



Prácticas Arquitectura de Computadores



Bloque II

Cronograma

- ▶ **Bloque II – Procesadores Segmentados**

- ▶ **Riesgos**

- ▶ **Riesgos de Datos**
 - Adelantamientos
 - Reordenamiento de código
 - ▶ **Riesgos Estructurales**
 - ▶ **Riesgos de Control**
 - Desenrollado de Bucles
-

- ▶ **Guión 4: Riesgos de Datos y Estructurales**

- ▶ Dependencias de datos
 - ▶ Adelantamientos
 - ▶ Reordenamiento
 - ▶ Ganancia

- ▶ **Guión 5: Riesgos de Control**

- ▶ Estudio del Flujo del programa
 - ▶ Riesgos de control
 - ▶ Desenrollado de Bucles
 - ▶ Ganancia

- ▶ **Guión 6: Riesgos de Datos y de Control**

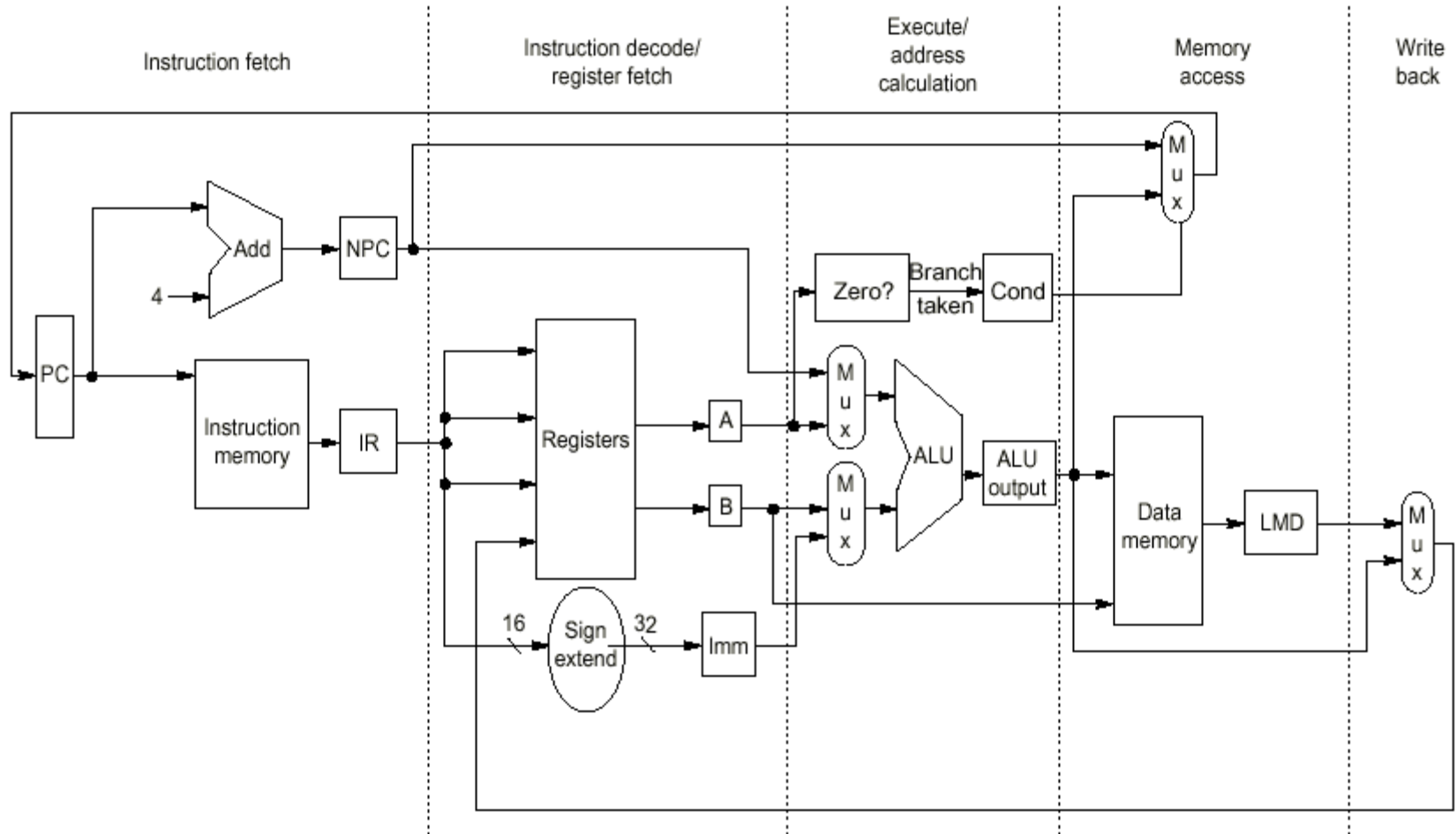
- ▶ Desenrollado de Bucles y Reordenamiento de instrucciones
 - ▶ Ganancia

- ▶ **Prueba de validación – 7/11/2017**

Bloque II – Procesadores Segmentados

Riesgos de Datos y Estructurales

Ruta de datos del DLX



DLX: Etapas en la ejecución de instrucciones

- ▶ Toda instrucción puede ser ejecutada en 5 ciclos de reloj, siguiendo estas etapas:
 - ▶ IF: Búsqueda de la instrucción
 - ▶ ID: Decodificación
 - ▶ EX: Ejecución
 - ▶ MEM: Acceso a memoria
 - ▶ WB: Escritura de resultados



Etapa IF

- ▶ Se carga la instrucción en curso en el *Registro de Instrucción* (RI)
- ▶ (IMAR) Instruction Memory Address Register.
- ▶ Se prepara el contador de programa para apuntar a la siguiente instrucción.

$$IMAR \leftarrow PC$$

$$IR \leftarrow Mem[PC]$$

$$NPC \leftarrow PC + 4$$



Etapa ID

- ▶ Se decodifica la instrucción
- ▶ Se cargan los diferentes campos en los registros especiales A, B, Imm.
- ▶ Saltos condicionales son calculados en esta etapa con el fin de reducir los riesgos de control

$$A \leftarrow \text{Regs}[\text{IR}6 \dots 10]$$

$$B \leftarrow \text{Regs}[\text{IR}11 \dots 15]$$

$$\text{Imm} \leftarrow ((\text{IR}16)16 \text{ \#\# } \text{IR}16 \dots 31)$$



Etapa EX

- ▶ Acceso a memoria (cálculo de la dirección efectiva)

$$ALUOutput \leftarrow A + Imm$$

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

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

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

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

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

$$ALUOutput \leftarrow NPC + Imm$$

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



Etapa MEM

- ▶ Acceso a memoria (lectura-Load o escritura-Store en memoria)

$$LMD \leftarrow Mem[ALUOutput]$$
$$Mem[ALUOutput] \leftarrow B$$

- ▶ Salto (carga de la dirección efectiva en el PC)

if (cond)

$$PC \leftarrow ALUOutput$$

else

$$PC \leftarrow NPC$$


Etapas WB

- ▶ Reg-Reg ALU:

$Regs[IR / 6 \dots 20] \leftarrow ALUOutput$

- ▶ Reg-Imm ALU:

$Regs[IR / 1 \dots 15] \leftarrow ALUOutput$

- ▶ Instrucción de carga (Load):

$Regs[IR / 1 \dots 15] \leftarrow LMD$



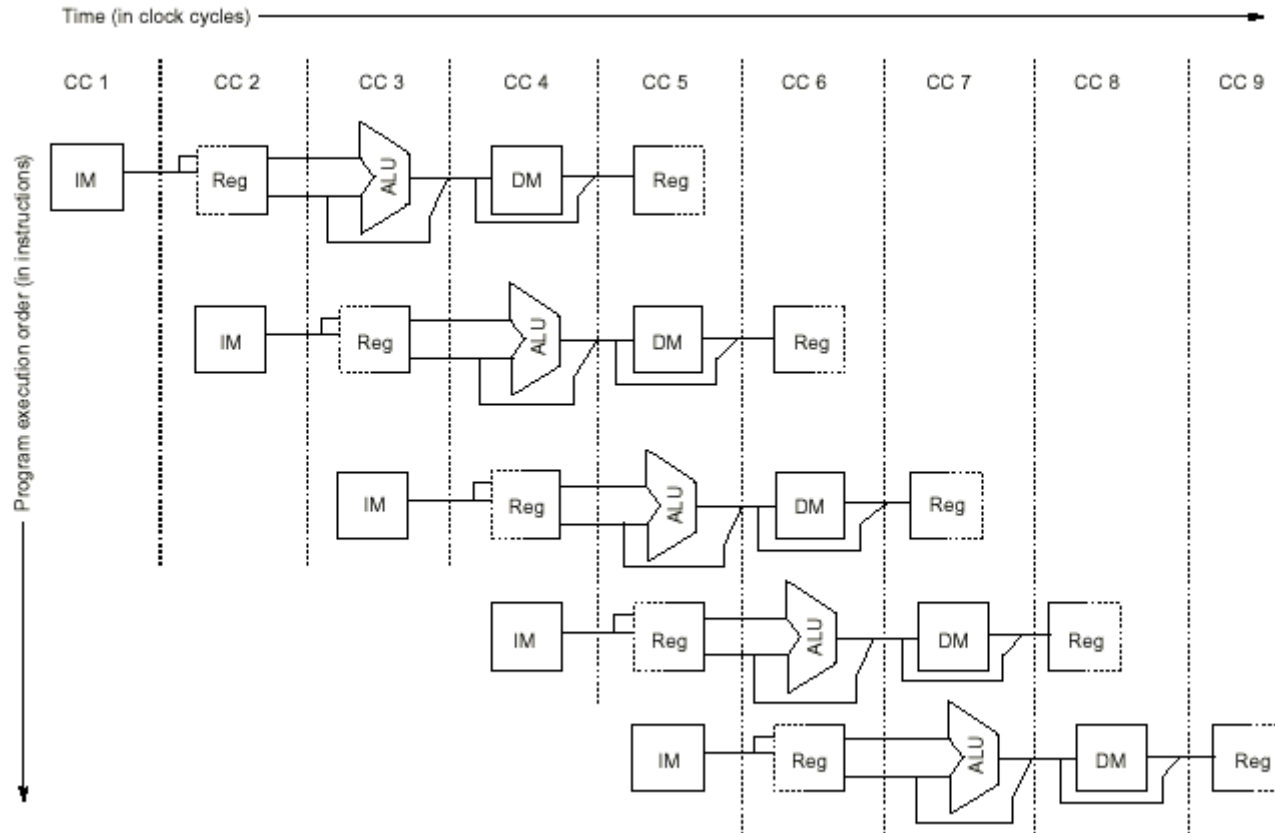
Segmentación a nivel de Instrucción

- ▶ ¿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 de ID y WB acceden al banco de registros simultáneamente.
- ▶ 2: Las etapas de IF y MEM acceden a la memoria simultáneamente.

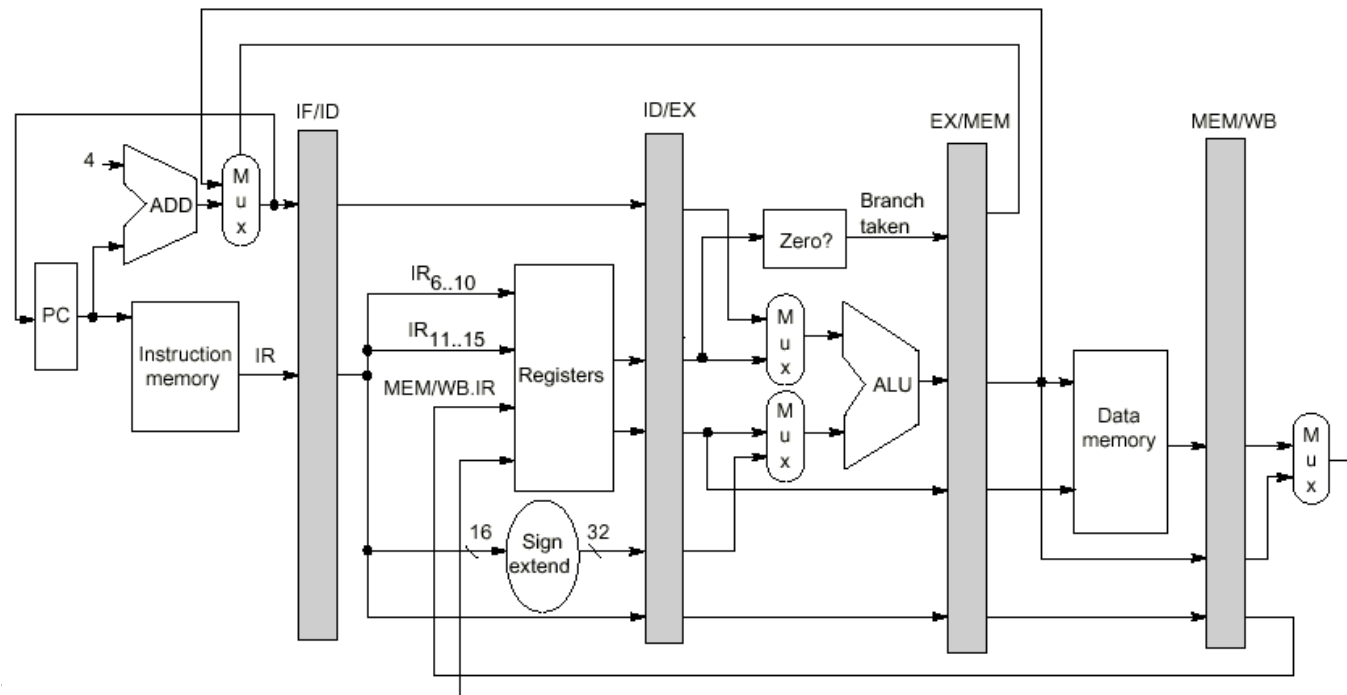
SOLUCIONES

- ▶ 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 para el 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

- ▶ Riesgos (*Hazards*): Situaciones que imposibilitan que la próxima instrucción se ejecute en el ciclo predeterminado.
- ▶ Los riesgos reducen el rendimiento ideal de la máquina (hacen que $CPI > 1$).
- ▶ Tipos:
 - ▶ Riesgos Estructurales
 - ▶ Riesgos de Datos
 - ▶ Riesgos de Control

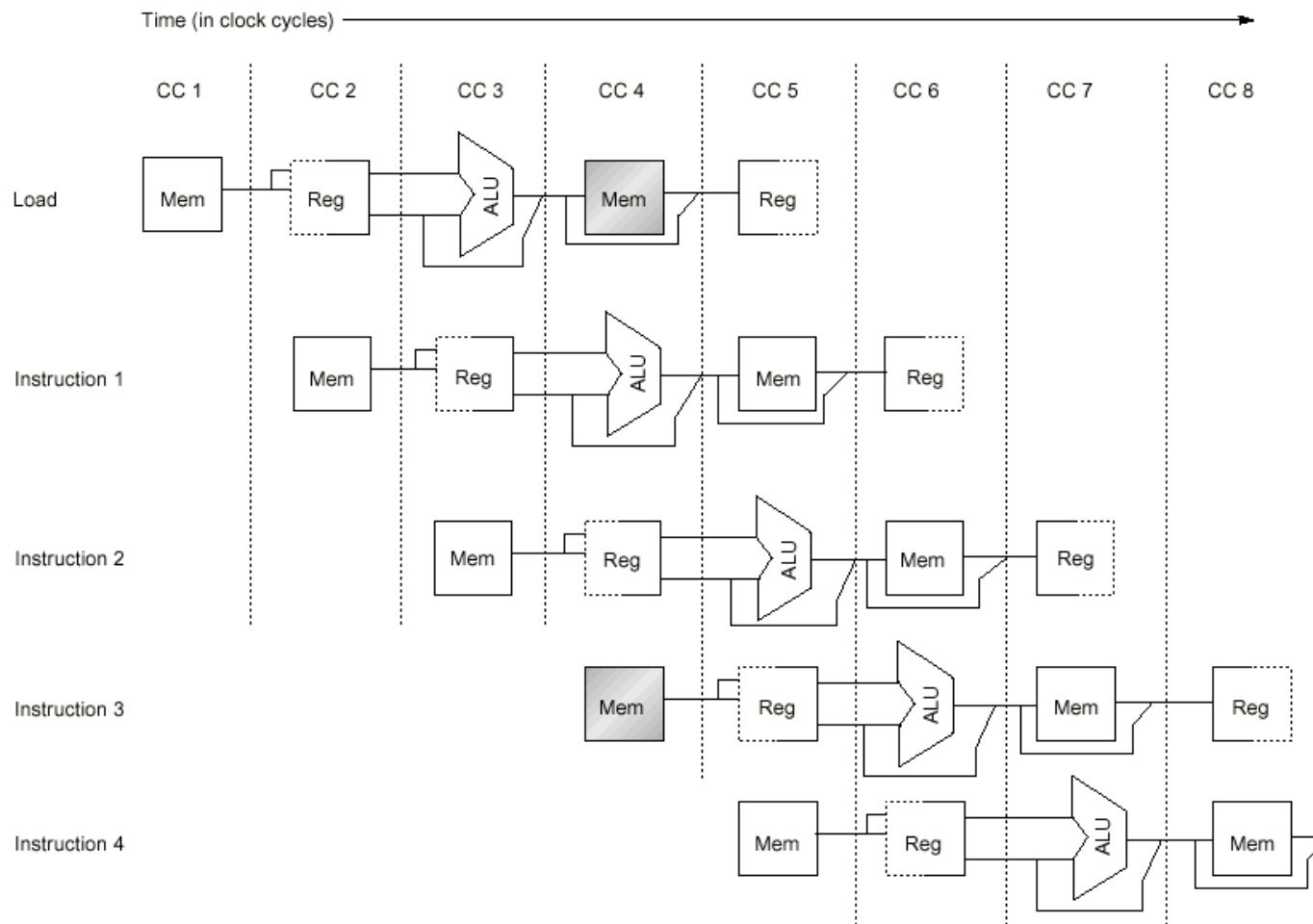


Riesgos Estructurales

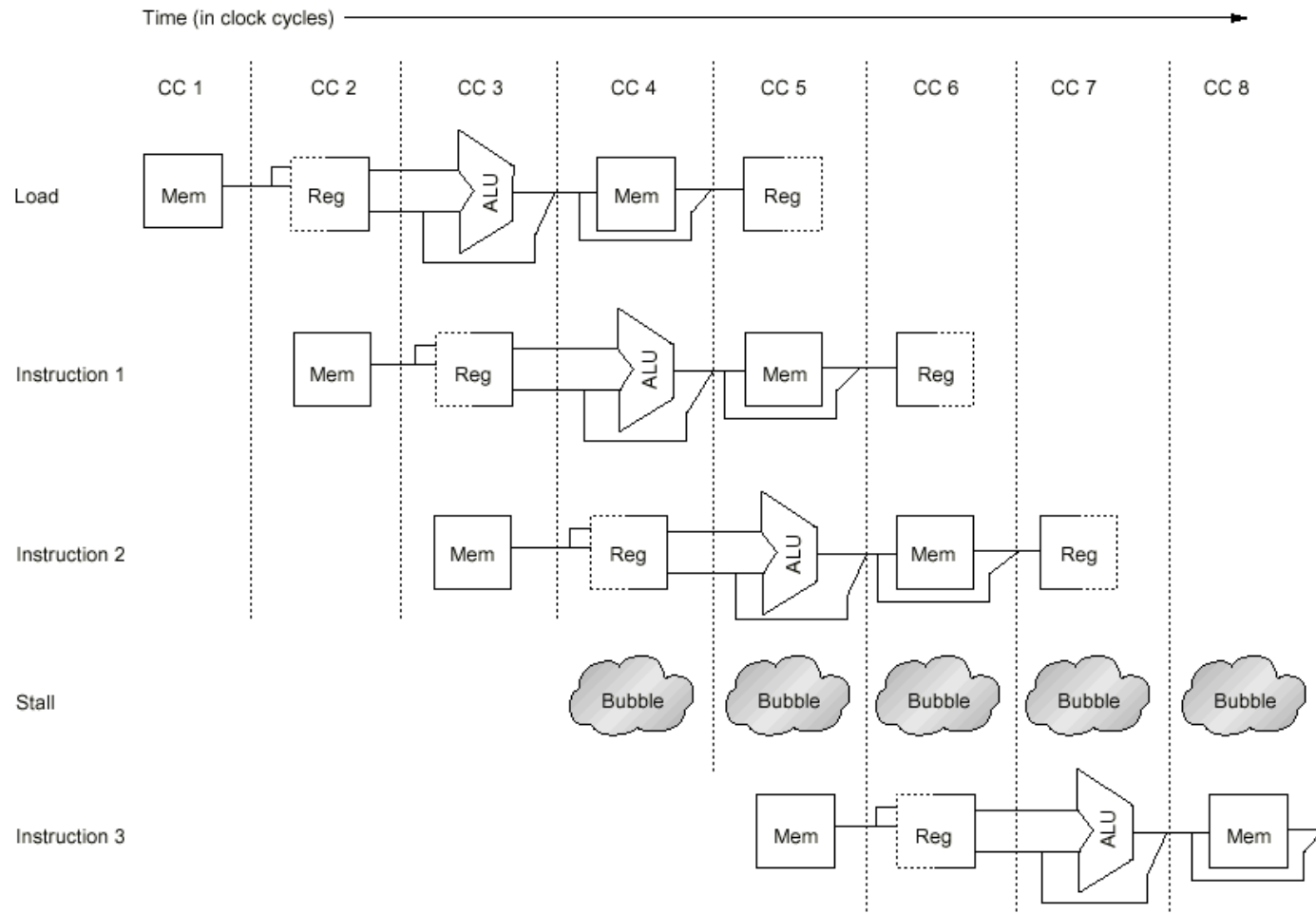
- ▶ Situación en la que dos o más instrucciones tratan de hacer uso de un único recurso.
- ▶ Casos más habituales:
 - ▶ Máquinas con una sola memoria (conflictos en lectura de datos e instrucciones)
 - ▶ Unidades funcionales multi-ciclo no segmentadas
- ▶ Solución al riesgo estructural: Parada de la unidad durante una etapa (introducción de una burbuja) ⇒ Reducción del rendimiento



Riesgos Estructurales



Riesgos Estructurales



Riesgos Estructurales

Instruction	Clock cycle number									
	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 (Lectura después de escritura):
 - ▶ También conocida como “Dependencia”.
 - ▶ WAR (Escritura después de lectura):
 - ▶ También conocida como “Anti-Dependencia”.
 - ▶ WAW (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.

ADD R1,R2,R3	IF	ID	EX	MEM	WB
SUB R4,R1,R5	IF	ID	EX	MEM	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

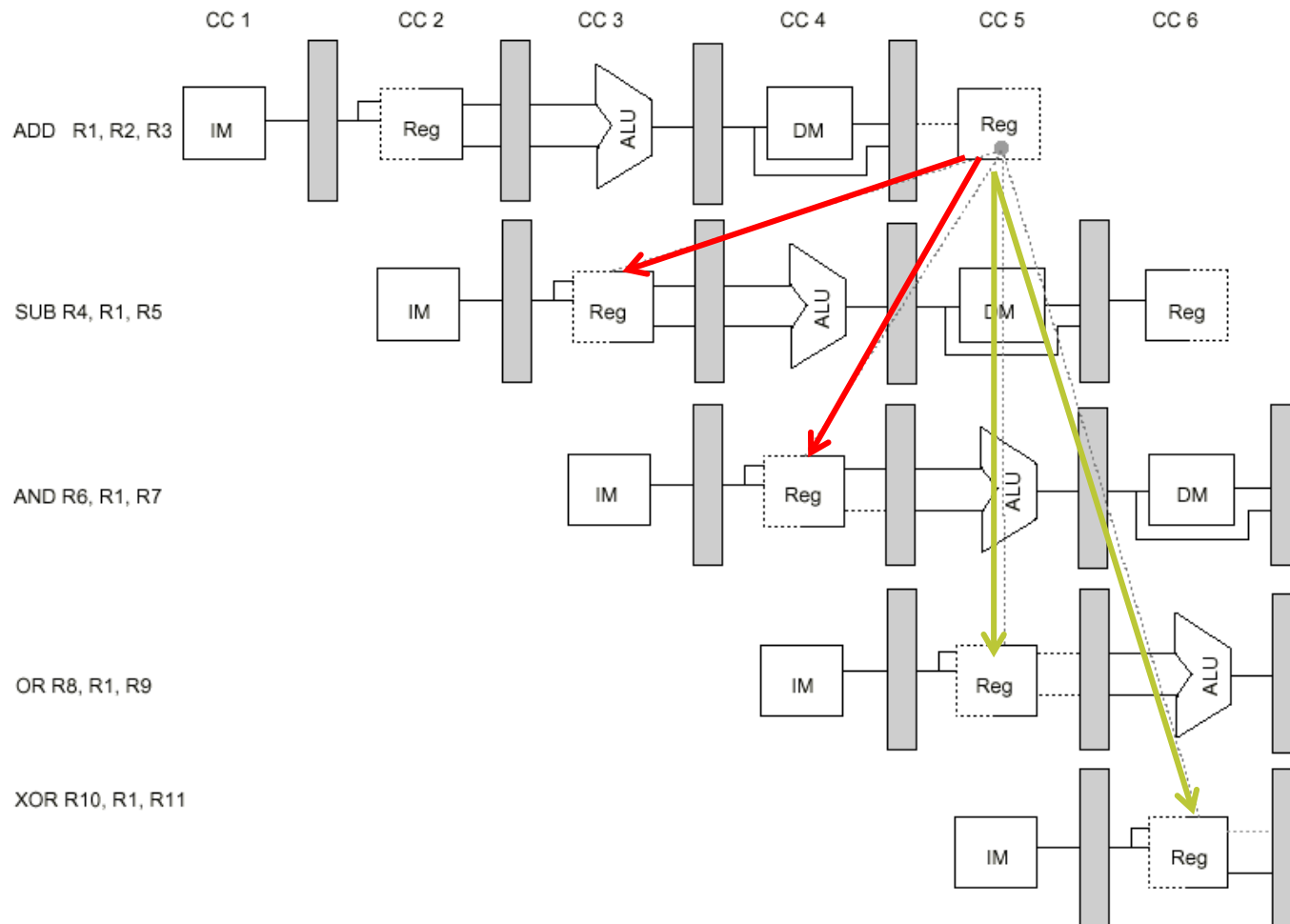
AND R6,R1,R7

OR R8,R1,R9

XOR R10,R1,R11



Ejemplo de Dependencia de Datos



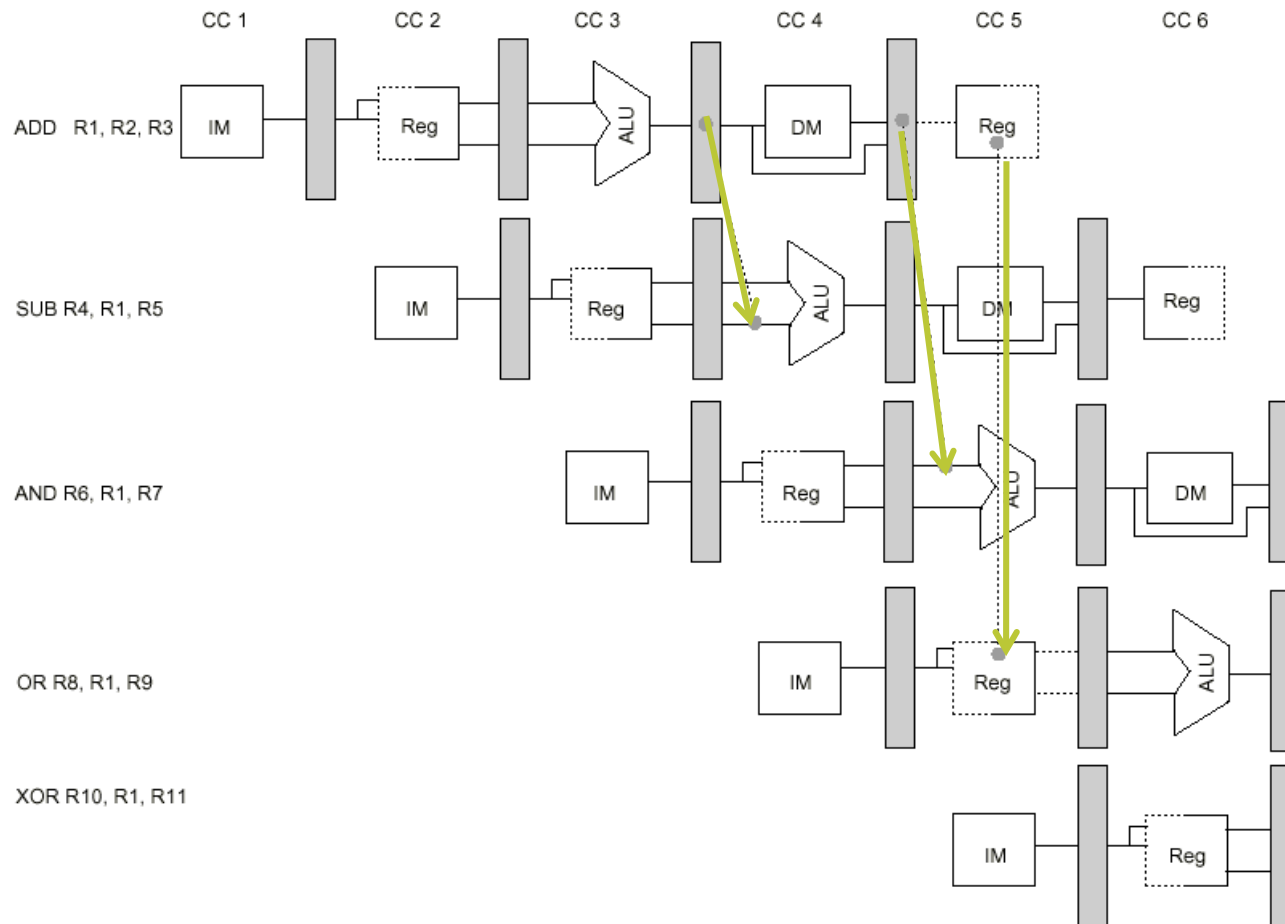
Anticipación de datos

- ▶ **Solución:**

- ▶ Anticipación de datos (*Forwarding*), también conocido como Adelantamiento.
- ▶ Los datos se adelantan 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

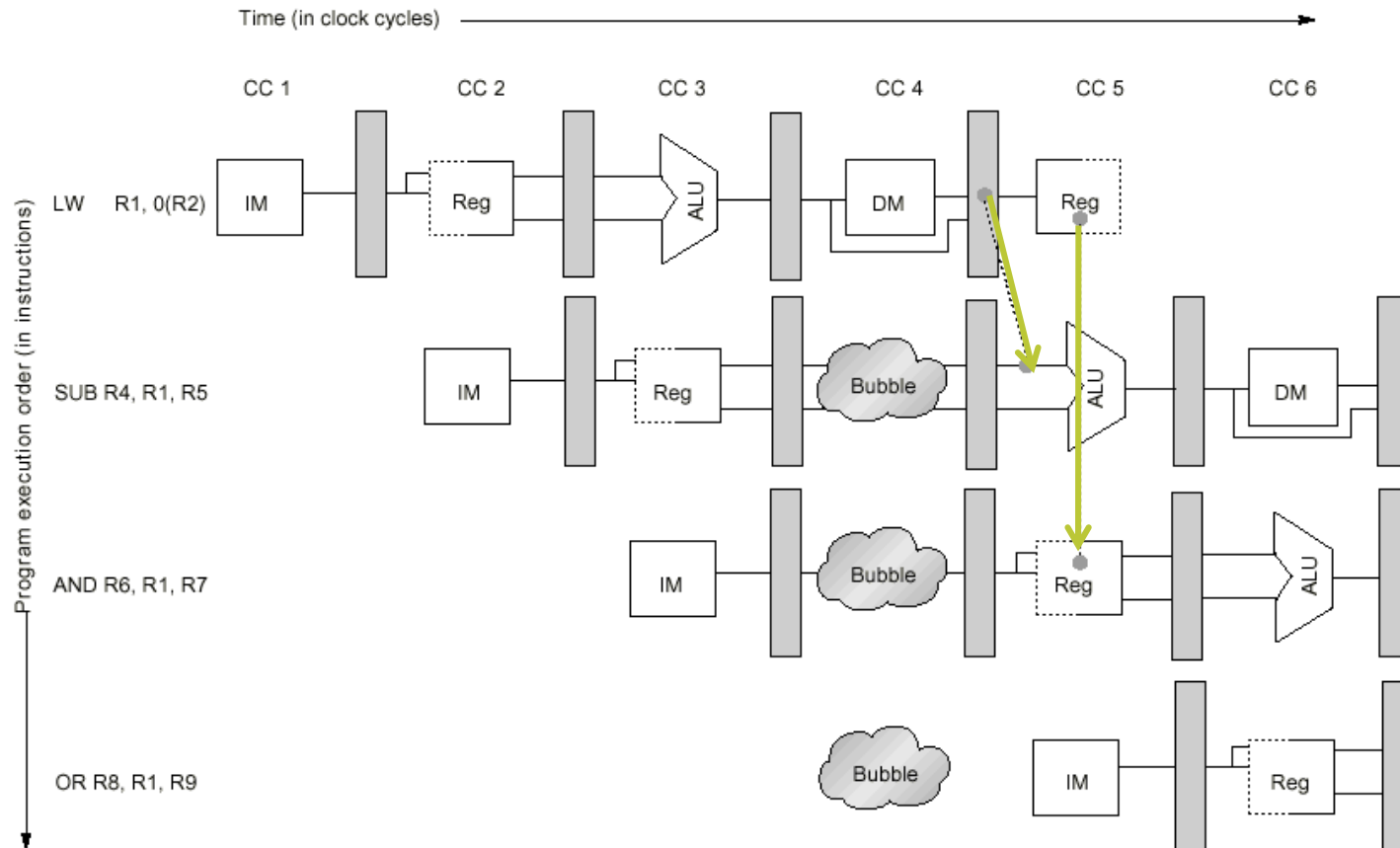
AND R6,R1,R7

OR R8,R1,R9

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

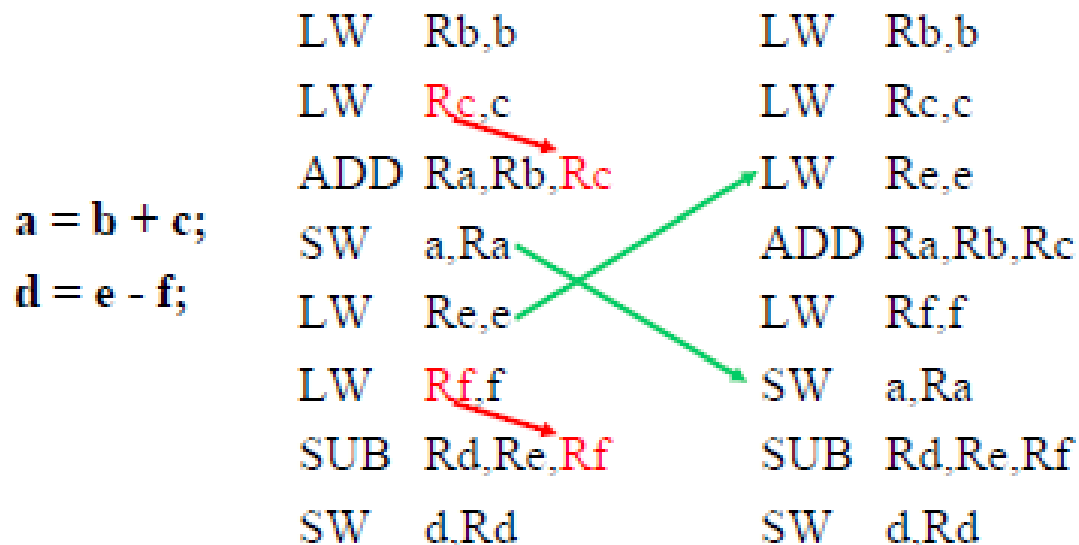


Riesgos de datos inevitables



Planificación estática: Cambio de orden

- ▶ El compilador cambia de orden las instrucciones:
- ▶ Estático, antes de la ejecución.



Planificación estática: Renombramiento

► Dependencias RAW:

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

LW R1,0(R2)

ADD R3,R1,R4

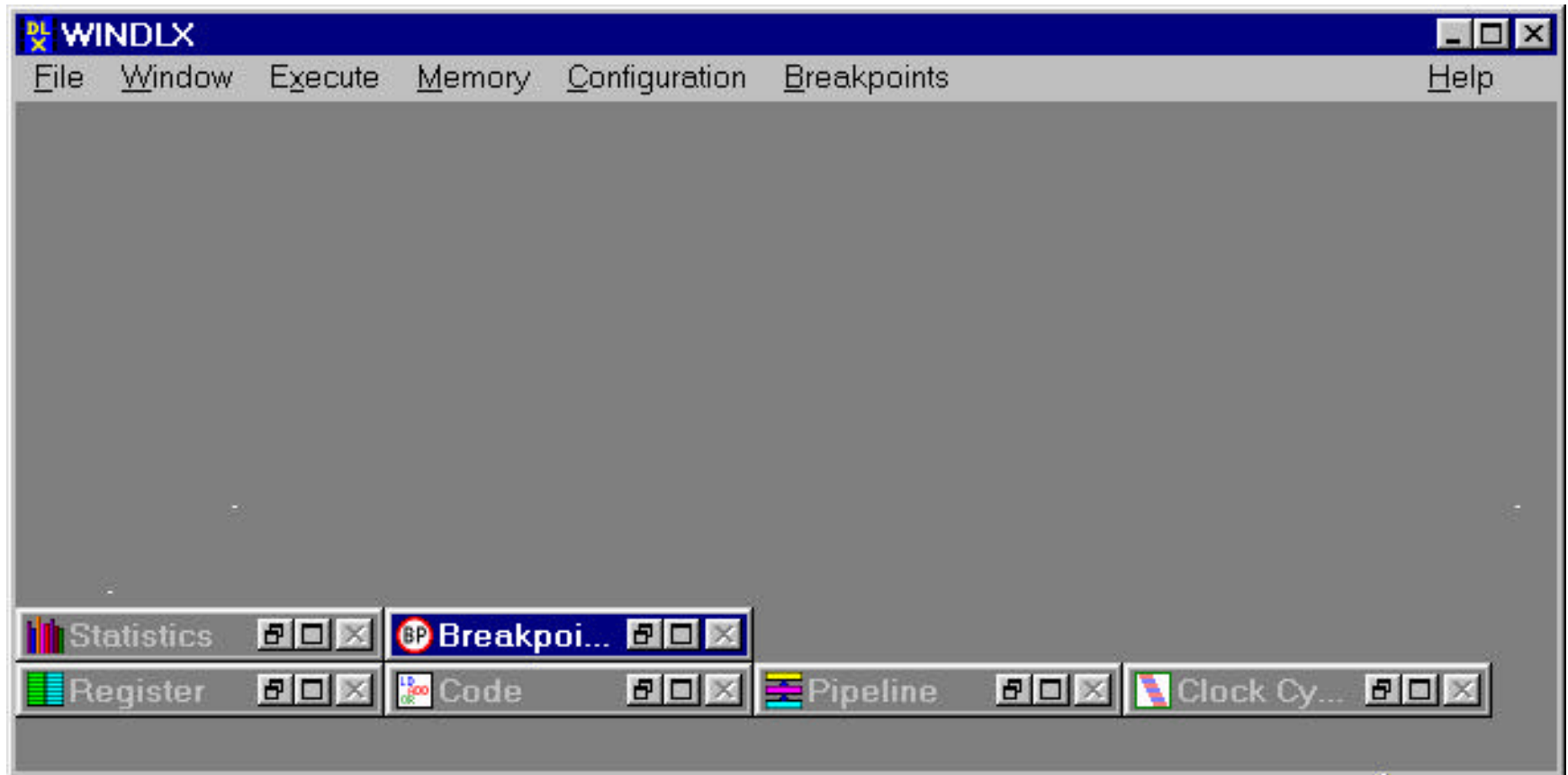
- Reordenamiento



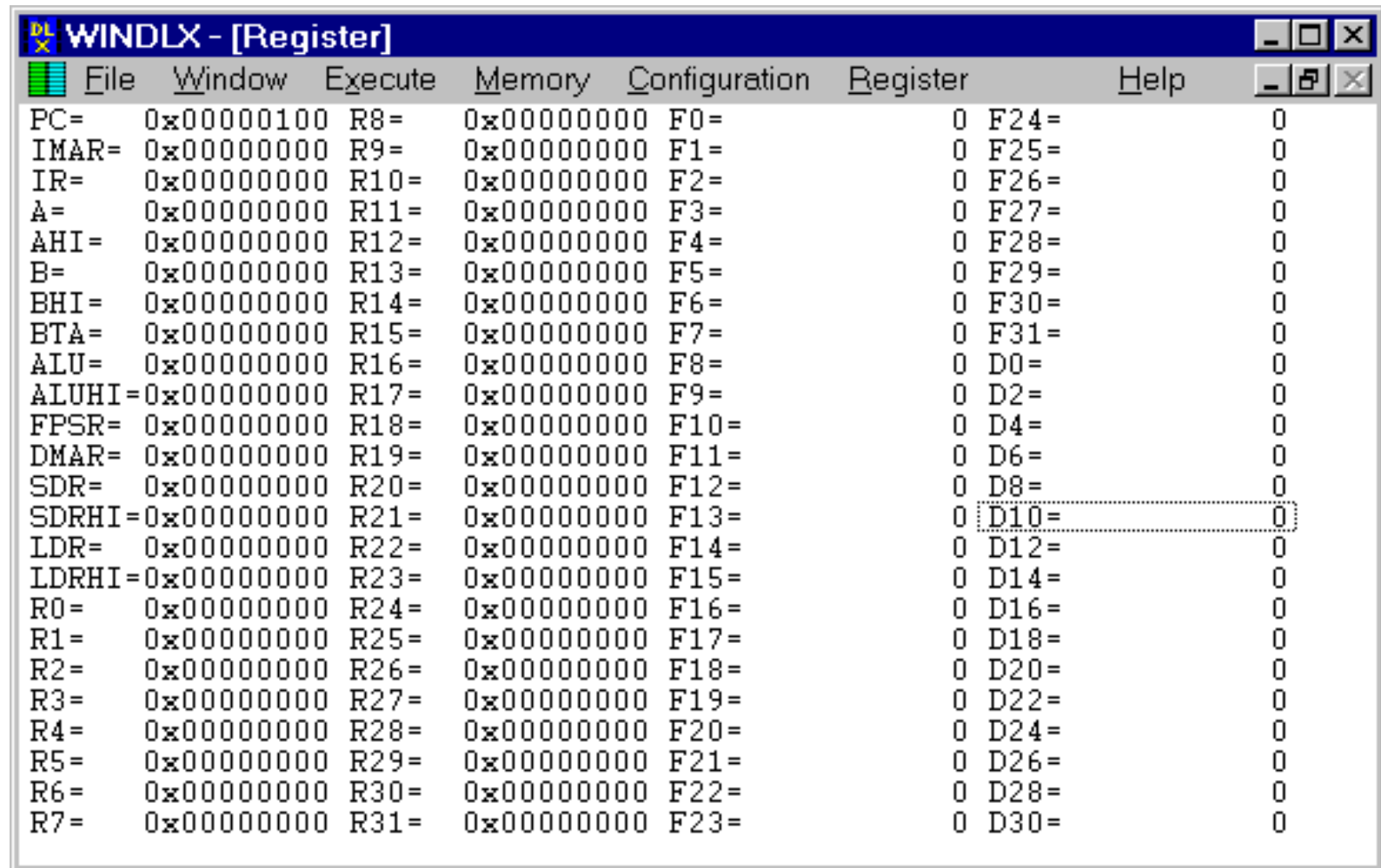
Bloque II – Procesadores Segmentados

WINDLX

Entorno de Simulación



Entorno de Simulación. Register

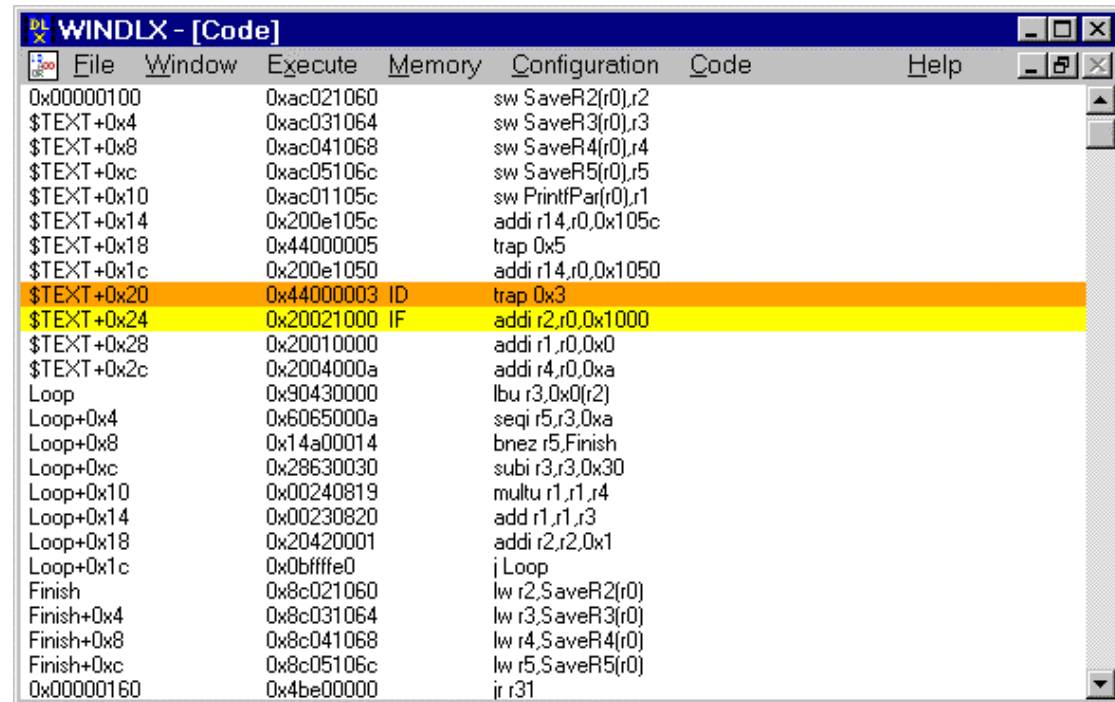


The screenshot shows a window titled "WINDLX - [Register]" with a menu bar containing "File", "Window", "Execute", "Memory", "Configuration", "Register", and "Help". The main area displays a list of registers and their values, organized in four columns. The registers are listed in hexadecimal format, and the values are in decimal format. The register "D10" is highlighted with a dotted border.

PC=	0x00000100	R8=	0x00000000	F0=	0	F24=	0
IMAR=	0x00000000	R9=	0x00000000	F1=	0	F25=	0
IR=	0x00000000	R10=	0x00000000	F2=	0	F26=	0
A=	0x00000000	R11=	0x00000000	F3=	0	F27=	0
AHI=	0x00000000	R12=	0x00000000	F4=	0	F28=	0
B=	0x00000000	R13=	0x00000000	F5=	0	F29=	0
BHI=	0x00000000	R14=	0x00000000	F6=	0	F30=	0
BTA=	0x00000000	R15=	0x00000000	F7=	0	F31=	0
ALU=	0x00000000	R16=	0x00000000	F8=	0	D0=	0
ALUHI=	0x00000000	R17=	0x00000000	F9=	0	D2=	0
FPSR=	0x00000000	R18=	0x00000000	F10=	0	D4=	0
DMAR=	0x00000000	R19=	0x00000000	F11=	0	D6=	0
SDR=	0x00000000	R20=	0x00000000	F12=	0	D8=	0
SDRHI=	0x00000000	R21=	0x00000000	F13=	0	D10=	0
LDR=	0x00000000	R22=	0x00000000	F14=	0	D12=	0
LDRHI=	0x00000000	R23=	0x00000000	F15=	0	D14=	0
R0=	0x00000000	R24=	0x00000000	F16=	0	D16=	0
R1=	0x00000000	R25=	0x00000000	F17=	0	D18=	0
R2=	0x00000000	R26=	0x00000000	F18=	0	D20=	0
R3=	0x00000000	R27=	0x00000000	F19=	0	D22=	0
R4=	0x00000000	R28=	0x00000000	F20=	0	D24=	0
R5=	0x00000000	R29=	0x00000000	F21=	0	D26=	0
R6=	0x00000000	R30=	0x00000000	F22=	0	D28=	0
R7=	0x00000000	R31=	0x00000000	F23=	0	D30=	0

Entorno de Simulación. Code

- ▶ *Visualizadas:*
 - ▶ *Instrucciones*
 - ▶ Puntos de ruptura (*breakpoints*)
- ▶ Instrucción está ejecutándose en una etapa determinada del *pipeline*:
 - ▶ *un color* característico de cada etapa
 - ▶ Aparece una etiqueta de la etapa.

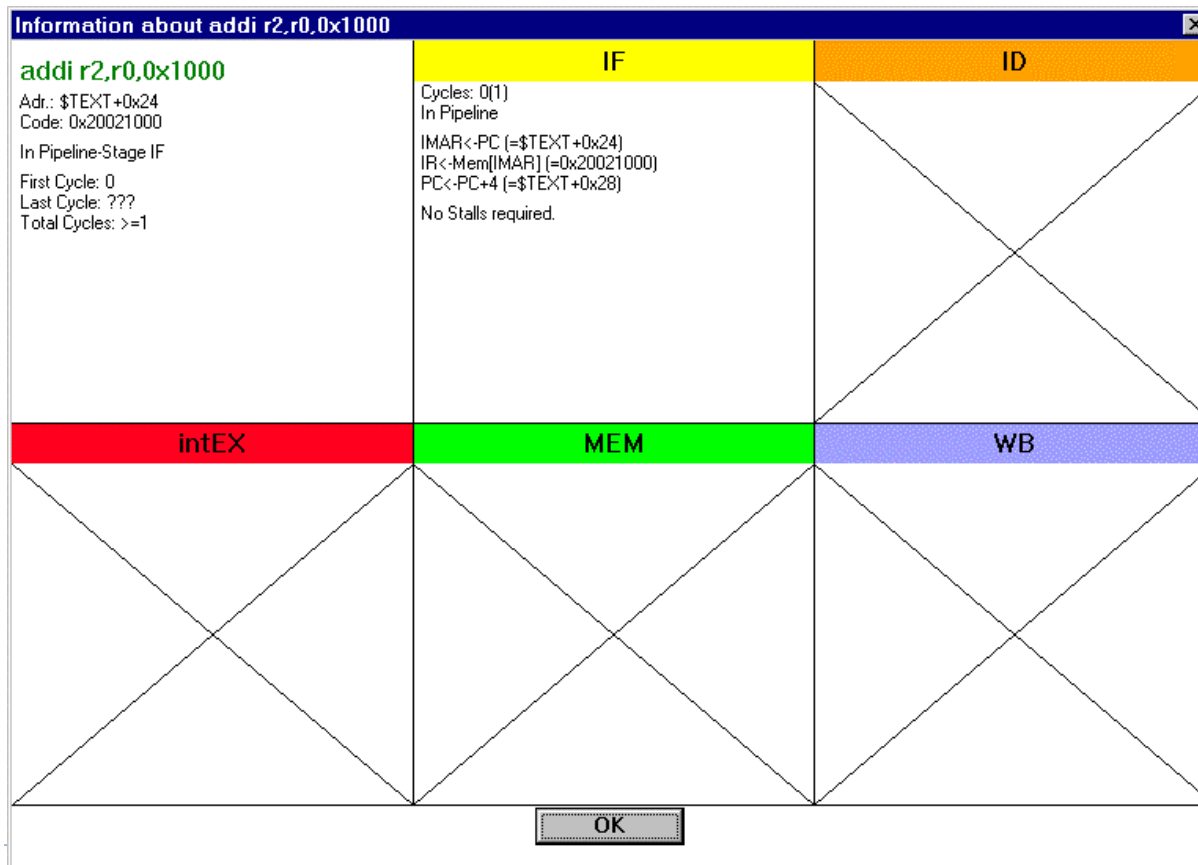


The screenshot shows a window titled "WINDLX - [Code]" with a menu bar (File, Window, Execute, Memory, Configuration, Code, Help) and a toolbar. The main area displays a list of instructions and their execution stages in a pipeline. The instructions are listed in the first column, and the execution stages are listed in the second column. The instructions are: 0x00000100, \$TEXT+0x4, \$TEXT+0x8, \$TEXT+0xc, \$TEXT+0x10, \$TEXT+0x14, \$TEXT+0x18, \$TEXT+0x1c, \$TEXT+0x20, \$TEXT+0x24, \$TEXT+0x28, \$TEXT+0x2c, Loop, Loop+0x4, Loop+0x8, Loop+0xc, Loop+0x10, Loop+0x14, Loop+0x18, Loop+0x1c, Finish, Finish+0x4, Finish+0x8, Finish+0xc, and 0x00000160. The execution stages are: 0xac021060, 0xac031064, 0xac041068, 0xac05106c, 0xac01105c, 0x200e105c, 0x44000005, 0x200e1050, 0x44000003 ID, 0x20021000 IF, 0x20010000, 0x2004000a, 0x90430000, 0x6065000a, 0x14a00014, 0x28630030, 0x00240819, 0x00230820, 0x20420001, 0x0bffffe0, 0x8c021060, 0x8c031064, 0x8c041068, 0x8c05106c, and 0x4be00000. The instructions are color-coded to represent different stages of the pipeline: 0x00000100 to 0x200e1050 are in the Instruction Fetch (IF) stage (yellow), 0x44000003 ID is in the Instruction Decode (ID) stage (orange), 0x20021000 IF is in the Instruction Fetch (IF) stage (yellow), 0x20010000 to 0x0bffffe0 are in the Instruction Decode (ID) stage (orange), 0x8c021060 to 0x8c05106c are in the Instruction Decode (ID) stage (orange), and 0x4be00000 is in the Instruction Decode (ID) stage (orange).

	Execute	Memory	Configuration	Code
0x00000100	0xac021060		sw SaveR2(r0),r2	
\$TEXT+0x4	0xac031064		sw SaveR3(r0),r3	
\$TEXT+0x8	0xac041068		sw SaveR4(r0),r4	
\$TEXT+0xc	0xac05106c		sw SaveR5(r0),r5	
\$TEXT+0x10	0xac01105c		sw PrintfPar(r0),r1	
\$TEXT+0x14	0x200e105c		addi r14,r0,0x105c	
\$TEXT+0x18	0x44000005		trap 0x5	
\$TEXT+0x1c	0x200e1050		addi r14,r0,0x1050	
\$TEXT+0x20	0x44000003 ID		trap 0x3	
\$TEXT+0x24	0x20021000 IF		addi r2,r0,0x1000	
\$TEXT+0x28	0x20010000		addi r1,r0,0x0	
\$TEXT+0x2c	0x2004000a		addi r4,r0,0xa	
Loop	0x90430000		lbu r3,0x0(r2)	
Loop+0x4	0x6065000a		seqi r5,r3,0xa	
Loop+0x8	0x14a00014		bnez r5,Finish	
Loop+0xc	0x28630030		subi r3,r3,0x30	
Loop+0x10	0x00240819		multu r1,r1,r4	
Loop+0x14	0x00230820		add r1,r1,r3	
Loop+0x18	0x20420001		addi r2,r2,0x1	
Loop+0x1c	0x0bffffe0		j Loop	
Finish	0x8c021060		lw r2,SaveR2(r0)	
Finish+0x4	0x8c031064		lw r3,SaveR3(r0)	
Finish+0x8	0x8c041068		lw r4,SaveR4(r0)	
Finish+0xc	0x8c05106c		lw r5,SaveR5(r0)	
0x00000160	0x4be00000		jr r31	

Entorno de Simulación. Code

► Información detallada de las instrucciones:



Entorno de Simulación.

Ventana y menú Pipeline

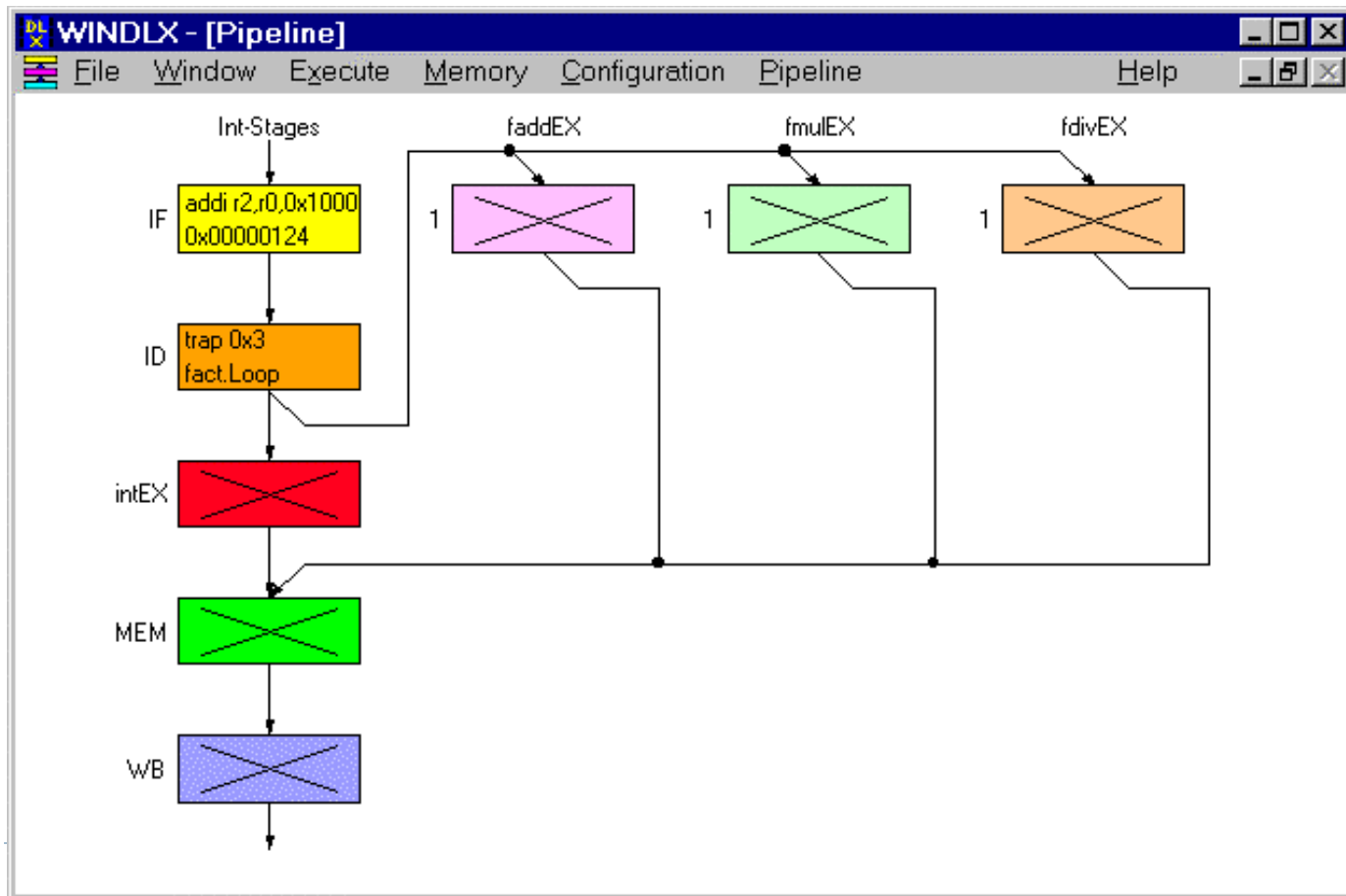
- ▶ Visualizan las etapas por las que pasan las instrucciones dentro de la estructura del pipeline del procesador.
- ▶ El menú Pipeline:
 - ▶ Display Floating point stages.
 - ▶ Activo:
 - ☐ las etapas en coma flotante
 - ▶ Desactivado
 - ☐ Las cinco etapas básicas del pipeline del DLX



Entorno de Simulación

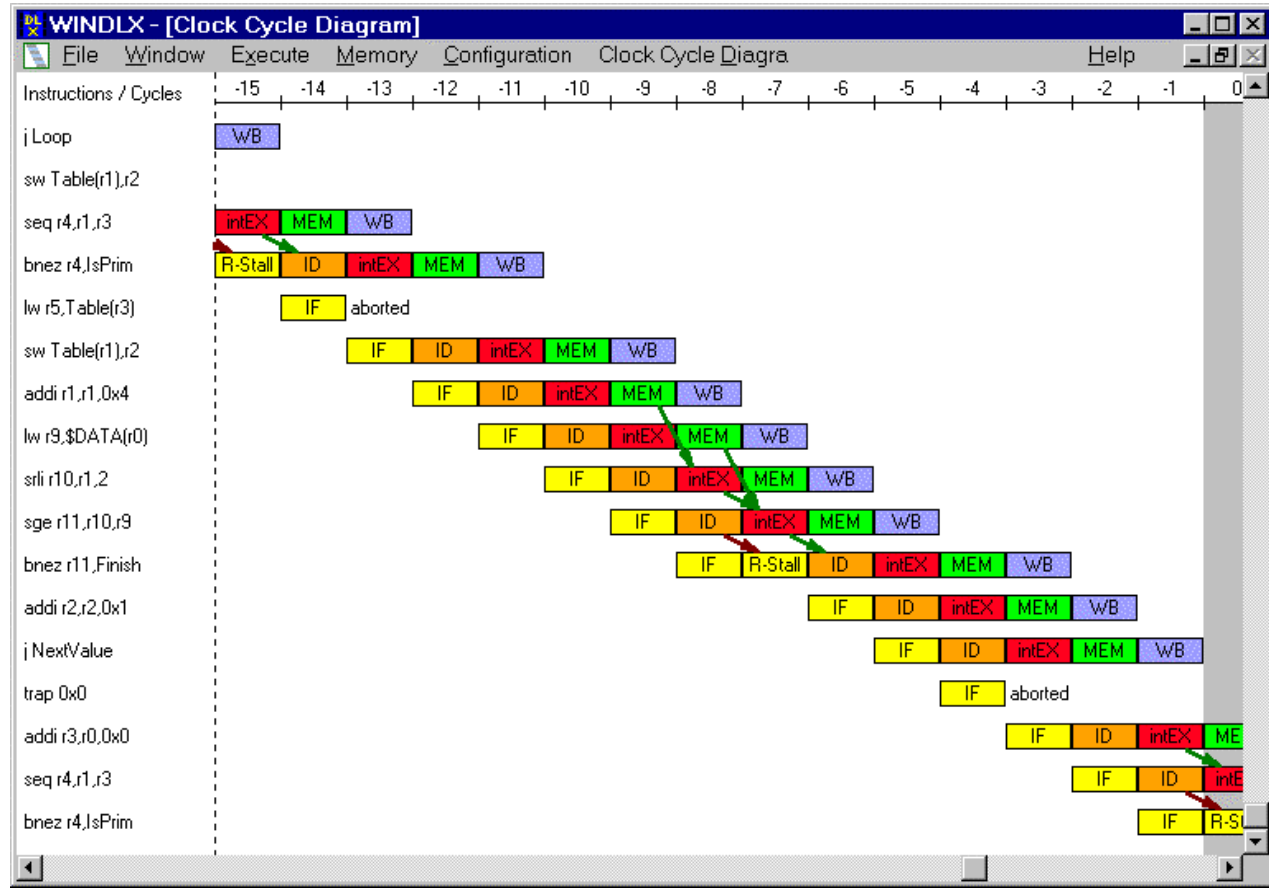
Ventana y menú Pipeline

- Información detallada de la instrucciones



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.*



Entorno de Simulación.

Ventana y menú Clock Cicle 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.
 - ▶ La instrucción de trap 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 DXL escribe sólo los registros en la etapa WVB, 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*.



Entorno de Simulación

Ventana y menú Clock Cycle Diagram

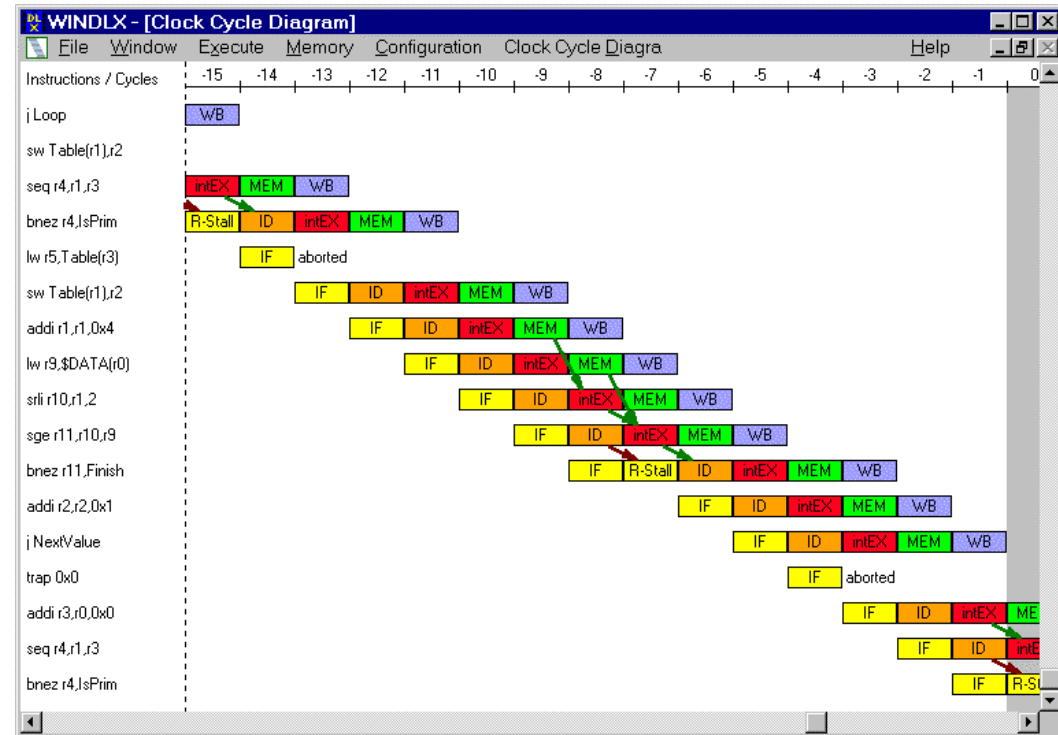
► El menú *Clock Cycle Diagram*:

► *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)*

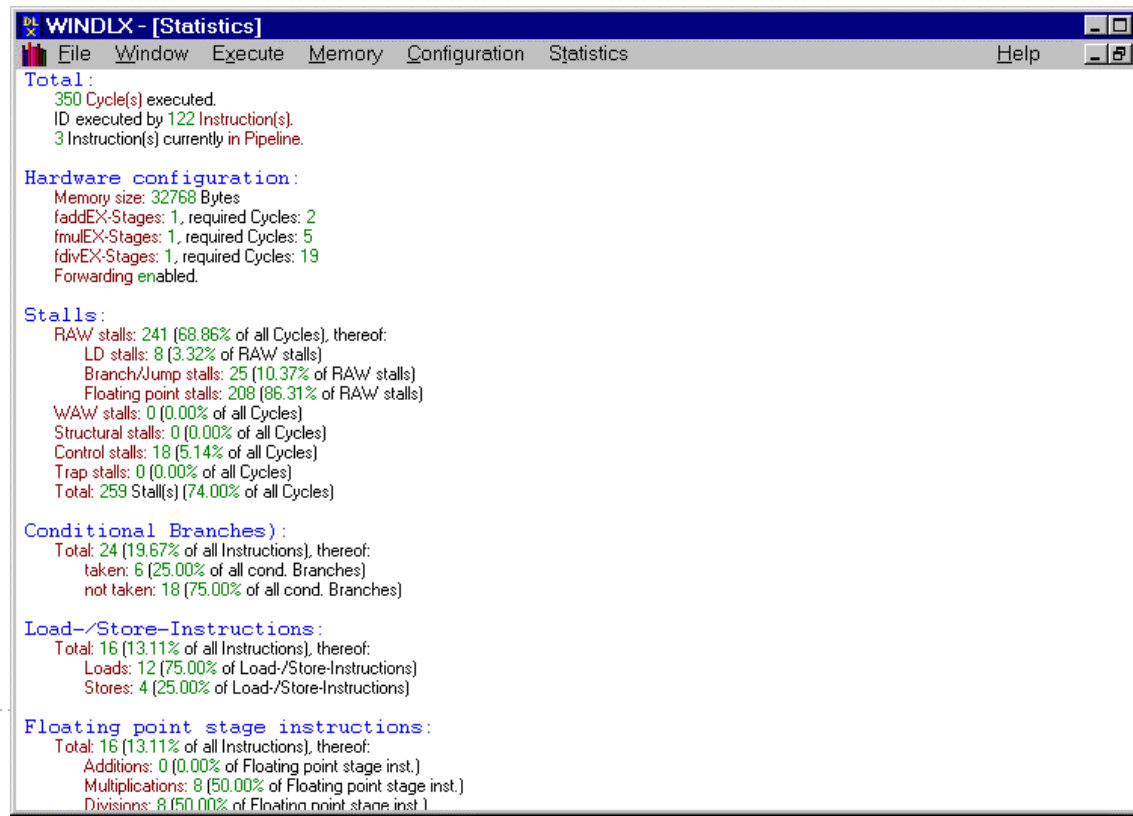
- Si esta opción está activa, la instrucción que causa una detención por riegos de datos (RAW o WAW) es marcada con una flecha roja.



Entorno de Simulación.

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*



The screenshot shows a window titled "WINDLX - [Statistics]" with a menu bar containing "File", "Window", "Execute", "Memory", "Configuration", "Statistics", and "Help". The main content area displays the following statistics:

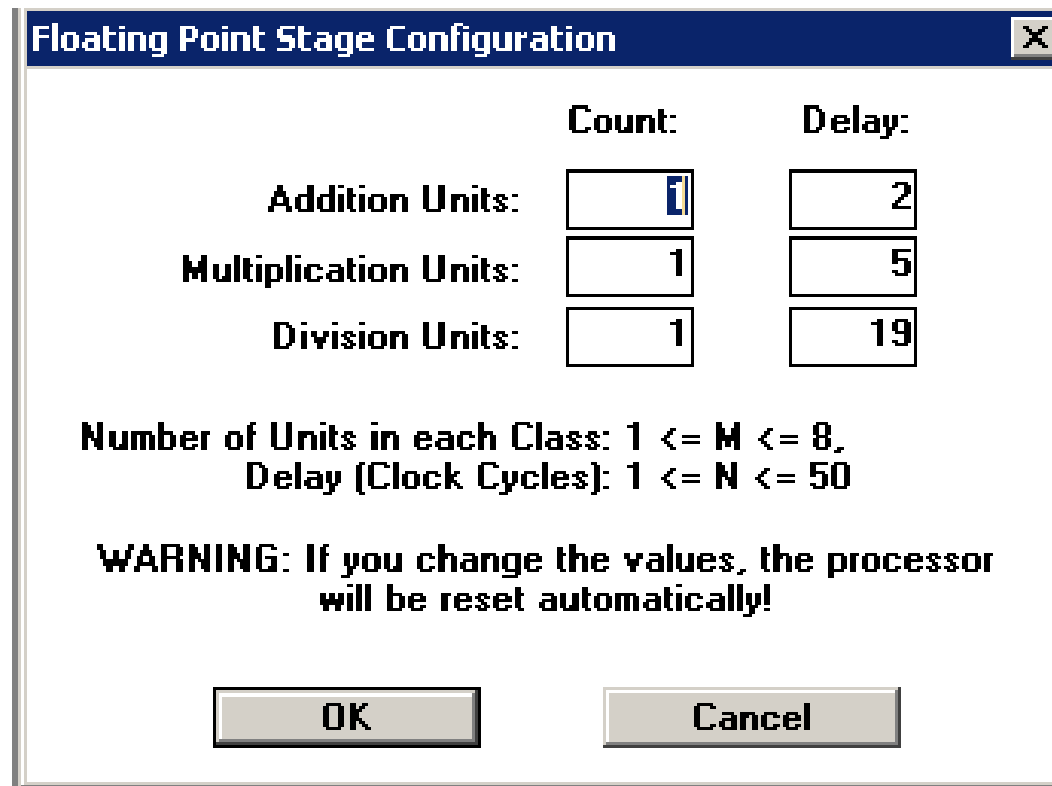
```
Total:  
350 Cycle(s) executed.  
ID executed by 122 Instruction(s).  
3 Instruction(s) currently in Pipeline.  
  
Hardware configuration:  
Memory size: 32768 Bytes  
faddEX-Stages: 1, required Cycles: 2  
fmulEX-Stages: 1, required Cycles: 5  
fdvEX-Stages: 1, required Cycles: 19  
Forwarding 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

Guión 4: Riesgos de Datos y Estructurales

Procesadores Segmentados

► Configuración de unidades de punto flotante



The dialog box is titled "Floating Point Stage Configuration" and contains a table for configuring floating-point units. The table has two columns: "Count" and "Delay". The rows are for Addition Units, Multiplication Units, and Division Units. The values are: Addition Units (Count: 1, Delay: 2), Multiplication Units (Count: 1, Delay: 5), and Division Units (Count: 1, Delay: 19). Below the table, there is a warning message: "WARNING: If you change the values, the processor will be reset automatically!". At the bottom, there are "OK" and "Cancel" buttons.

	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

Guión 4: Riesgos de Datos y Estructurales

I. 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

Guión 4: 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) → f10
- ▶ Calcule la multiplicación de los números que ocupan la posición impar (105) → f12,
 - ▶ 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

Guión 4: 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

Guión 5: Riesgos de Control

Guión 5: Riesgos de Control

2. 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 → f10 y calcule la multiplicación de los números que ocupan la posición impar → f12, usando reordenando instrucciones y considerando bypass.
- ▶ Informe
 - ▶ Número de ciclos
 - ▶ Detenciones debidas a riesgos de datos
 - ▶ Detenciones debidas a riesgos control
- c. 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

Bloque II – Procesadores Segmentados

Guión 6: Riesgos de Datos y de Control

Guión 6: Riesgos de Datos y de Control

- d. Considerando el programa del guión 4, 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
