID se recogen los contenidos de los datos indicados
* Además, en cada fase se conserva la información de la instrucción que se esta ejecutando Esto está descrito de un modo similar en el Hennessy-Patterson capítulos 5 y 6.

Debes conocer la teoría relacionada con Riesgos de Datos y dentro de estos riesgos saber identificar los de tipo Read After Write (RAW), y usra un nuevo similador: WinDLXV

# Parte I: WinDLXV y Esquemas del cauce

La profesora explicará el uso de este simulador brevemente

1. Ejecuta en WinDLXV el programa del ejercicio 7 (contadores de ciclos) de la Sesión 1. Compara los ciclos resultantes con los obtenidos en DLXVsim: cuántos toma en cada simulador? Explica Por qué

La ejecución en el DLXVsim no tiene en cuenta los primeros 5 ciclos de procesador que se necesita para completar la primera instrucción. En el otro simulador se puede observar como a partir del 5 ciclo todas las instrucciones se hacen en un solo ciclo de reloj gracias a la segmentación.

1. Haz un programa que lea dos números, los sume, lea un 3º y lo acumule a lo anterior, dejando el resultado en una posición de memoria llamada result. Declara los datos necesarios. La idea es crear y observar dependencias RAW en el código.
   1. En WinDLXV seleccionar SIN adelanto de operandos en el Menú “Configuración”.

Observa y dibuja **un diagrama del cauce ciclo a ciclo para este caso**

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

* 1. Sin adelantamiento de operandos, las paradas son siempre de 2 ciclos para instrucciones con dependencias RAW ¿Cierto o falso?

Cierto.

# Parte II: Adelantamiento de Operandos

La técnica hardware llamada **adelantamiento de operandos** permite mitigar el efecto de las dependencias RAW (Read After Write). Las etapas EX y MEM pueden realimentar los resultados generados hacia las entradas de la etapa EX. El cauce de DLX utiliza esta técnica en DLXVSim siempre pero en WinDLXV puede activarse o desactivarse.

En un cauce de etapas monociclo, la técnica de adelantamiento permite que dependencias RAW entre **dos instrucciones ALU** seguidas no cause **ninguna parada**.

2. Deberás verificar esto y dibujar un diagrama del cauce.

Si la dependencia RAW es entre una instrucción de **carga y una instrucción ALU** subsiguiente, el adelantamiento permite que haya **un solo ciclo de detención** en el cauce. Veamos de nuevo el caso de una dependencia RAW entre una instrucción de carga y una ALU:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lw **r1**, a | IF | ID | EX | ***ME*** | WB |  |  |  |
| add r3,r2,**r1** |  | IF | ID | detención | ***EX*** | ME | WB |  |
| Instrucción X |  |  | IF | detención | ID | EX | ME | WB |

A pesar del adelantamiento aún existe un ciclo de parada. En ocasiones esto puede evitarse **reordenando** el código con una instrucción que **no tiene dependencias**. Por ejemplo la Instrucción X a continuación permite aprovechar el ciclo de parada y permite que el dato se envíe por el *bypass* de ME a EX a tiempo:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| lw **r1**, a | IF | ID | EX | ***ME*** | WB |  |  |
| Instrucción X |  | IF | ID | EX | ME | WB |  |
| add r3,r2,**r1** |  |  | IF | ID | ***EX*** | ME | WB |

Además de existir entre instrucciones LW-ALU y ALU-ALU las dependencias RAW se dan entre otros tipos de instrucciones. Es conveniente identificar, para cada tipo de instrucción, **dónde hay producción (escritura)** y **dónde consumo (lectura) de datos** para analizar si dependencias productor-consumidor pueden **detener la emisión** de instrucciones en el cauce. Para ello es necesario comprender qué sucede en cada etapa del cauce según el tipo de instrucción, tal y como se describe en la primera página. Un actividad más adelante te ayudará a estudiar en detalle este punto.

|  |  |
| --- | --- |
| Se proponen los siguientes experimentos que deberás hacer utilizando **siempre ADELANTO** | |
| **DE RESULTADOS en los simuladores.** | En DLXVSim, el adelantamiento está prefijado pero en |

WinDLXV he de activarse (Menú “Configuración”) para ver el comportamiento del cauce sin circuito de adelantamiento de resultados.

3. Haz a mano un diagrama del cauce para cada uno de los casos siguientes consultando la

tabla de operaciones de la primera página en cada caso: identifica las fases en que hay PRODUCCIÓN y CONSUMO y si hay entre ellas adelantamiento o no.

Una vez hecho esto comprueba tus predicciones con los dos simuladores.

|  |
| --- |
| **sub R1,R2,R3 add R4,R2,R1** |
| **lw R1,n(r0) add R2,R0,R1** |
| **lw R1,n(r0) sw n+4(r0),R1** |
| **lw R1,n(r0) sw n+4(R1),R7** |
| **lw R1,n(r0) bnez R1,fin** |
| **sub R1,R2,R3 bnez R1,fin** |

**Nota**: las instrucciones LW-SW no se comportan igual en los simuladores. Uno de ellos permite trasladar un resultado desde la salida de ME a su entrada mediante un circuito de adelantamiento, pero este circuito no se ha descrito realmente en la documentación del DLX.

Texto

Descripción generada automáticamente

**Recuerda: siempre con adelantamiento de operandos**

4- En el siguiente programa, predice y luego comprueba cuántos ciclos de parada se producen, dónde y por qué., Apúntalos al lado de la instrucción en que ocurren dentro de un comentario.

Sin adelantamientos:

Lo tiene Diego

Con adelantamientos:

Gráfico, Gráfico en cascada

Descripción generada automáticamente

Se producen dos paradas si intenta leer un elemento de la ALU, si intenta leer de memoria haciendo un load solamente realiza una parada.

¿Cuántos ciclos toma la ejecución de este código **en cada simulador**? Debes obtener una diferencia de **4 ciclos** entre lo que contabiliza un simulador y el otro – Recuerda por qué.

DLXVSim: 14 ciclos.

WinDLXVSim: 19 ciclos.

Se produce una diferencia de 4 ciclos porque el primer simulador ignora el tiempo de ejecución de la primera instrucción y lo mete como 1 ciclo.

|  |
| --- |
| **.data 0 a: .word 1, 2, 3, 4 b: .word 17, 18, 19, 20 c: .word 0, 0, 0, 0**  **.text 100 ini:**  **xor r7, r7, r7 addi r4, r0, b ; qué valor hay en r4?**  **lw r1, 4(r4) lw r2, b(r7)**  **add r3, r1, r2**  **sw 0(r4), r3 addi r7, r7, #4 add r3, r1, r7**  **sub r4,r4,r3 subi r2, r7, #4**  **bnez r2, ini nop**  **trap #6** |

# Reordenación de instrucciones

Los compiladores pueden realizar cambios en el orden en que aparecen algunas instrucciones en los programas para evitar paradas tal y como se mostró anteriormente para evitar la parada entre una instrucción de carga y una ALU que tienen dependencia de datos RAW.

Pero al hacer reordenación **no se puede alterar el algoritmo ni los registros usados.** únicamente algún dato inmediato en las instrucciones de carga o almacenamiento para ajustar cálculos de direcciones efectivas si éstas se ven afectadas por la reordenación.

1. Reordena el código del programa anterior para evitar las paradas identificadas. Verifica queno quedan paradas.

Código sin modificar: 15 ciclos y 4 paradas

Código modificado: 13 ciclos sin paradas.

1. Hacer un programa “**reverso.s**” que sume dos vectores a y b. El número de elementos a

sumar lo especificará una variable **n** y el resultado debe aparecer en orden inverso en el vector c. Por ejemplo, para la siguiente declaración de 6 elementos:

|  |  |
| --- | --- |
| **n:** | **.word 6** |
| **a:** | **.word 1, 2, 3, 4, 5, 6** |
| **b:** | **.word 11,12,13,14,15,16** |
| **c:** | **.space 24** |

en c debe aparecer 22, 20, 18, 16, 14, 12. Recuerda que has de poner una instrucción NOP después del salto (de momento).

* + Deberás usar la instrucción **SLLI** para calcular un índice que sea 4\*n pero en el simulador WinDLXV esta instrucción tiene problemas (no desplaza más allá del 5º bit). Entonces **no usar WinDLXV para este ejercicio sino únicamente DLXVSim**.
  + En primera instancia hacerlo **sin optimizar**, es decir, provocando el **máximo** de paradas que se puedan dar. **Analiza** los resultados estadísticos y las dependencias que producen detenciones en el cauce.
  + Realiza un cálculo **teórico** de los ciclos que consumirá el programa, identificando las paradas RAW (0,1 o 2) al lado de cada instrucción en que se produzca (\*). Escribe el total de ciclos con una ecuación que los detalle usando “i” ara instrucciones y “p” para paradas. Luego utiliza el simulador DLXVSim para comprobar si has acertado.

Lo tiene Diego.

El código consume 70 ciclos y tiene una parada.

1. Optimizar el programa anterior, reordenando las instrucciones para MINIMIZAR los ciclosconsumidos por la ejecución (evitar todas las paradas). Comprobar resultados y comparar con la ejecución sin reordenación.
   1. Obtener la Ganancia o Aceleración como la razón entre los ciclos de uno y otro caso para n=10.
   2. Recalcular la **aceleración para n=200.**

**–**

El código optimizado consume únicamente 58 ciclos y no tiene paradas.

Los cálculos de los apartados a y b los tiene Diego.

(\*) Diagrama de cauce hecho **a mano** **al lado de cada instrucción en que hay PARADA por dependencia RAW** mostrando cómo se produce.

◦ **ATENCIÓN**: no se ha de hacer el diagrama para todas las instrucciones, **únicamente** para las dos involucradas en la dependencia