

Prácticas Arquitectura de Computadores

BLOQUE II

Cronograma

Bloque 2 – Procesadores segmentados

- Riesgos
 - Riesgos de Datos: adelantamientos y reordenamiento de código
 - Riesgos Estructurales
 - Riesgos de Control: desenrollado de Bucles
-

Guion 1: Riesgos de datos y estructurales

- Dependencias de datos
- Adelantamientos
- Reordenamiento
- Ganancia

Guion 2: Riesgos de datos y de control

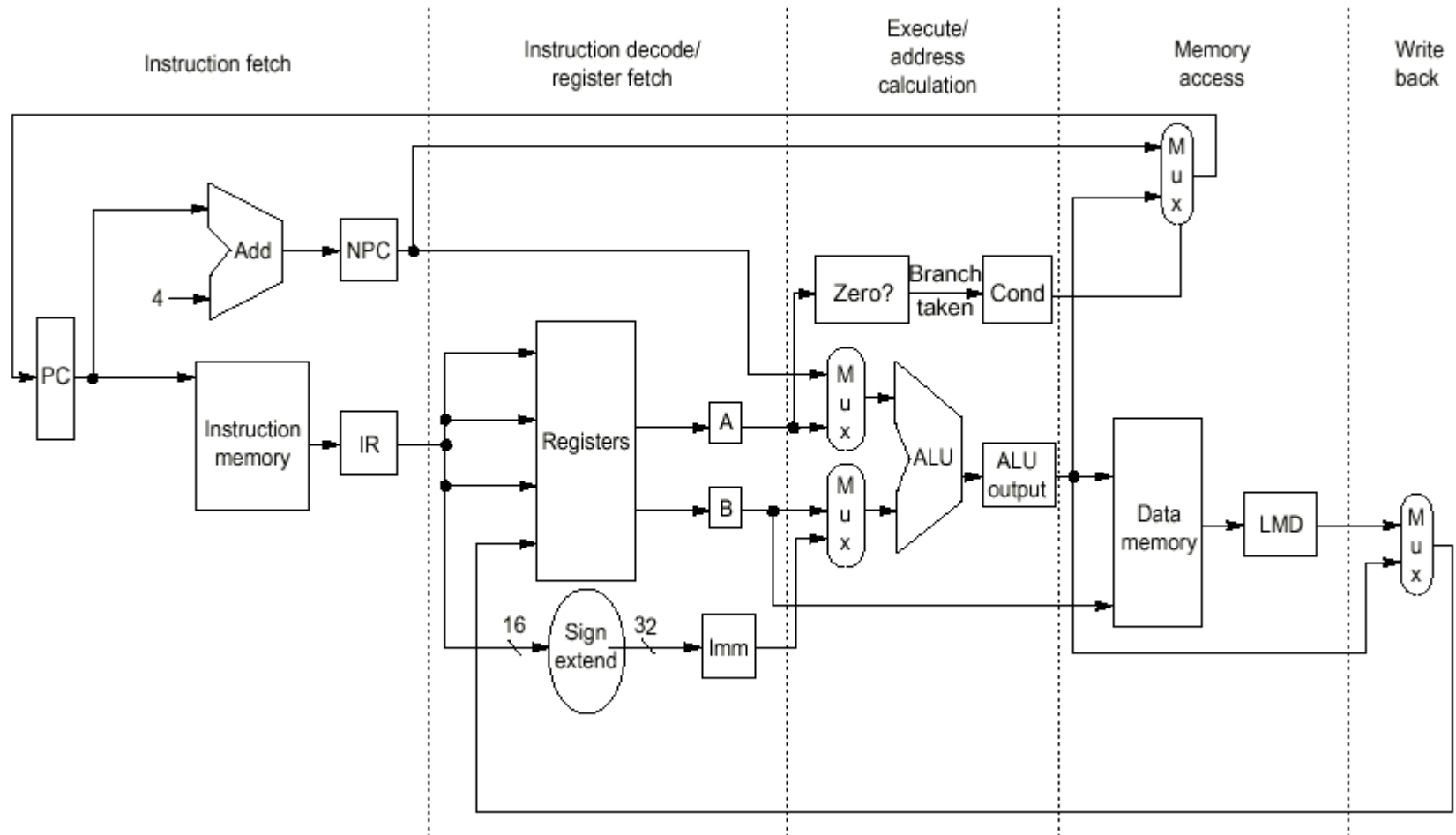
- Estudio del flujo del programa
- Riesgos de control
- Desenrollado de bucles
- Desenrollado de bucles y reordenamiento de instrucciones
- Ganancia

Prueba de validación – 31/10/2022

Bloque II – Procesadores Segmentados

RIESGOS DE DATOS Y ESTRUCTURALES

Ruta de datos del DLX



DLX: Etapas en la ejecución de instrucciones

En el procesador DLX toda instrucción puede ser ejecutada en 5 ciclos de reloj, siguiendo estas etapas:

- **IF** (*Instruction Fetch*). → Captación de la instrucción
- **ID** (*Instruction Decode*). → Descodificación
- **EX** (*Execute*). → Ejecución
- **MEM** (*Memory Access*). → Acceso a memoria
- **WB** (*Writeback*). → Escritura de resultados

Etapa IF

1. Se lleva al registro IMAR (*Instruction Memory Address Register*) la dirección de la siguiente instrucción a procesar, alojada en el PC.
2. Se carga la instrucción en curso en el IR (*Instruction Register*).
3. Se prepara el contador de programa para apuntar a la siguiente instrucción.

$$\text{IMAR} \leftarrow \text{PC}$$
$$\text{IR} \leftarrow \text{Mem}[\text{IMAR}]$$
$$\text{NPC} \leftarrow \text{PC} + 4$$

Etapa ID

1. Se descodifica la instrucción previamente captada
2. Se cargan los diferentes campos en los registros especiales A, B e Imm.
3. Los saltos condicionales son calculados en esta etapa con el fin de reducir los riesgos de control

$$A \leftarrow \text{Regs}[\text{IR6...10}]$$
$$B \leftarrow \text{Regs}[\text{IR11...15}]$$
$$\text{Imm} \leftarrow ((\text{IR16} \gg 16) \ll \text{IR16...31})$$

Etapa EX

En esta etapa el trabajo realizado depende del tipo de instrucción:

- **Acceso a memoria** (cálculo de la dirección de carga o almacenamiento)

$$\text{ALUOutput} \leftarrow A + \text{Imm}$$

- **Aritmética Reg-Reg ALU** (operación especificada por *func*)

$$\text{ALUOutput} \leftarrow A \text{ func } B$$

- **Aritmética Reg-Imm ALU** (operación especificada por *op*)

$$\text{ALUOutput} \leftarrow A \text{ op } \text{Imm}$$

- **Salto** (cálculo del PC destino y de la condición)

$$\text{ALUOutput} \leftarrow \text{NPC} + \text{Imm}$$

$$\text{Cond} \leftarrow (A \text{ op } 0)$$

Etapa MEM

En esta etapa el trabajo realizado depende del tipo de instrucción:

- **Acceso a memoria:** lectura en el registro LMD (*Load Memory Data*)

$$\text{LMD} \leftarrow \text{Mem}[\text{ALUOutput}]$$

- **Acceso a memoria:** escritura del contenido del registro B

$$\text{Mem}[\text{ALUOutput}] \leftarrow \text{B}$$

- **Salto:** carga de la dirección efectiva en el PC

```
if (cond)
    PC ← ALUOutput
else
    PC ← NPC
```

Etapa WB

En esta etapa el trabajo realizado depende del tipo de instrucción:

- **Aritmética** Reg-Reg ALU:

$$\text{Regs}[\text{IR16...20}] \leftarrow \text{ALUOutput}$$

- **Aritmética** Reg-Imm ALU:

$$\text{Regs}[\text{IR11...15}] \leftarrow \text{ALUOutput}$$

- **Carga** desde memoria (Load):

$$\text{Regs}[\text{IR11...15}] \leftarrow \text{LMD}$$

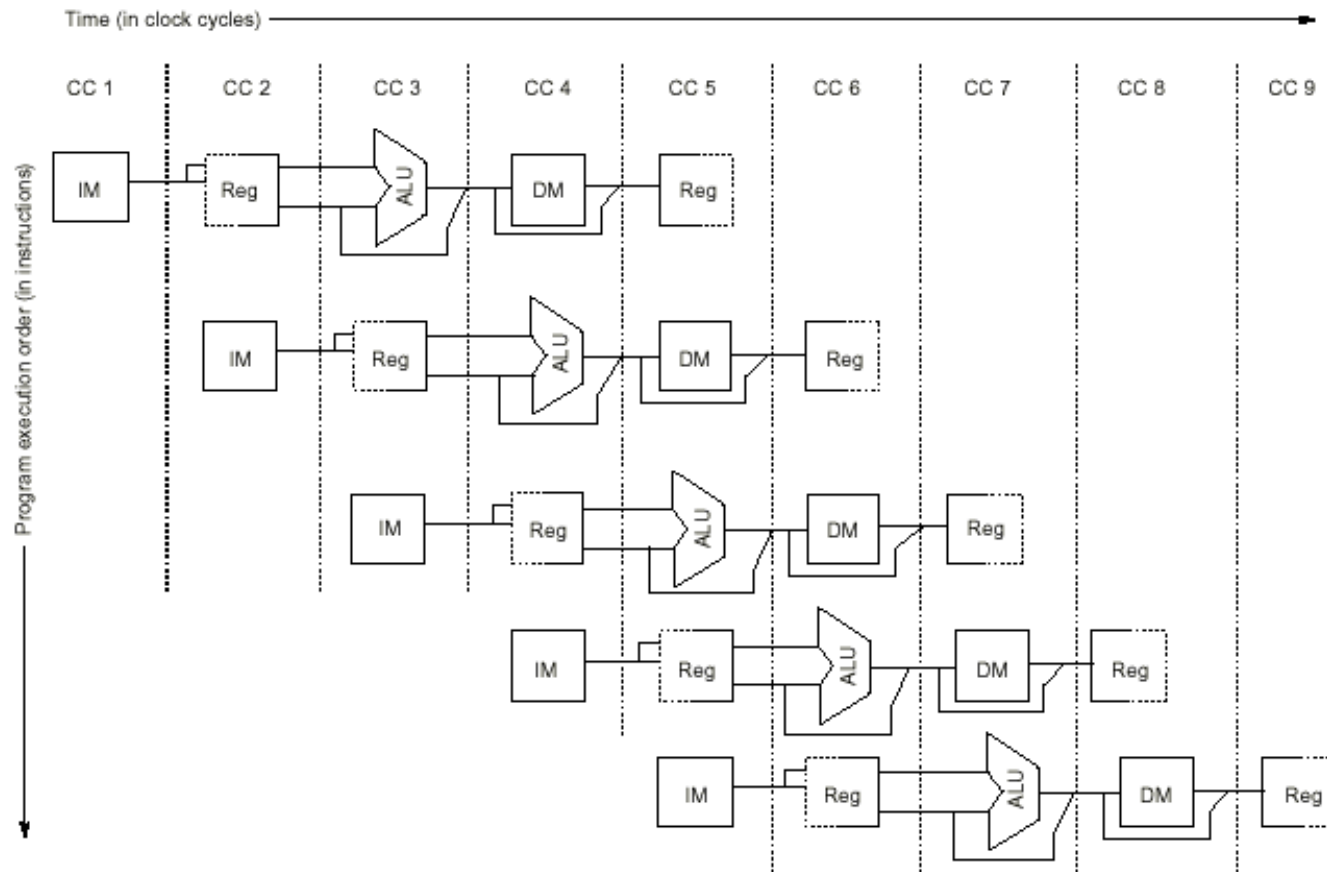
Segmentación a nivel de Instrucción

¿Qué es la segmentación a nivel de instrucción?

- Dividir la ruta de datos del procesador en varias etapas, a fin de optimizar su rendimiento.
- De esta manera, varias instrucciones pueden coexistir en la ruta de datos del procesador, siempre que se encuentren en distintas etapas.

Instruction number	Clock number								
	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction $i + 1$		IF	ID	EX	MEM	WB			
Instruction $i + 2$			IF	ID	EX	MEM	WB		
Instruction $i + 3$				IF	ID	EX	MEM	WB	
Instruction $i + 4$					IF	ID	EX	MEM	WB

Segmentación a nivel de Instrucción



¿Hay conflictos en el modelo?

PROBLEMAS

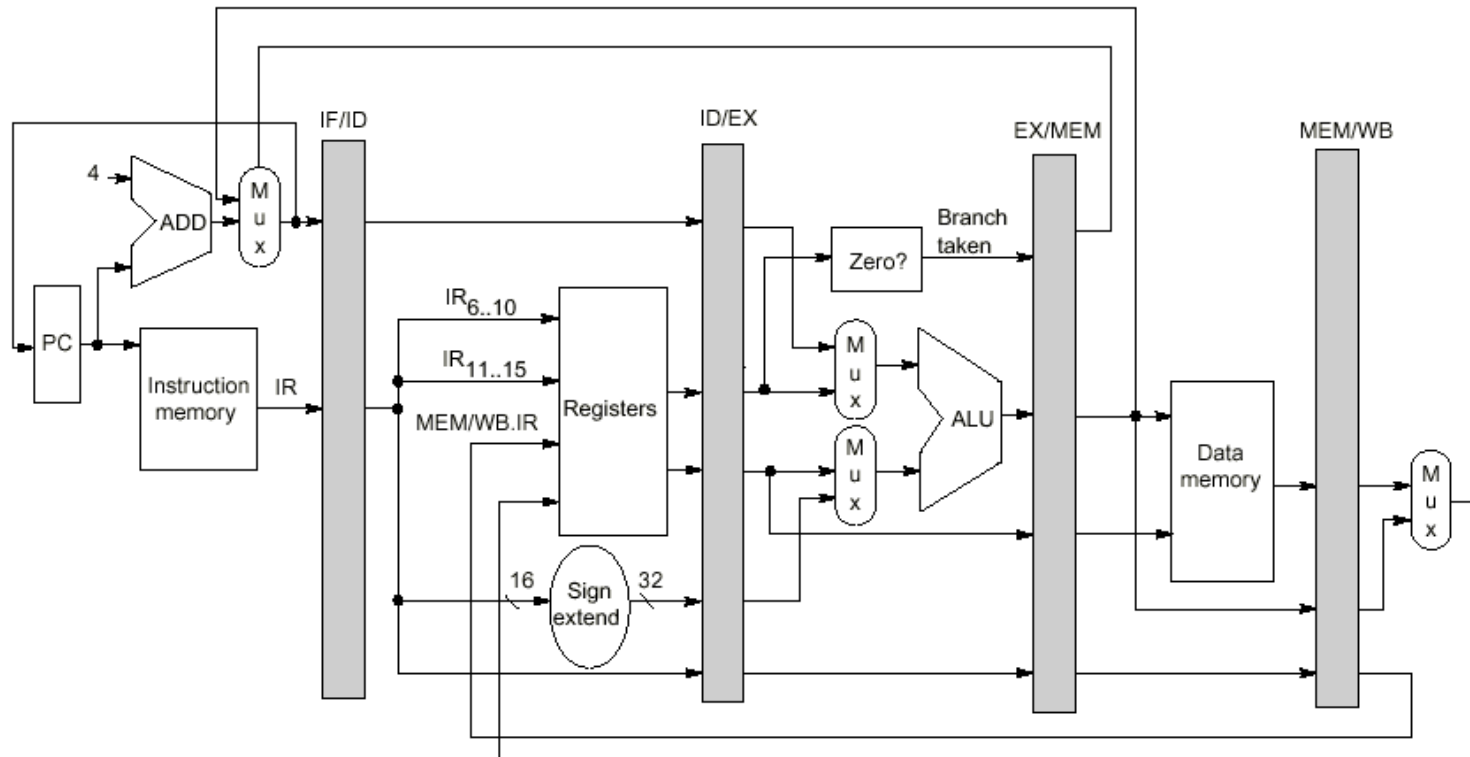
- 1. Las etapas **ID** y **WB** acceden al banco de registros simultáneamente.
- 2. Las etapas **IF** y **MEM** acceden a la memoria simultáneamente.

SOLUCIONES

- 1. El banco de registros se gestiona a doble ciclo (se escribe en la primera parte y se lee en la segunda).
- 2. Hay dos caches separadas (una de instrucciones y otra de datos).

Rutas datos segmentada del DLX

Los registros intermedios se funden en la etapa de *latch*. Se adelanta la carga del PC en saltos a la etapa de IF



Riesgos de la segmentación

Definición: se denominan riesgos (*Hazards*) a las situaciones que imposibilitan que la siguiente instrucción se ejecute en el ciclo predeterminado.

- Los riesgos reducen el rendimiento ideal de la máquina, provocando que el $CPI > 1$.

- Tipos de riesgos:

- Estructurales
- De datos
- De control

Riesgos estructurales

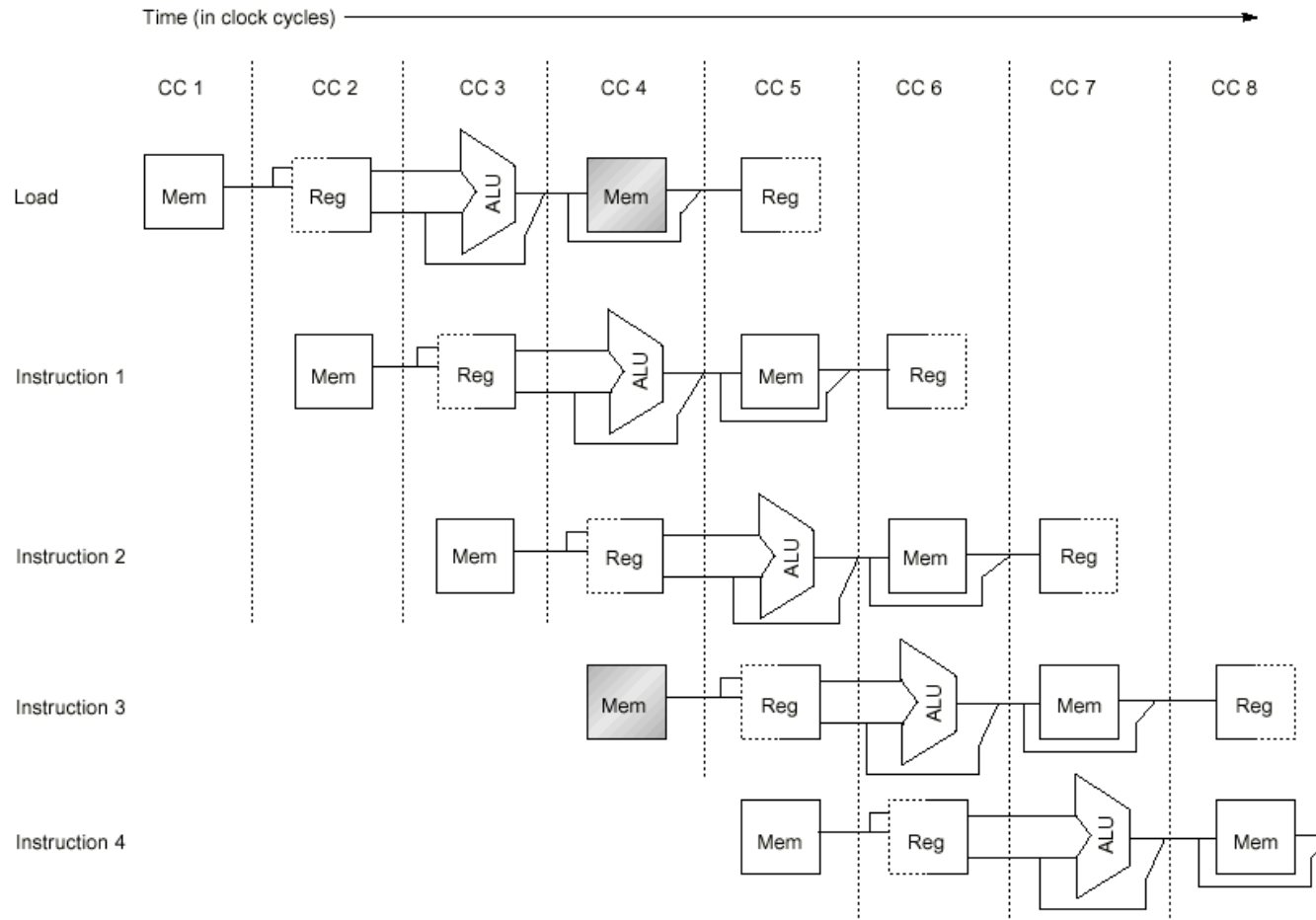
Situación en la que dos o más instrucciones tratan de hacer uso de un único recurso hardware.

Casos más habituales:

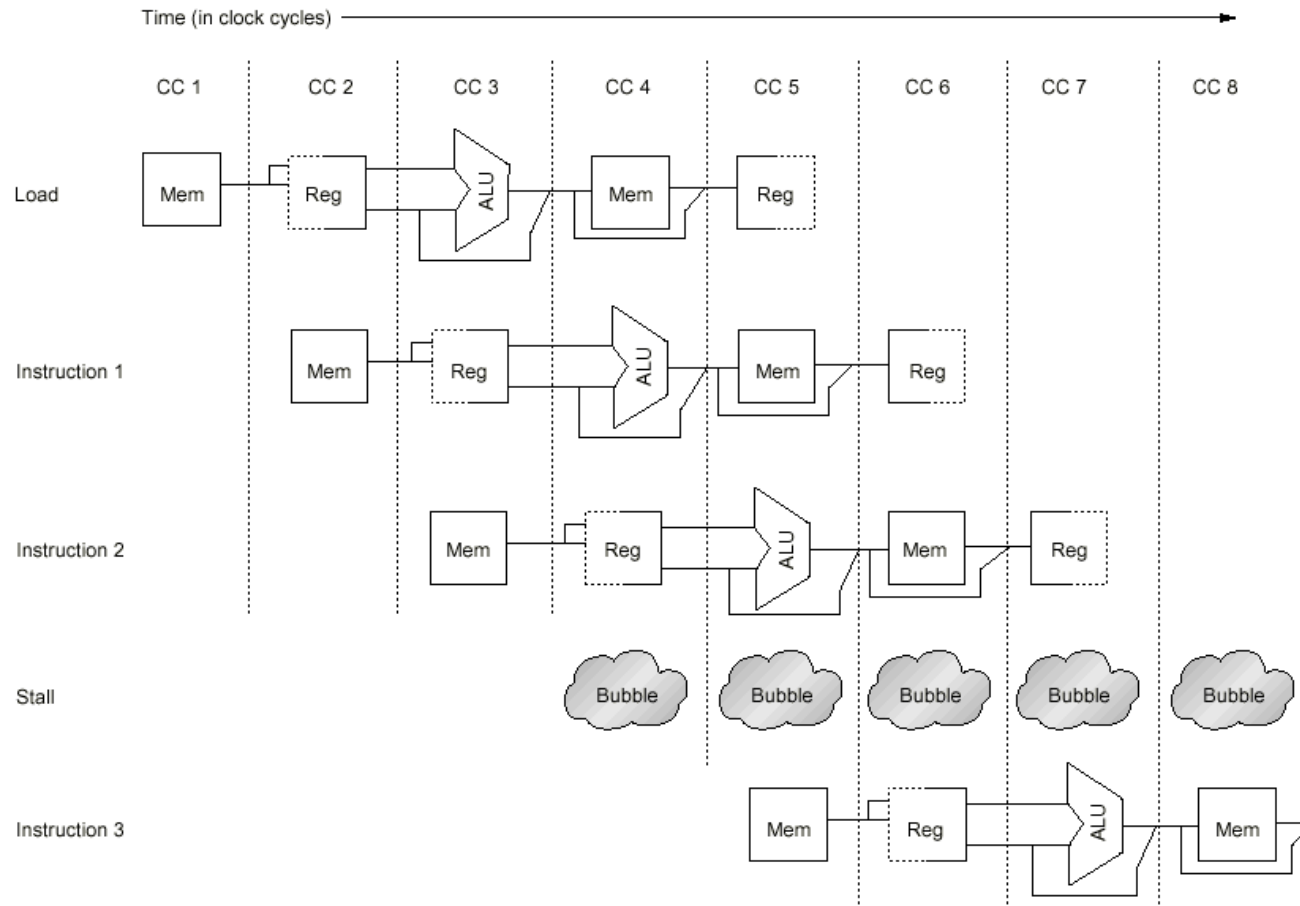
- Máquinas con una sola memoria (conflictos en lectura de datos e instrucciones)
- Unidades funcionales multi-ciclo no segmentadas

Solución al riesgo estructural: Parada de la unidad durante una etapa (introducción de una burbuja) \Rightarrow Reducción del rendimiento

Riesgos estructurales



Riesgos estructurales



Riesgos estructurales

	Clock cycle number									
Instruction	1	2	3	4	5	6	7	8	9	10
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Riesgos de datos

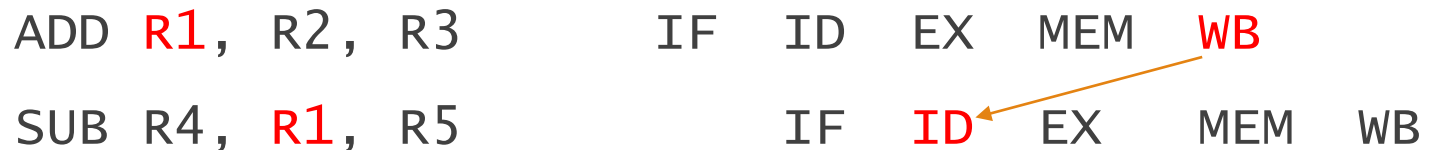
Situación en la que dos instrucciones que comparten datos tienen problemas de sincronización.

Tipos:

- **RAW** (*Read-After-Write*, Lectura después de escritura):
 - También conocida como “Dependencia”.
- **WAR** (*Write-After-Read*, Escritura después de lectura):
 - También conocida como “Anti-Dependencia”.
- **WAW** (*Write-After-Write*, Escritura después de escritura):
 - También conocida como “Dependencia de salida”.

Dependencia (RAW)

Ocurre cuando una instrucción necesita leer un dato que otra instrucción previa aun no han producido.



En este caso la instrucción SUB necesita en R1 el valor generado por la instrucción ADD, pero este no está disponible hasta la etapa WB.

Ejemplo de dependencia de datos

- La primera instrucción (ADD) genera un resultado en R1.
- El resto de instrucciones necesitan R1 como dato.
- No lo pueden usar hasta que sea generado.

ADD R1, R2, R3

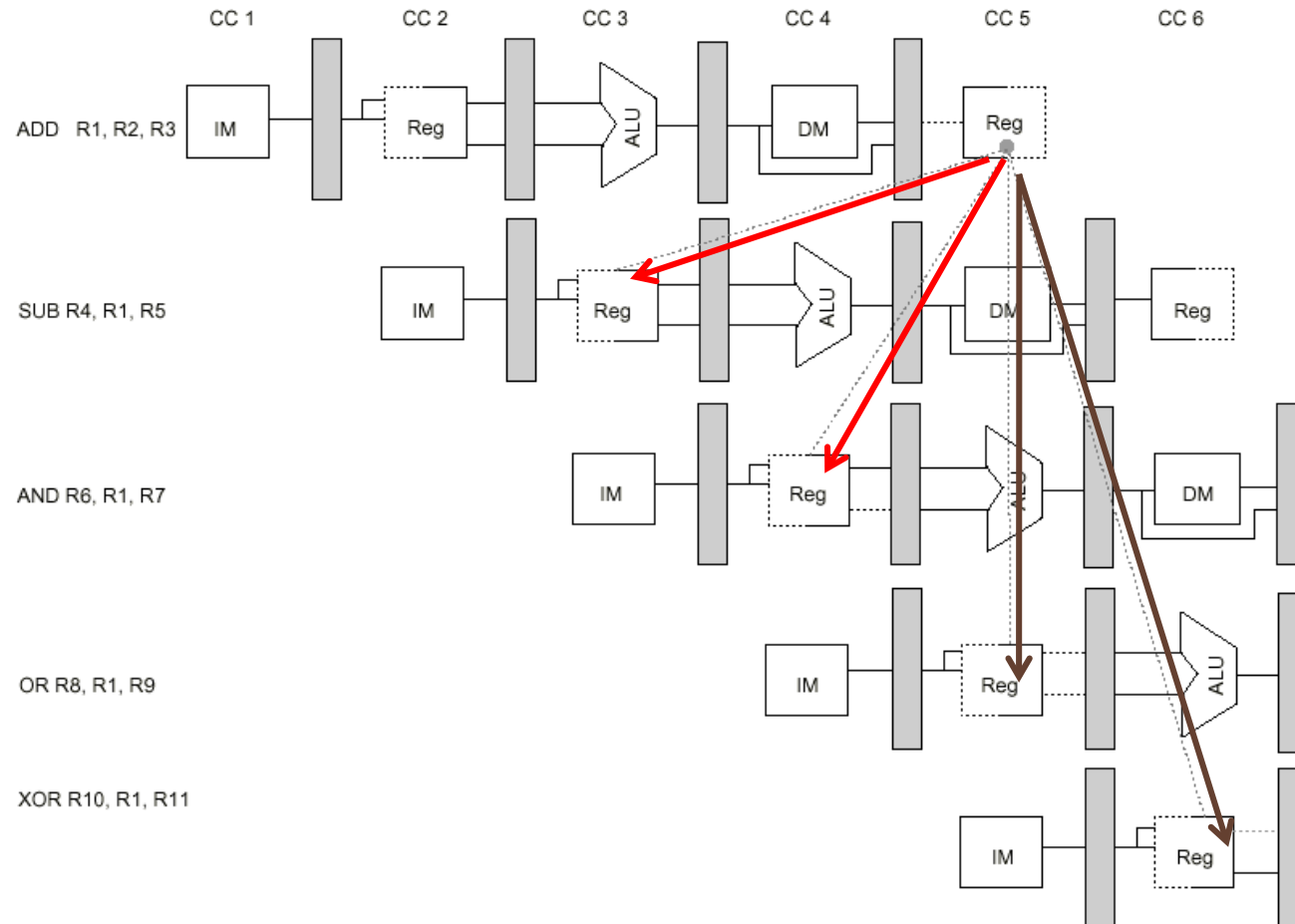
SUB R4, R1, R5

AND R6, R1, R7

OR R8, R1, R9

XOR R10, R1, R11

Ejemplo de dependencia de datos



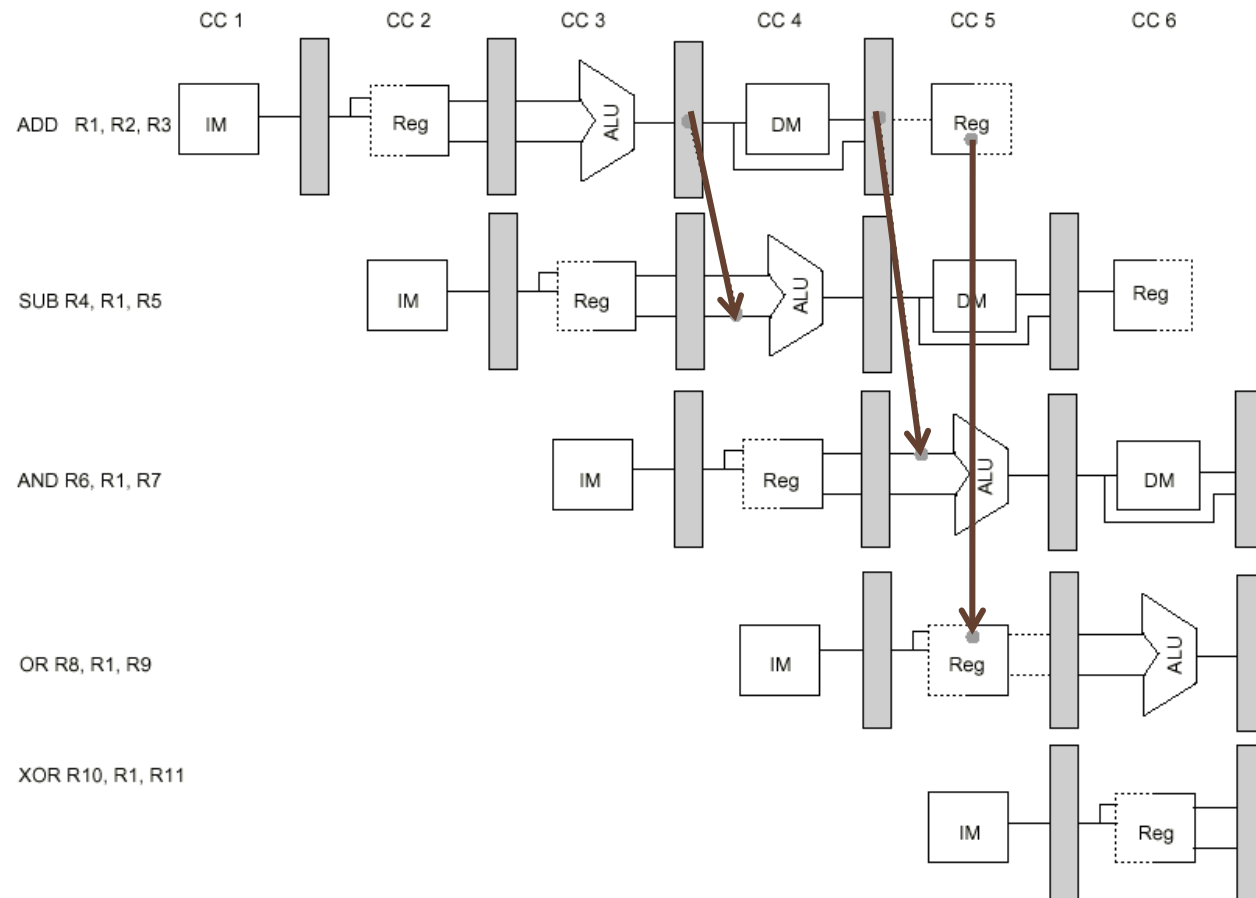
Anticipación de datos

Solución:

- Anticipación de datos (*Forwarding*), también conocido como Adelantamiento.

Los datos se transfieren directamente desde la unidad que los produce hasta la unidad que los consume, sin pasar previamente por el banco de registros.

Anticipación de datos



Riesgos de datos inevitables

No todos los riesgos de datos se pueden evitar por adelantamiento:

LW **R1**, 0(R2)

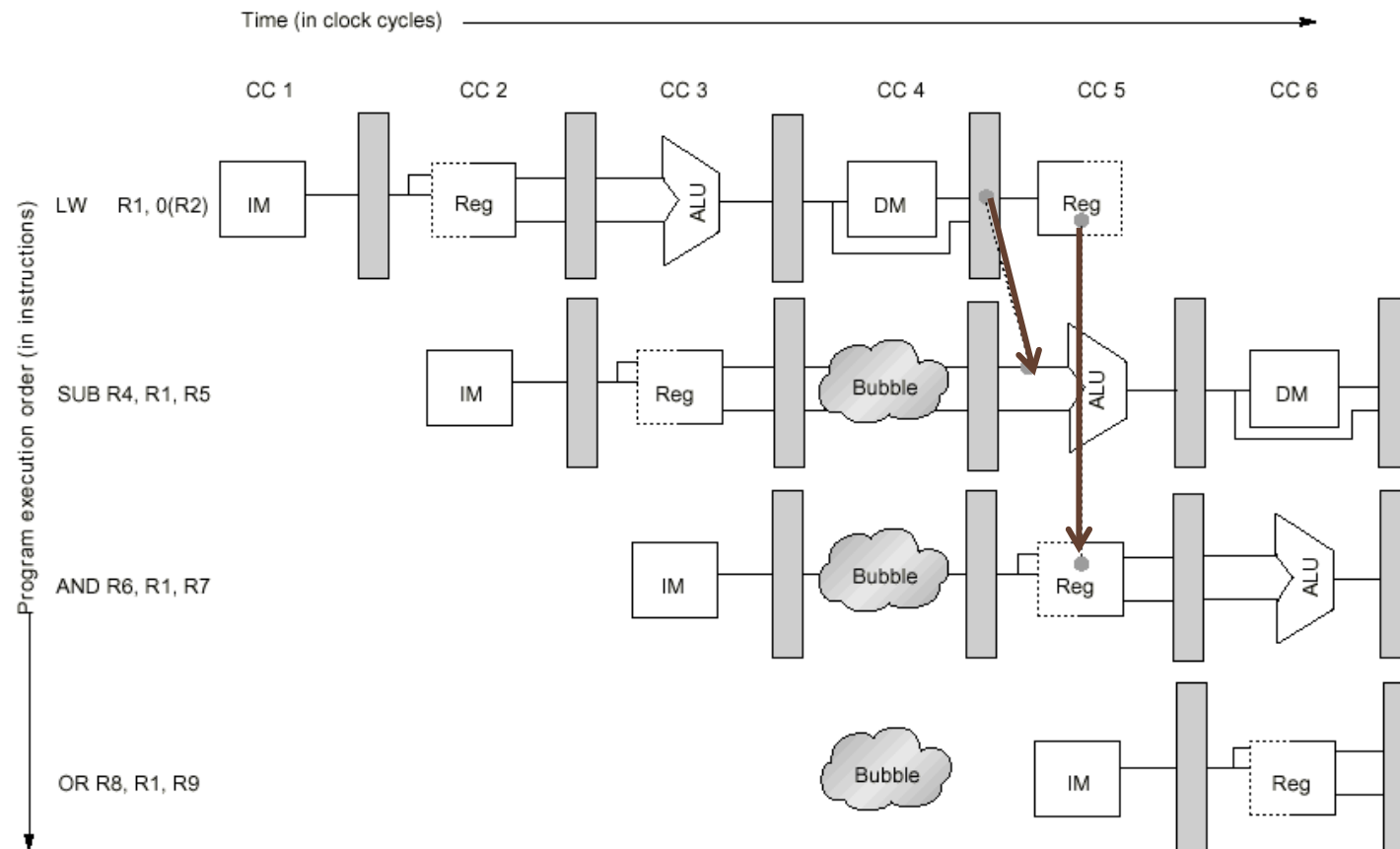
SUB R4, **R1**, R5

AND R6, **R1**, R7

OR R8, **R1**, R9

Ahora, el dato de R1 no se produce en EX (como antes), sino al final de MEM

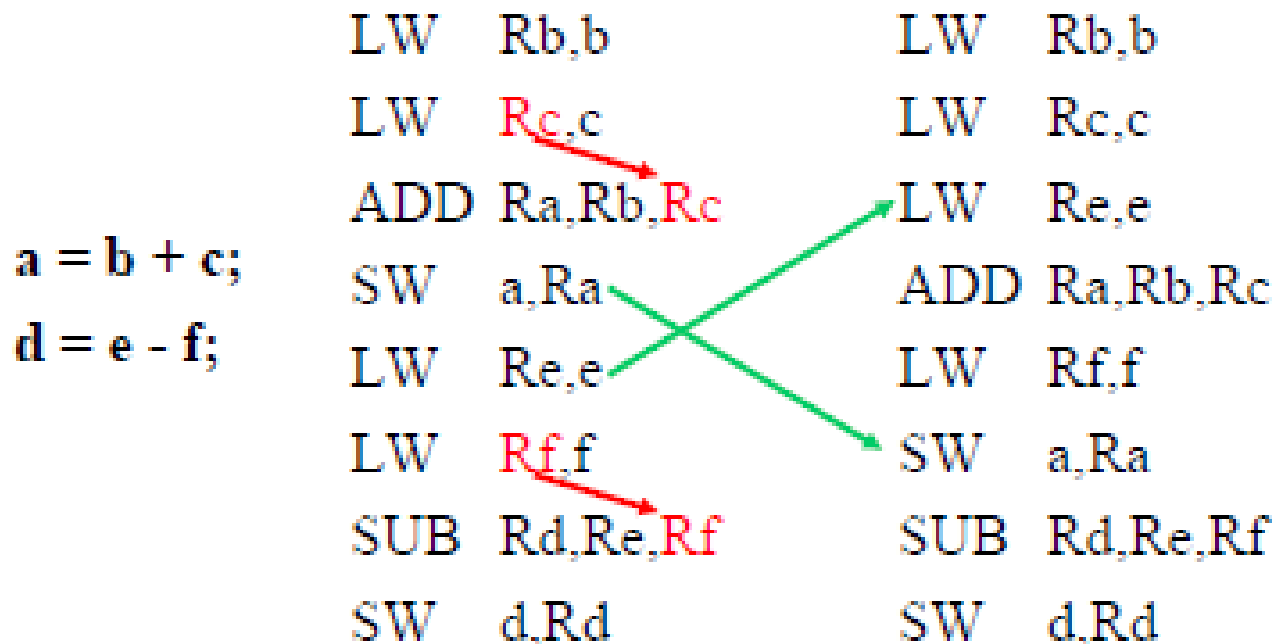
Riesgos de datos inevitables



Planificación estática: cambio de orden

El compilador cambia de orden las instrucciones:

Estático, antes de la ejecución.



Planificación estática: reordenamiento

Dependencias RAW:

- **Auténticas:** se tiene que producir el dato antes de poder usarlo:

LW **R1**, 0(R2)

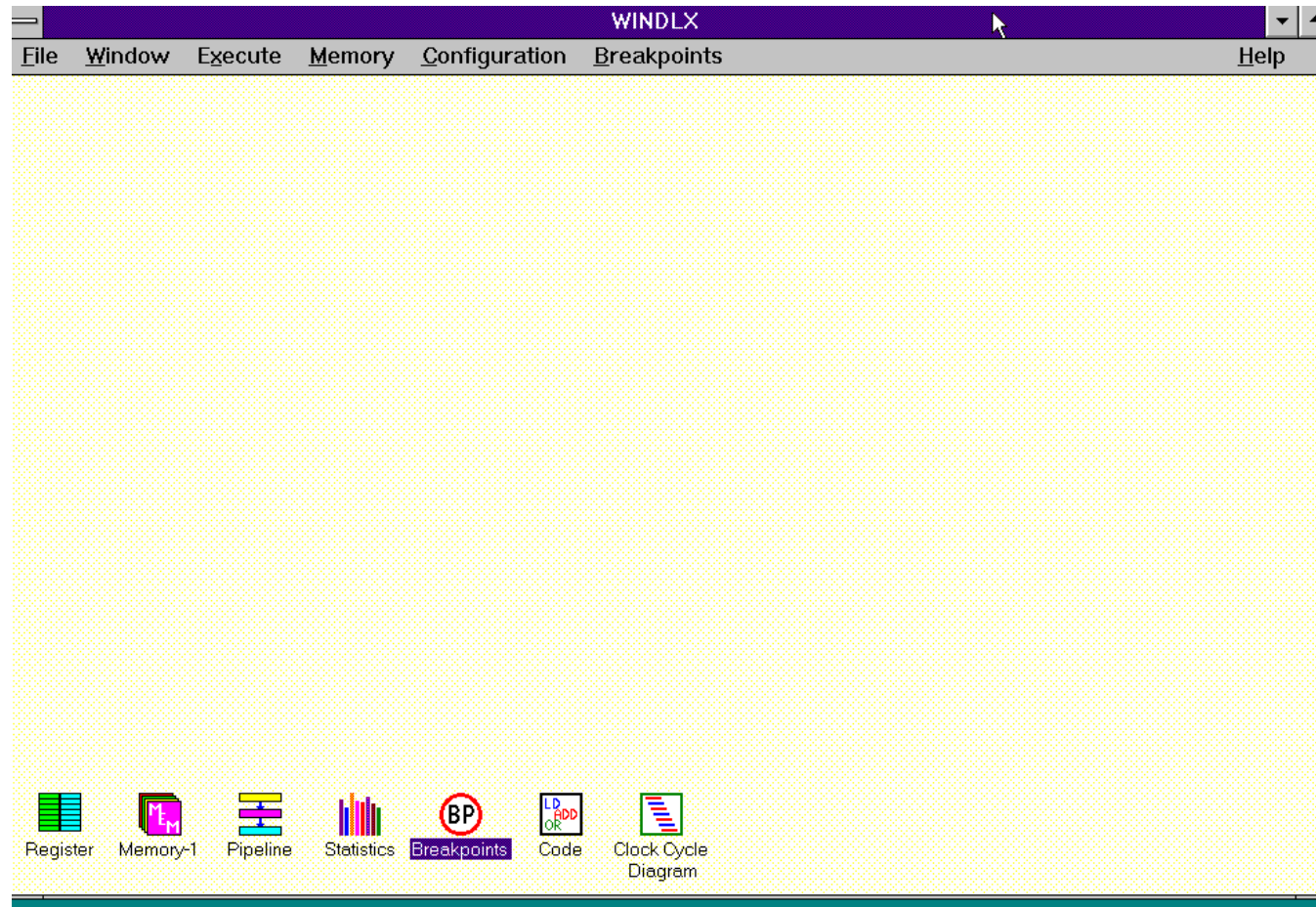
ADD R3, **R1**, R4

- **Solución:** reordenamiento

Bloque II – Procesadores Segmentados

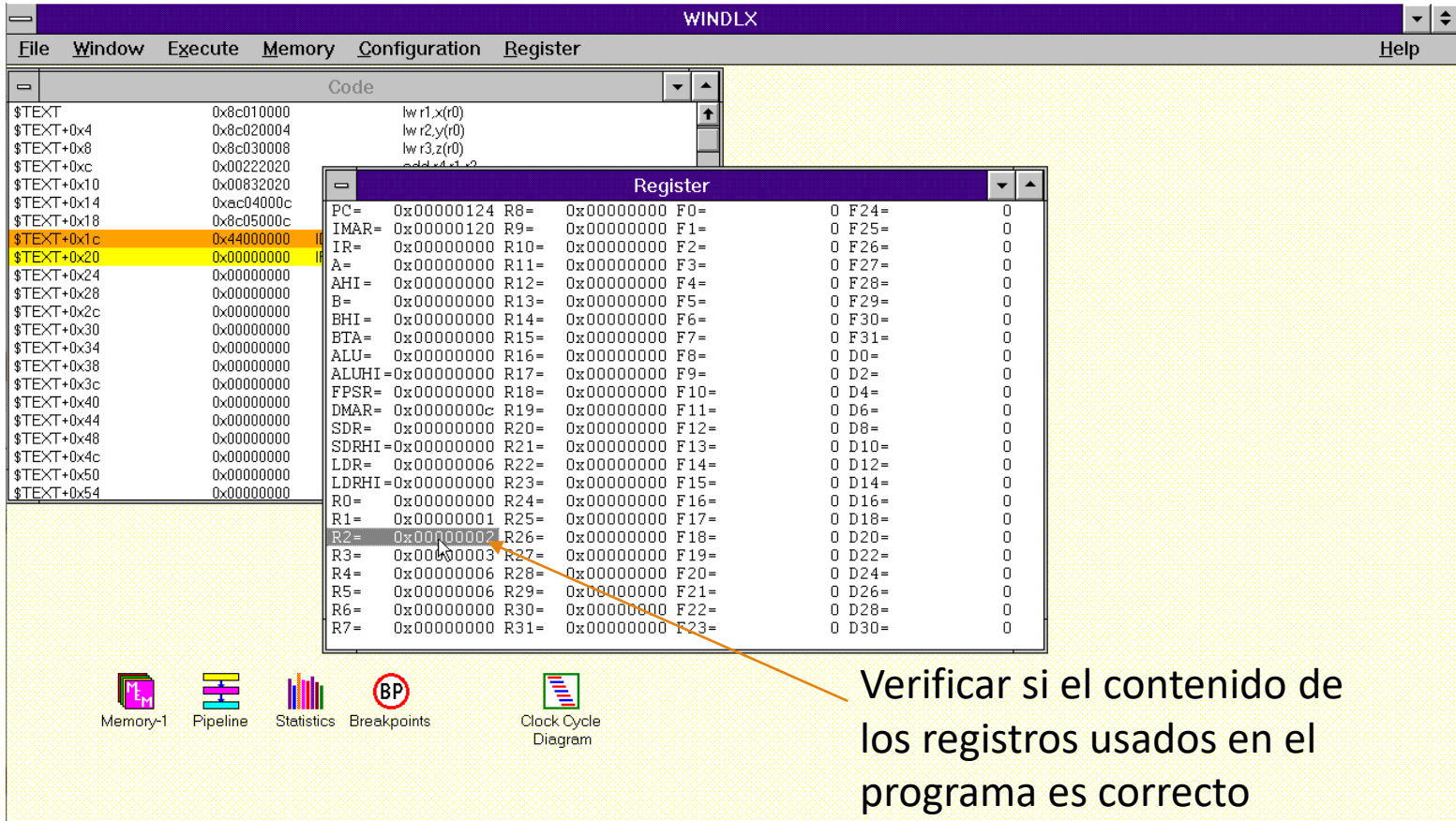
WINDLX

Interfaz de WinDLX



Interfaz de WinDLX

Comprobación de los registros



The screenshot shows the WinDLX interface with the Register window open. The Register window displays the values of 32 registers (R0-R31) and 32 floating-point registers (F0-F31). The R2 register is highlighted with a red arrow pointing to it from the text "Verificar si el contenido de los registros usados en el programa es correcto".

The Register window shows the following values:

Register	Value
PC	0x00000124
IMAR	0x00000120
IR	0x00000000
A	0x00000000
AHI	0x00000000
B	0x00000000
BHI	0x00000000
BTA	0x00000000
ALU	0x00000000
ALUHI	0x00000000
FPSR	0x00000000
DMAR	0x0000000c
SDR	0x00000000
SDRHI	0x00000000
LDR	0x00000006
LDRHI	0x00000000
R0	0x00000000
R1	0x00000001
R2	0x00000002
R3	0x00000003
R4	0x00000006
R5	0x00000006
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00000000
R15	0x00000000
R16	0x00000000
R17	0x00000000
R18	0x00000000
R19	0x00000000
R20	0x00000000
R21	0x00000000
R22	0x00000000
R23	0x00000000
R24	0x00000000
R25	0x00000000
R26	0x00000000
R27	0x00000000
R28	0x00000000
R29	0x00000000
R30	0x00000000
R31	0x00000000
F0	0
F1	0
F2	0
F3	0
F4	0
F5	0
F6	0
F7	0
F8	0
F9	0
F10	0
F11	0
F12	0
F13	0
F14	0
F15	0
F16	0
F17	0
F18	0
F19	0
F20	0
F21	0
F22	0
F23	0
F24	0
F25	0
F26	0
F27	0
F28	0
F29	0
F30	0
F31	0
D0	0
D1	0
D2	0
D3	0
D4	0
D5	0
D6	0
D7	0
D8	0
D9	0
D10	0
D11	0
D12	0
D13	0
D14	0
D15	0
D16	0
D17	0
D18	0
D19	0
D20	0
D21	0
D22	0
D23	0
D24	0
D25	0
D26	0
D27	0
D28	0
D29	0
D30	0
D31	0

Verificar si el contenido de los registros usados en el programa es correcto

Interfaz de WinDLX

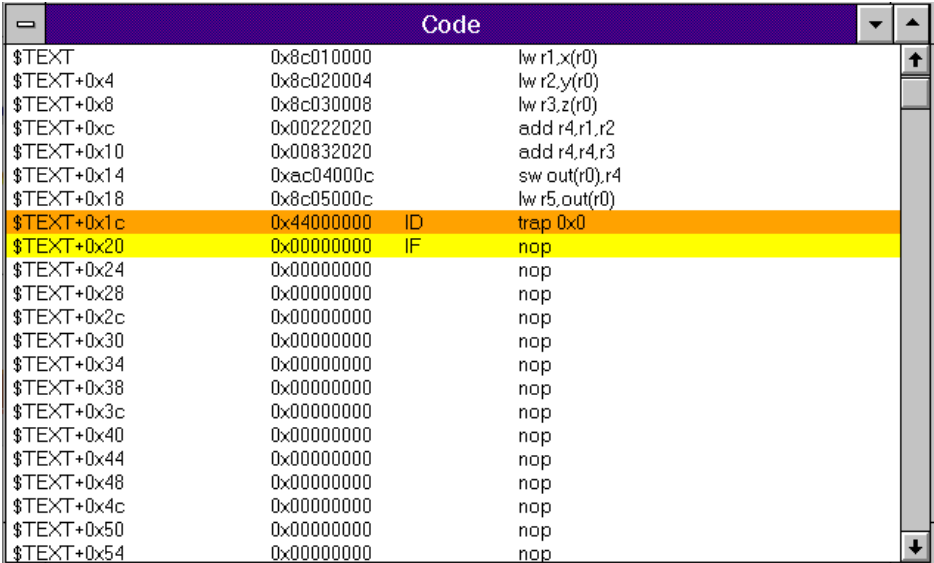
Ventana Code

Visualización:

- Instrucciones
- Puntos de parada (breakpoints)

Instrucción está ejecutándose en una etapa determinada del pipeline:

- Un color característico de cada etapa
- Aparece una etiqueta de la etapa



Code			
\$TEXT	0x8c010000		lw r1,x(r0)
\$TEXT+0x4	0x8c020004		lw r2,y(r0)
\$TEXT+0x8	0x8c030008		lw r3,z(r0)
\$TEXT+0xc	0x00222020		add r4,r1,r2
\$TEXT+0x10	0x00832020		add r4,r4,r3
\$TEXT+0x14	0xac04000c		sw out(r0),r4
\$TEXT+0x18	0x8c05000c		lw r5,out(r0)
\$TEXT+0x1c	0x44000000	ID	trap 0x0
\$TEXT+0x20	0x00000000	IF	nop
\$TEXT+0x24	0x00000000		nop
\$TEXT+0x28	0x00000000		nop
\$TEXT+0x2c	0x00000000		nop
\$TEXT+0x30	0x00000000		nop
\$TEXT+0x34	0x00000000		nop
\$TEXT+0x38	0x00000000		nop
\$TEXT+0x3c	0x00000000		nop
\$TEXT+0x40	0x00000000		nop
\$TEXT+0x44	0x00000000		nop
\$TEXT+0x48	0x00000000		nop
\$TEXT+0x4c	0x00000000		nop
\$TEXT+0x50	0x00000000		nop
\$TEXT+0x54	0x00000000		nop

Interfaz de WinDLX

Ventana Code - Detalles

Doble clic sobre cualquier instrucción abre el panel de detalles de la derecha

Information about add r4,r4,r3		
add r4,r4,r3 Adr.: \$TEXT+0x10 Code: 0x00832020 Terminated successfully First Cycle: -7 Last Cycle: -3 Total Cycles: 5	IF Cycles: -7(1) Terminated successfully IMAR<-PC (=\$TEXT+0x10) IR<-Mem[IMAR] (=0x00832020) PC<-PC+4 (=\$TEXT+0x14) No Stalls required.	ID Cycles: -6(1) Terminated successfully A<-R4 (=0x0) B<-R3 (=0x0) No Stalls required.
intEX Cycles: -5(1) Terminated successfully ALU<-A+B (=0x6) (A=0x3, B=0x3) No Stalls required. Forwarding applicated: A<-0x3 (add r4,r1,r2) B<-0x3 (lw r3,z(r0))	MEM Cycles: -4(1) Terminated successfully Nothing to do. No Stalls required.	WB Cycles: -3(1) Terminated successfully R4<-ALU (=0x6) No Stalls required.
<div>OK</div>		

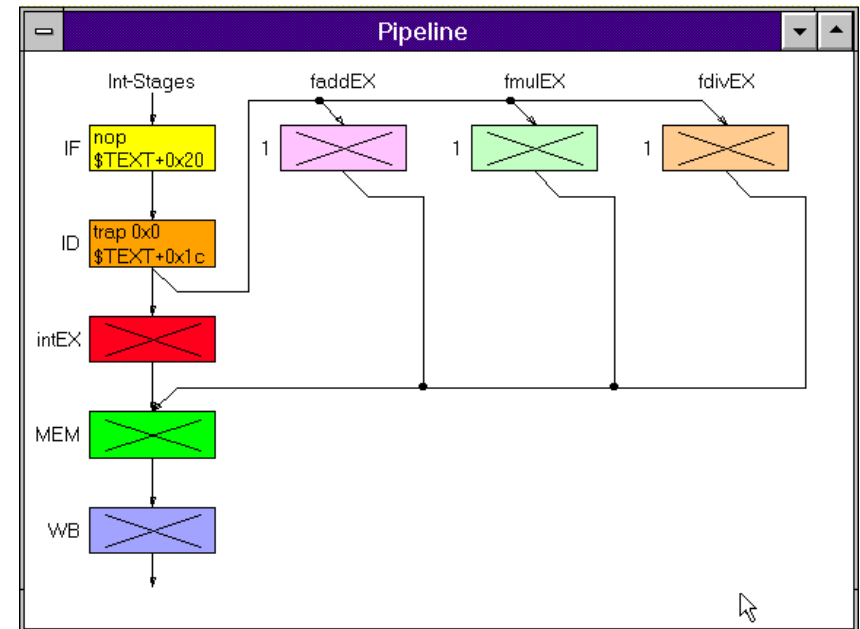
Interfaz de WinDLX

Ventana y menú Pipeline

Visualizan las etapas por las que pasan las instrucciones dentro de la estructura del pipeline del procesador.

El menú Pipeline:

- Display Floating point stages.
 - Activo:
 - Las etapas en coma flotante
 - Desactivado
 - Las cinco etapas básicas del pipeline del DLX



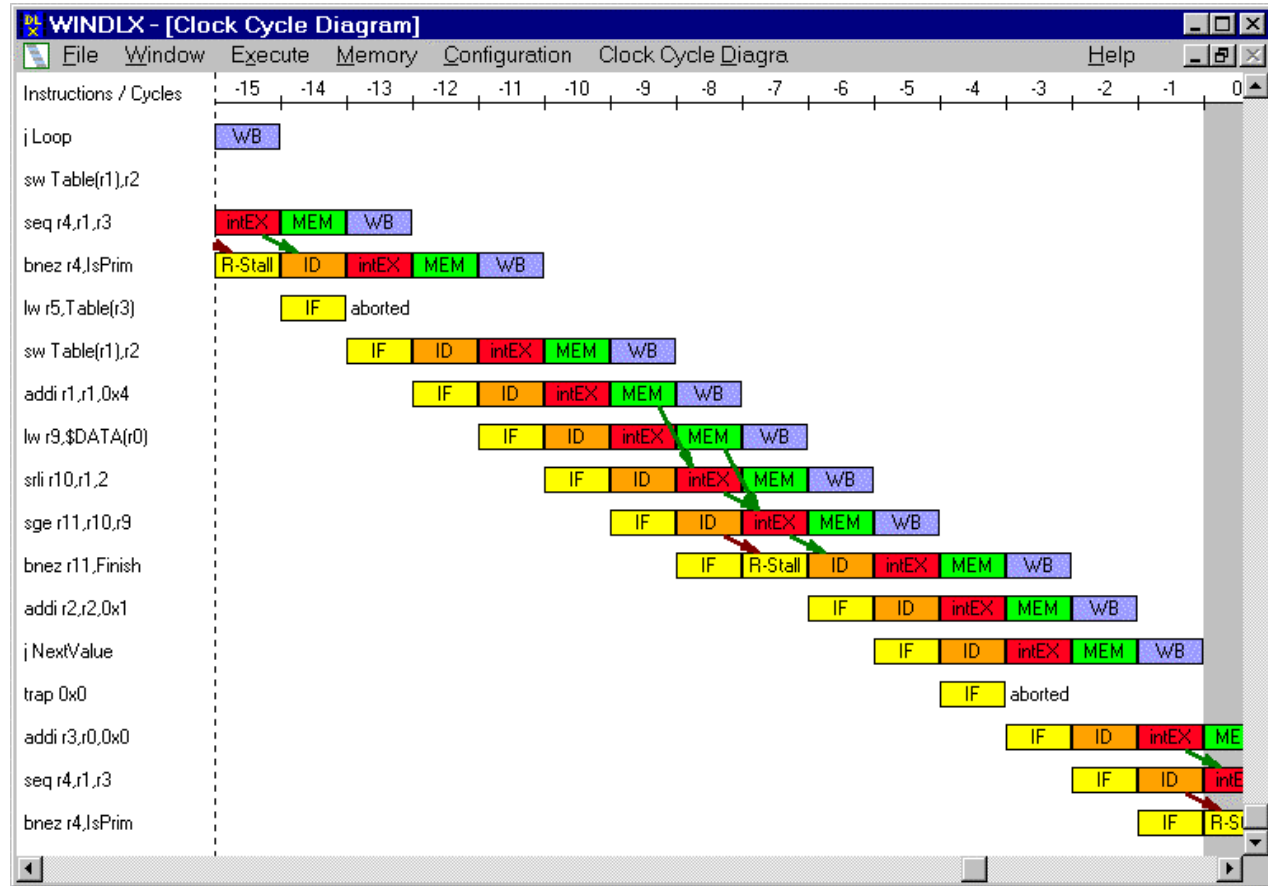
Interfaz de WinDLX

Ventana y menú Clock Cicle Diagram

Visualizan las operaciones que se realizan en cada ciclo de reloj y en cada etapa.

Cada columna representa el estado del *pipeline* en un ciclo de reloj.

El estado actual del *pipeline es representado en color gris en la columna situada en el extremo Derecho.*



Interfaz de WinDLX

Ventana y menú Clock Cycle Diagram

Las detenciones (*stalls*) son representadas en cajas coloreadas en el color asociado a la etapa detenida.

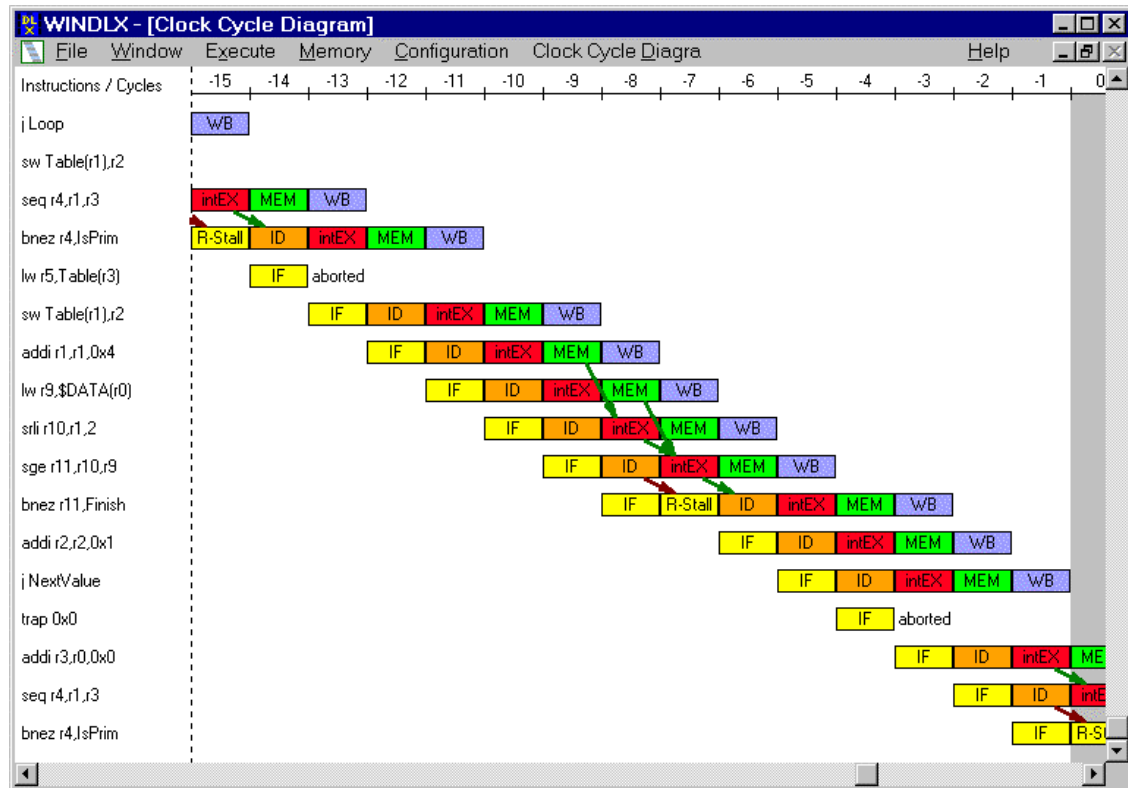
- **R-Stall (*Read After Write Stall*)**. Una flecha en color rojo señala la instrucción que está produciendo la detención por causa de este tipo de riesgo de datos.
- **T-Stall (*Trap Stall*)**. Esta detención sólo se produce ante una instrucción de TRAP. Esta permanece en la etapa IF hasta que no queden más instrucciones en el interior del pipeline.
- **W-Stall (*Write After Write Stall*)**.
 - Una flecha roja señala la instrucción que causa la detención.
 - Este riesgo sólo se presenta en pipelines que escriben en los registros o en memoria en varias etapas.
 - El pipeline de DLX escribe sólo los registros en la etapa WB, evitando esta clase de riesgos para las instrucciones enteras, pero no con las operaciones en coma flotante, como veremos más adelante.
- **S-Stall (*Structural Stall*)**. No existen suficientes recursos hardware para ejecutar la instrucción.
- **Stall**. Cuando una instrucción de coma flotante está en la etapa MEM, la próxima instrucción será detenida en la etapa intEX etiquetándola con la palabra *Stall*.

Interfaz de WinDLX

Ventana y menú Clock Cycle Diagram

El menú Clock Cycle Diagram:

- Display Forwarding (*Activo*)
 - *La etapa origen como la etapa destino del adelantamiento de datos son unidas con una flecha verde en el diagrama de ciclos de reloj.*
- Display Cause of Stalls (*Activo*)
 - *Si esta opción está activa, la instrucción que causa una detención por riegos de datos (RAW o WAW) es marcada con una flecha roja.*



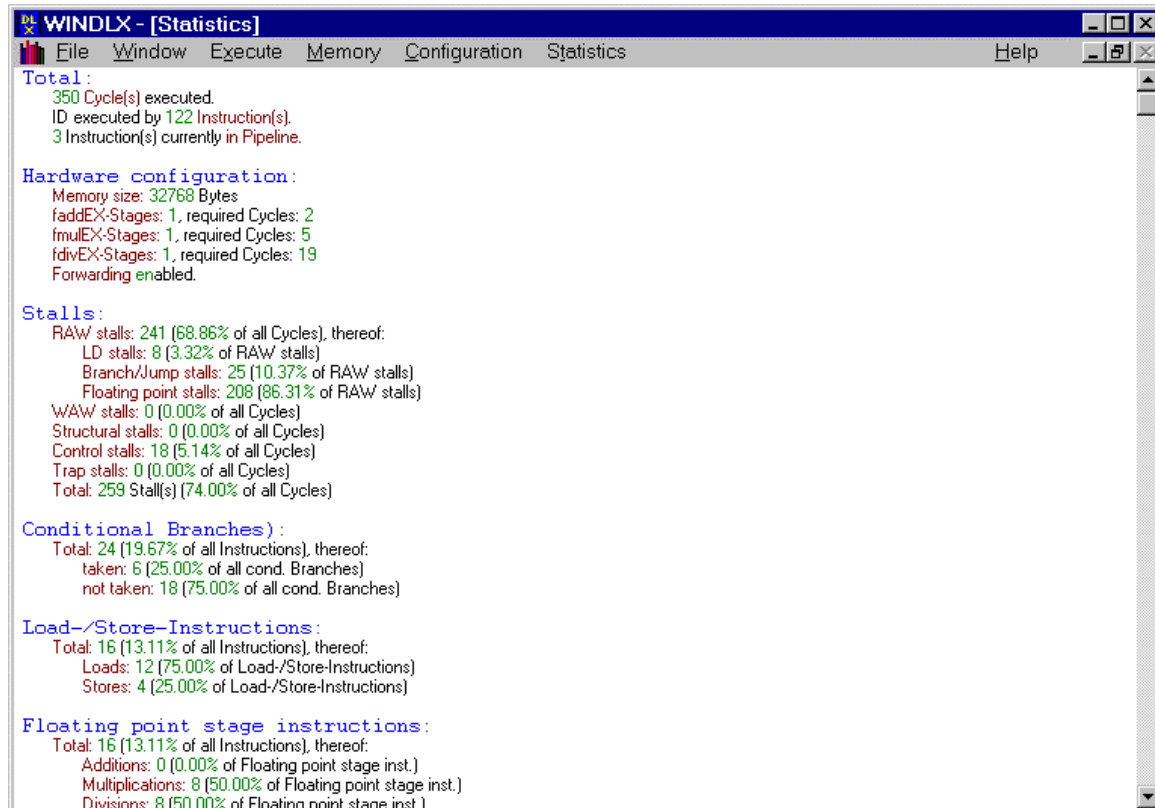
Interfaz de WinDLX

Ventana y menú Statistics

La ventana *Statistics* es utilizada para visualizar estadísticas sobre la simulación que está siendo realizada.

Los datos son organizados en los siguiente grupos:

- *Total*
- *Hardware configuration*
- *Stalls*
- *Conditional Branches.*
- *Load/Store-Instructions*
- *Floating point stages instructions*
- *Traps*



```
WINDLX - [Statistics]
File Window Execute Memory Configuration Statistics Help

Total:
350 Cycle(s) executed.
ID executed by 122 Instruction(s).
3 Instruction(s) currently in Pipeline.

Hardware configuration:
Memory size: 32768 Bytes
faddEX-Stages: 1, required Cycles: 2
fmulEX-Stages: 1, required Cycles: 5
fdvEX-Stages: 1, required Cycles: 19
Forwarding enabled.

Stalls:
RAW stalls: 241 (68.86% of all Cycles), thereof:
  LD stalls: 8 (3.32% of RAW stalls)
  Branch/Jump stalls: 25 (10.37% of RAW stalls)
  Floating point stalls: 208 (86.31% of RAW stalls)
WAW stalls: 0 (0.00% of all Cycles)
Structural stalls: 0 (0.00% of all Cycles)
Control stalls: 18 (5.14% of all Cycles)
Trap stalls: 0 (0.00% of all Cycles)
Total: 259 Stall(s) (74.00% of all Cycles)

Conditional Branches):
Total: 24 (19.67% of all Instructions), thereof:
  taken: 6 (25.00% of all cond. Branches)
  not taken: 18 (75.00% of all cond. Branches)

Load-/Store-Instructions:
Total: 16 (13.11% of all Instructions), thereof:
  Loads: 12 (75.00% of Load-/Store-Instructions)
  Stores: 4 (25.00% of Load-/Store-Instructions)

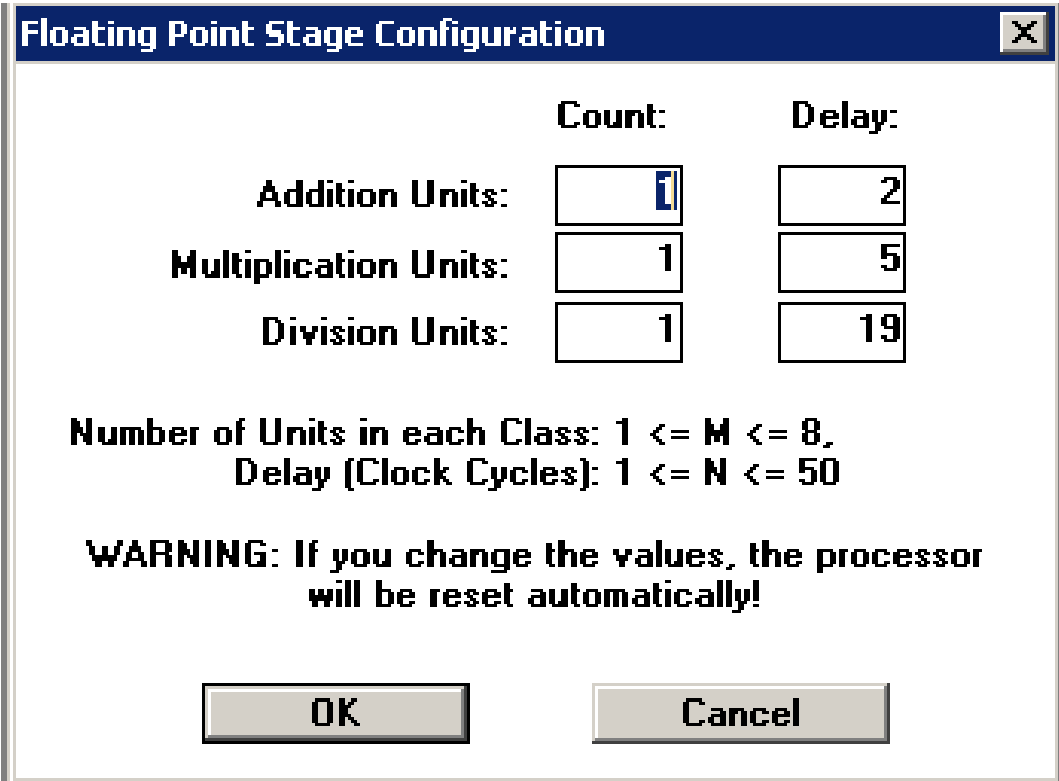
Floating point stage instructions:
Total: 16 (13.11% of all Instructions), thereof:
  Additions: 0 (0.00% of Floating point stage inst.)
  Multiplications: 8 (50.00% of Floating point stage inst.)
  Divisions: 8 (50.00% of Floating point stage inst.)
```

Bloque II – Procesadores Segmentados

GUION 4: RIESGOS DE DATOS Y ESTRUCTURALES

Procesadores Segmentados

Configuración de unidades de punto flotante



The image shows a Windows-style dialog box titled "Floating Point Stage Configuration". It contains three rows of configuration options, each with a label, a "Count" input field, and a "Delay" input field. The "Count" fields have values 1, 1, and 1 respectively. The "Delay" fields have values 2, 5, and 19 respectively. Below these fields, there is a warning message and two buttons: "OK" and "Cancel".

	Count:	Delay:
Addition Units:	1	2
Multiplication Units:	1	5
Division Units:	1	19

Number of Units in each Class: $1 \leq M \leq 8$.
Delay (Clock Cycles): $1 \leq N \leq 50$

WARNING: If you change the values, the processor will be reset automatically!

OK Cancel

Guion 1. Riesgos de datos y estructurales

1. Realizar un programa donde:
 - Estén cargados 20 números de doble precisión
 - 3,2,1,4,8, 9,2,7,4,5, 3,2,8,4,5, 3,2,6,4,5
 - Calcular el mínimo de todos ellos en el registro F6
 - Ganancia
 - Número de ciclos en un procesador sin segmentar
 - Número de ciclos en un procesador segmentado
 - Con bypass
 - Sin bypass
 - Eliminar detenciones → Reordenando instrucciones
 - Número de ciclos
 - Ganancia frente a procesador sin segmentar
 - Ganancia frente a procesador segmentado con caminos de bypass

Guion 1. Riesgos de datos y estructurales

2. Realizar un programa donde:

- Estén cargados 8 números enteros en memoria (posición 0000):
1,2,3,4,5,6,7,8
- Calcule la multiplicación de los números que ocupan la posición par (384) → R10
- Calcule la multiplicación de los números que ocupan la posición impar (105) → R12,
- Número de ciclos considerando bypass
- Eliminar detenciones, usando reordenando instrucciones y considerando bypass
- Número de ciclos
- Informe
 - Detenciones debidas a riesgos de datos
 - Detenciones debidas a riesgos estructurales
 - Informe de eliminación de detenciones

Guion 1. Riesgos de datos y estructurales

3. Realizar un programa donde:

- Se almacena la siguiente matriz de elementos de *double*

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- Almacenar la suma de cada columna (f10-f12-f14-f16)
 - Número de ciclos
- Eliminar detenciones, usando reordenando instrucciones y considerando bypass
 - Número de ciclos
- Informe
 - Detenciones debidas a riesgos de datos
 - Detenciones debidas a riesgos estructurales
 - Informe de eliminación de detenciones

Bloque II – Procesadores Segmentados

GUION 2: RIESGOS DE DATOS Y DE CONTROL

Guion 2. Riesgos de datos y de control

1. Realizar un programa donde:

- a) Estén cargados 12 números enteros en memoria (posición 0000): 1,2,3,4, 1,2,3,4, 1,2,3,4
- b) Calcule la multiplicación de los números que ocupan la posición par → R10 y calcule la multiplicación de los números que ocupan la posición impar → R12, usando reordenamiento de instrucciones y considerando bypass.
- c) Informe
 - Número de ciclos
 - Detenciones debidas a riesgos de datos
 - Detenciones debidas a riesgos control
- d) Realizar distintas versiones para aplicar la técnica de desenrollado de bucles donde en una iteración se realicen 2,3 y 4 operaciones.
 - Calcular número de ciclos para cada versión
 - Calcular la ganancia de cada versión del apartado d con la versión del programa realizada en c
 - Informe de eliminación de detenciones debidas a riesgos de control

Guion 2. Riesgos de datos y de control

2. Considerando el programa del guion 1, realizar distintas versiones para aplicar la técnica de reordenamiento de instrucciones para las versiones del apartado c.
 - Calcular número de ciclos para cada versión
 - Calcular la ganancia de cada versión del apartado d con la versión del programa realizada en c
 - Informe de eliminación de Detenciones debidas a Riesgos de Datos

Simulación de Validación Bloque II

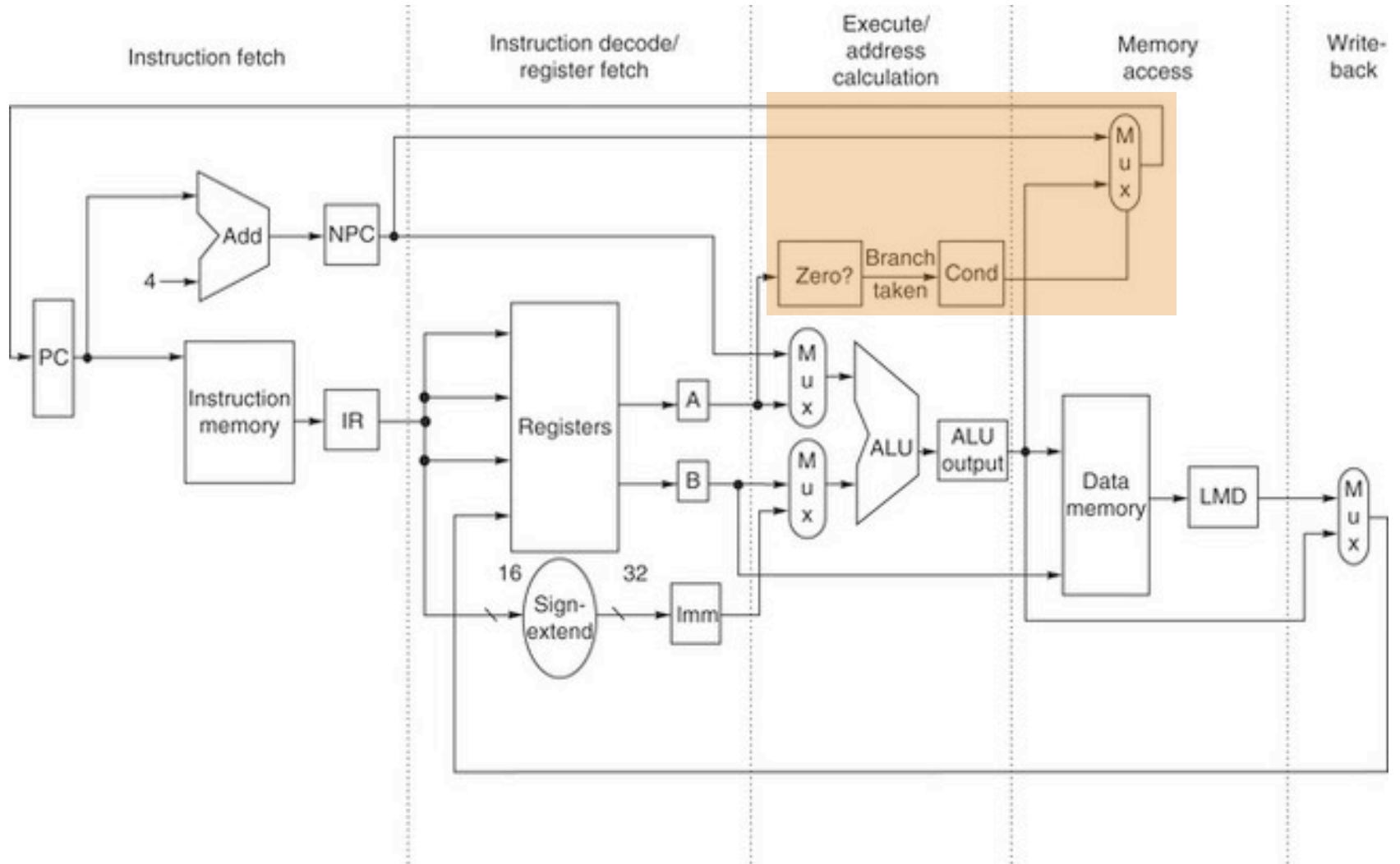
```
.data 0  
N:    .word 3,6,9,2,6,9,1,6,3,9
```

```
.text 256  
addi R7, R0, N  
addi R3, R0, #10  
  
bucle:  
    lw R1, 0(R7)  
    add R2, R1, R1  
    sw 0(R7), R2  
  
    addi R7, R7, #4  
    subi R3, R3, #1  
    bnez R3, bucle
```

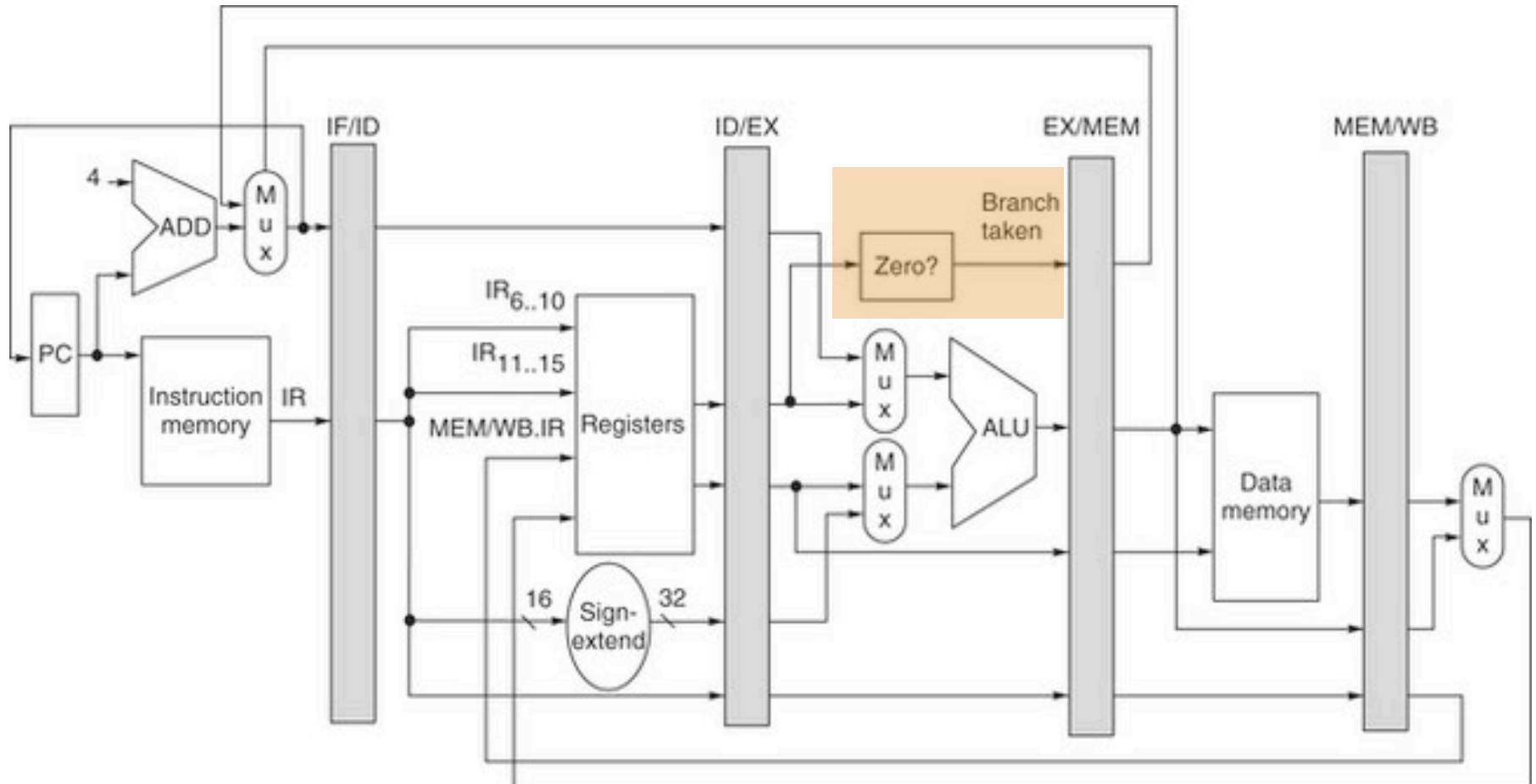

Bloque II – Procesadores Segmentados

CONFIGURACIÓN DE SALTOS Y TÉCNICAS
PRÁCTICAS

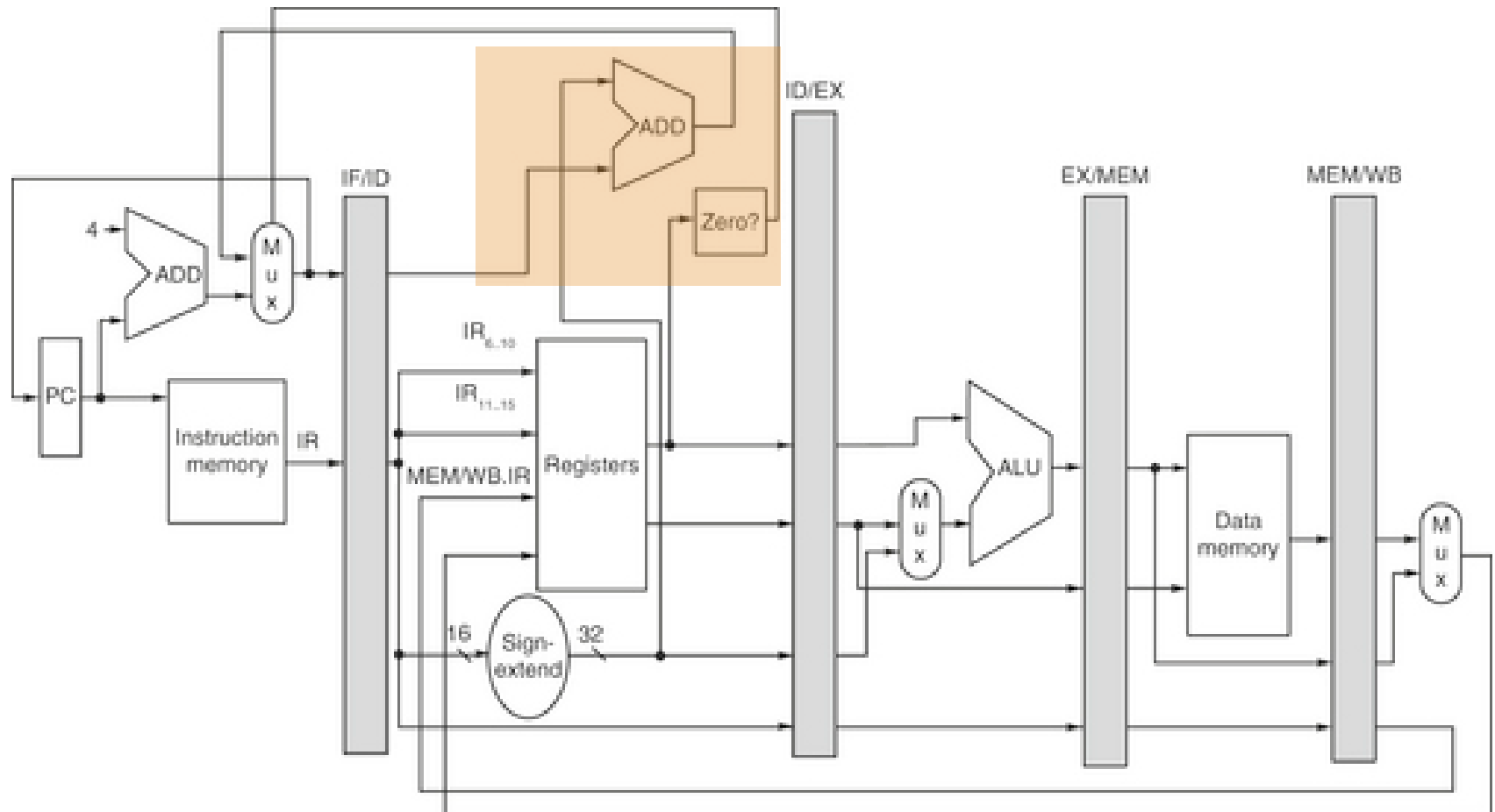
DLX - Secuencial



DLX – Segmentación de las etapas



DLX – Adelantamiento de saltos



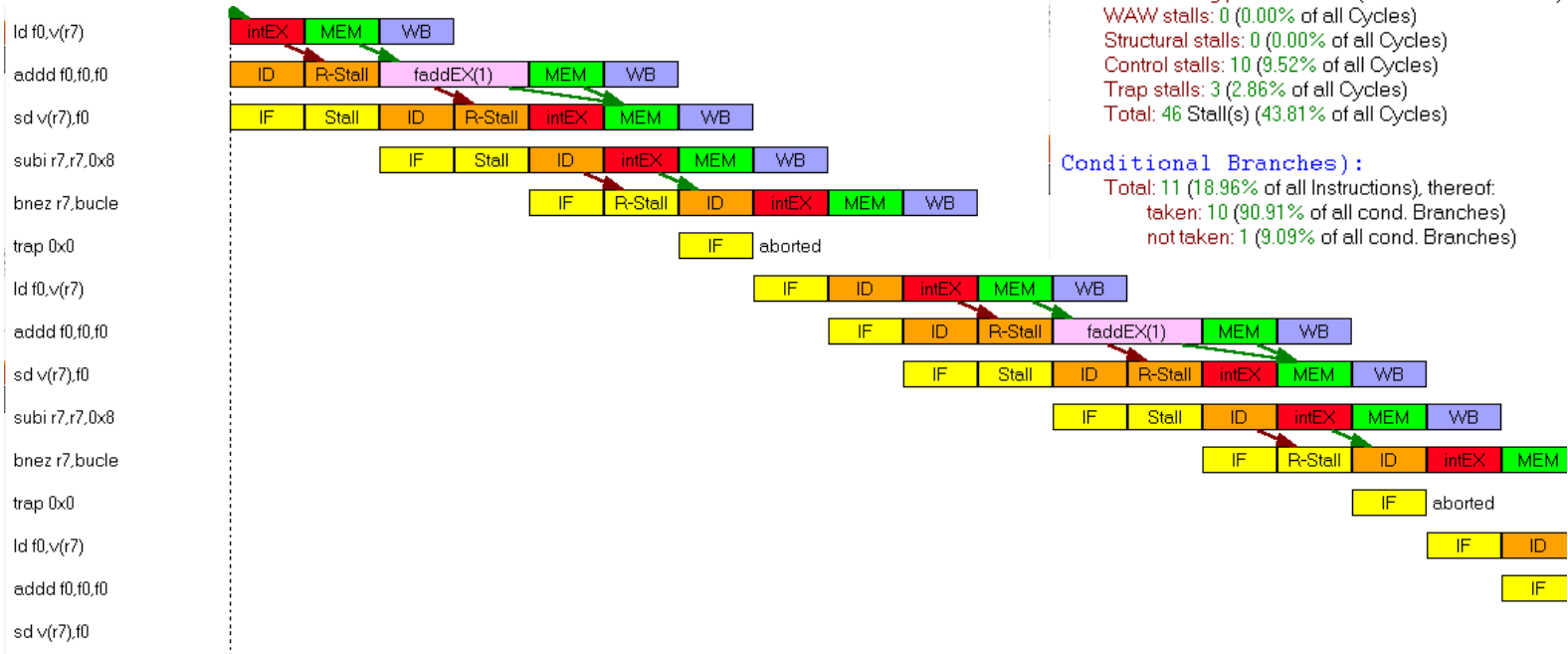
Riesgos de datos y de control – Situación inicial

bucle:

```
ld f0,-8(r7)
add f0,f0,f0
sd -8(r7),f0
```

```
subi r7,r7,#8
bnez r7,bucle
```

trap #0



Total:

105 Cycle(s) executed.

ID executed by 58 Instruction(s).

2 Instruction(s) currently in Pipeline.

Hardware configuration:

Memory size: 32768 Bytes

faddEX-Stages: 1, required Cycles: 2

fmulEX-Stages: 1, required Cycles: 5

fdivEX-Stages: 1, required Cycles: 19

Forwarding enabled.

Stalls:

RAW stalls: 33 (31.43% of all Cycles), thereof:

LD stalls: 11 (33.33% of RAW stalls)

Branch/Jump stalls: 11 (33.33% of RAW stalls)

Floating point stalls: 11 (33.33% of RAW stalls)

WAW stalls: 0 (0.00% of all Cycles)

Structural stalls: 0 (0.00% of all Cycles)

Control stalls: 10 (9.52% of all Cycles)

Trap stalls: 3 (2.86% of all Cycles)

Total: 46 Stall(s) (43.81% of all Cycles)

Conditional Branches):

Total: 11 (18.96% of all Instructions), thereof:

taken: 10 (90.91% of all cond. Branches)

not taken: 1 (9.09% of all cond. Branches)

Riesgos de datos y de control - Desenrollado

bucle:

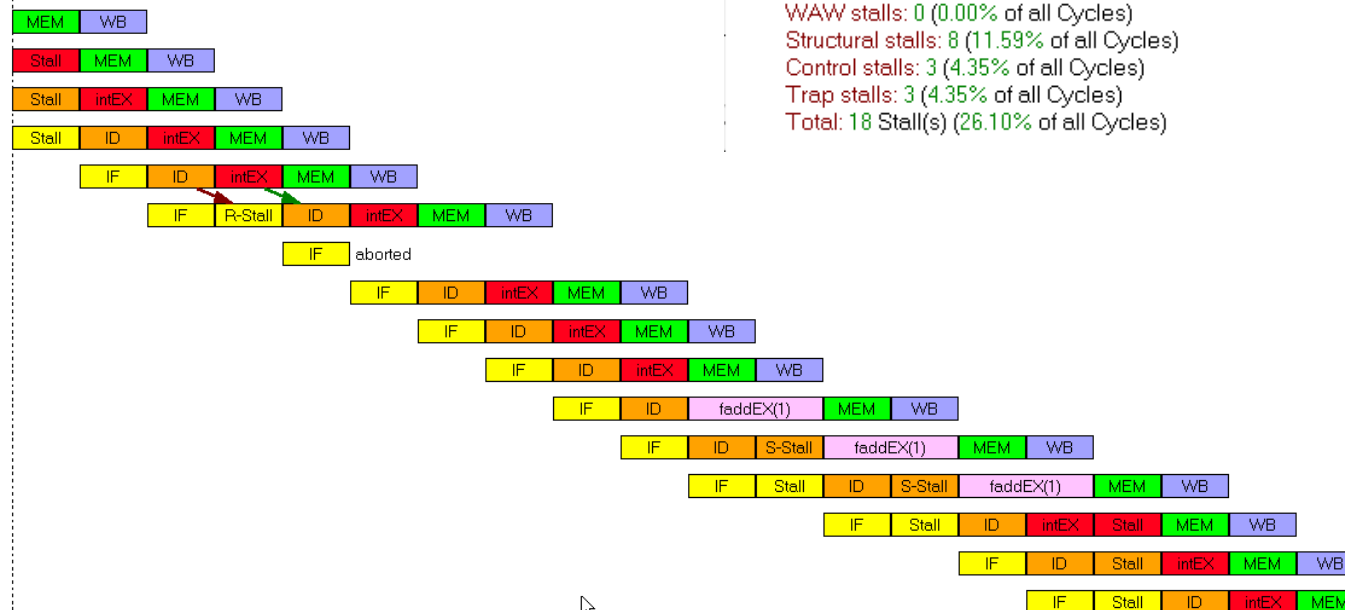
```
ld f0,-8(r7)
ld f2,-16(r7)
ld f4,1-24(r7)
```

```
addd f0,f0,f0
addd f2,f2,f2
addd f4,f4,f4
```

```
sd -8(r7),f0
sd -16(r7),f2
sd -24(r7),f4
```

```
subi r7,r7,#24
bnez r7,bucle
```

```
addd f4,f4,f4
sd 0xfff0(r7),f0
sd 0xfff0(r7),f2
sd 0xffe8(r7),f4
subi r7,r7,0x18
bnez r7,bucle
trap 0x0
ld f0,0xfff8(r7)
ld f2,0xfff0(r7)
ld f4,0xffe8(r7)
addd f0,f0,f0
addd f2,f2,f2
addd f4,f4,f4
sd 0xfff8(r7),f0
sd 0xfff0(r7),f2
sd 0xffe8(r7),f4
```



Total:

69 Cycle(s) executed.
ID executed by 46 Instruction(s).
2 Instruction(s) currently in Pipeline.

Hardware configuration:

Memory size: 32768 Bytes
faddEX-Stages: 1, required Cycles: 2
fmulEX-Stages: 1, required Cycles: 5
fdivEX-Stages: 1, required Cycles: 19
Forwarding enabled.

Stalls:

RAW stalls: 4 (5.80% of all Cycles), thereof:
LD stalls: 0 (0.00% of RAW stalls)
Branch/Jump stalls: 4 (100.00% of RAW stalls)
Floating point stalls: 0 (0.00% of RAW stalls)
WAW stalls: 0 (0.00% of all Cycles)
Structural stalls: 8 (11.59% of all Cycles)
Control stalls: 3 (4.35% of all Cycles)
Trap stalls: 3 (4.35% of all Cycles)
Total: 18 Stall(s) (26.10% of all Cycles)

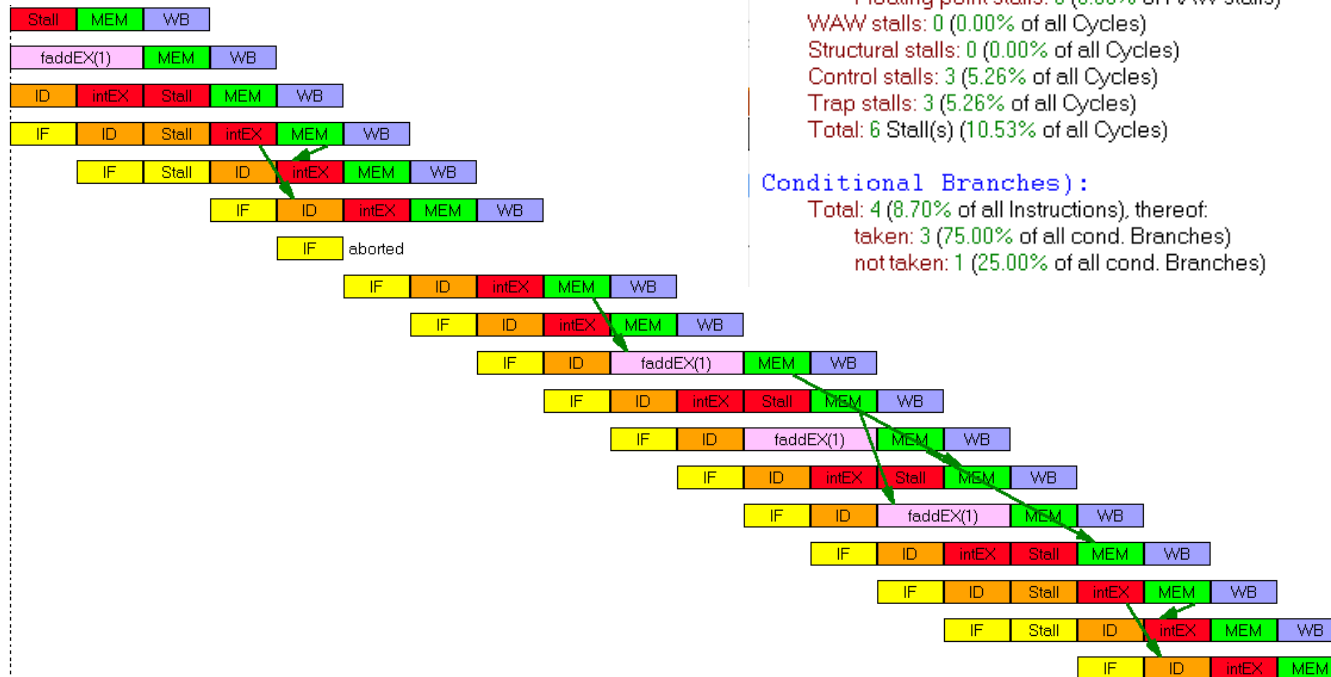
Riesgos de datos y de control – Desenrollado + Reordenado

bucle:

```
ld f0,-8(r7)
ld f2,-16(r7)
add f0,f0,f0
ld f4,-24(r7)
add f2,f2,f2
sd -8(r7),f0
add f4,f4,f4
sd -16(r7),f2
subi r7,r7,#24
sd 0(r7),f4
```

bnez r7,bucle

```
sd 0xffff(r7),f0
add f4,f4,f4
sd 0xffff(r7),f2
subi r7,r7,0x18
sd v(r7),f4
bnez r7,bucle
trap 0x0
ld f0,0xffff(r7)
ld f2,0xffff(r7)
add f0,f0,f0
ld f4,0xffe8(r7)
add f2,f2,f2
sd 0xffff(r7),f0
add f4,f4,f4
sd 0xffff(r7),f2
subi r7,r7,0x18
sd v(r7),f4
bnez r7,bucle
```



Total:

57 Cycle(s) executed.
ID executed by 46 Instruction(s).
2 Instruction(s) currently in Pipeline.

Hardware configuration:

Memory size: 32768 Bytes
faddEX-Stages: 1, required Cycles: 2
fmulEX-Stages: 1, required Cycles: 5
fdivEX-Stages: 1, required Cycles: 19
Forwarding enabled.

Stalls:

RAW stalls: 0 (0.00% of all Cycles), thereof:
LD stalls: 0 (0.00% of RAW stalls)
Branch/Jump stalls: 0 (0.00% of RAW stalls)
Floating point stalls: 0 (0.00% of RAW stalls)
WAW stalls: 0 (0.00% of all Cycles)
Structural stalls: 0 (0.00% of all Cycles)
Control stalls: 3 (5.26% of all Cycles)
Trap stalls: 3 (5.26% of all Cycles)
Total: 6 Stall(s) (10.53% of all Cycles)

Conditional Branches):

Total: 4 (8.70% of all Instructions), thereof:
taken: 3 (75.00% of all cond. Branches)
not taken: 1 (25.00% of all cond. Branches)

Desenrollado de bucles al compilar

Sin optimización

```
#include <stdio.h>

int main() {
    int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int suma = 0, i;

    for(i = 0; i < 10; i++) {
        suma += v[i];
    }

    printf("%d\n", suma);
}
```

Con optimización

```
00: 00
int suma = 0, i;

for(i = 0; i < 10; i++) {
67: 48 89 e0          mov     rax,rsip
6a: 48 8d 4c 24 28    lea     rcx,[rsi+0x28]
int suma = 0, i;
6f: ba 00 00 00 00    mov     edx,0x0
    suma += v[i];
74: 03 10            add     edx,DWORD PTR [rax]
for(i = 0; i < 10; i++) {
76: 48 83 c0 04        add     rax,0x4
7a: 48 39 c8          cmp     rax,rcx
7d: 75 f5            jne     74 <main+0x74>
}
```

```
for(i = 0; i < 10; i++) {
67: 48 89 e0          mov     rax,rsip
6a: 48 8d 4c 24 28    lea     rcx,[rsi+0x28]
int suma = 0, i;
6f: ba 00 00 00 00    mov     edx,0x0
    suma += v[i];
74: 03 10            add     edx,DWORD PTR [rax]
76: 03 50 04          add     edx,DWORD PTR [rax+0x4]
79: 03 50 08          add     edx,DWORD PTR [rax+0x8]
7c: 03 50 0c          add     edx,DWORD PTR [rax+0xc]
7f: 03 50 10          add     edx,DWORD PTR [rax+0x10]
for(i = 0; i < 10; i++) {
82: 48 83 c0 14        add     rax,0x14
86: 48 39 c8          cmp     rax,rcx
89: 75 e9            jne     74 <main+0x74>
}
```