

Dept. de Teoría de la Señal y Comunicaciones y Sistemas
Telemáticos y Computación
Área de Telemática (GSyC)

Segmentación

Katia Leal Algara

katia.leal@urjc.es

<http://gsyc.escet.urjc.es/~katia/>



Paralelismo

- ☐ Las arquitecturas basadas en Von Neumann presentan limitaciones técnicas en la velocidad de proceso que pueden alcanzar
- ☐ Arquitecturas alternativas que utilizan varias unidades de procesamiento
 - ☐ **Arquitecturas paralelas**
- ☐ Tipos de paralelismo
 - ☐ **Paralelismo interno** (con una única CPU)
 - ☐ **Segmentación**
 - ☐ **Paralelismo explícito** (con varias CPUs)
 - ☐ SIMD, un solo flujo de instrucciones y múltiples datos
 - ☐ MISD, varios flujos de instrucciones y uno solo de datos
 - ☐ MIMD, arquitectura multiprocesador con varios flujos tanto de instrucciones como de datos

Paralelismo explícito

- ❑ **SIMD**: repertorios que consisten en instrucciones que realizan una misma operación sobre un conjunto de datos. Todos los procesadores reciben la misma instrucción de la unidad de control, pero operan sobre diferentes datos
 - ❑ Ejemplos de estos repertorios son las extensiones multimedia **3DNow!** de AMD y **SSE** de Intel
- ❑ **MIMD**: tienen un número de procesadores que funcionan de forma asíncrona e independiente. En cualquier momento, cualquier procesador puede ejecutar diferentes instrucciones sobre distintos datos

Tipos de procesadores

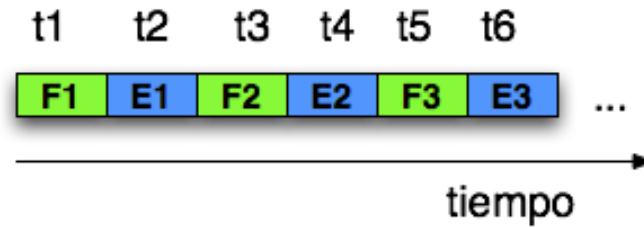
- ❑ **Procesadores secuenciales**, hasta que no termina de ejecutar una instrucción no comienza a ejecutar la siguiente
 - ❑ **Procesador monociclo**: la duración del ciclo de reloj vendrá fijada por la instrucción más lenta
 - ❑ **Procesador multiciclo**: la duración de la etapa más larga es la que fija el periodo del reloj
- ❑ **Procesadores segmentados**, permite solapar en el tiempo la ejecución de varias instrucciones
 - ❑ Aprovecha el paralelismo a nivel de instrucción, *pipelining*
 - ❑ Es normal que los microprocesadores actuales de propósito general incorporen *pipelining*

Encauzamiento

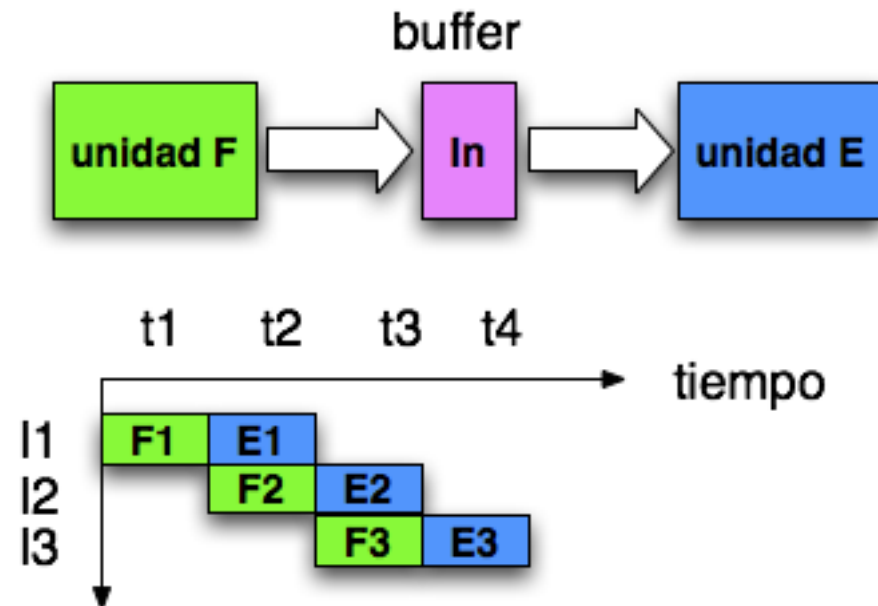
- ☐ Técnica para la generación de paralelismo implícito en computadores
 - ☐ Ejecución de varias instrucciones, o partes de instrucciones, usando una única unidad de proceso
- ☐ La segmentación o ***pipeline*** consiste en dividir una función en subfunciones independientes que se pueden realizar simultáneamente
 - ☐ Se tratan distintos procesos a la vez aunque en fases distintas
- ☐ Proceso similar a una **cadena de montaje**
 - ☐ En la entrada se aceptan nuevos elementos antes de que los previamente aceptados salgan por la salida
 - ☐ El **encauzamiento** consigue la ejecución de instrucciones en un tiempo muy inferior a los procesos no encauzados

Encauzamiento

- ❑ Ejecución secuencial



- ❑ Nueva organización del HW

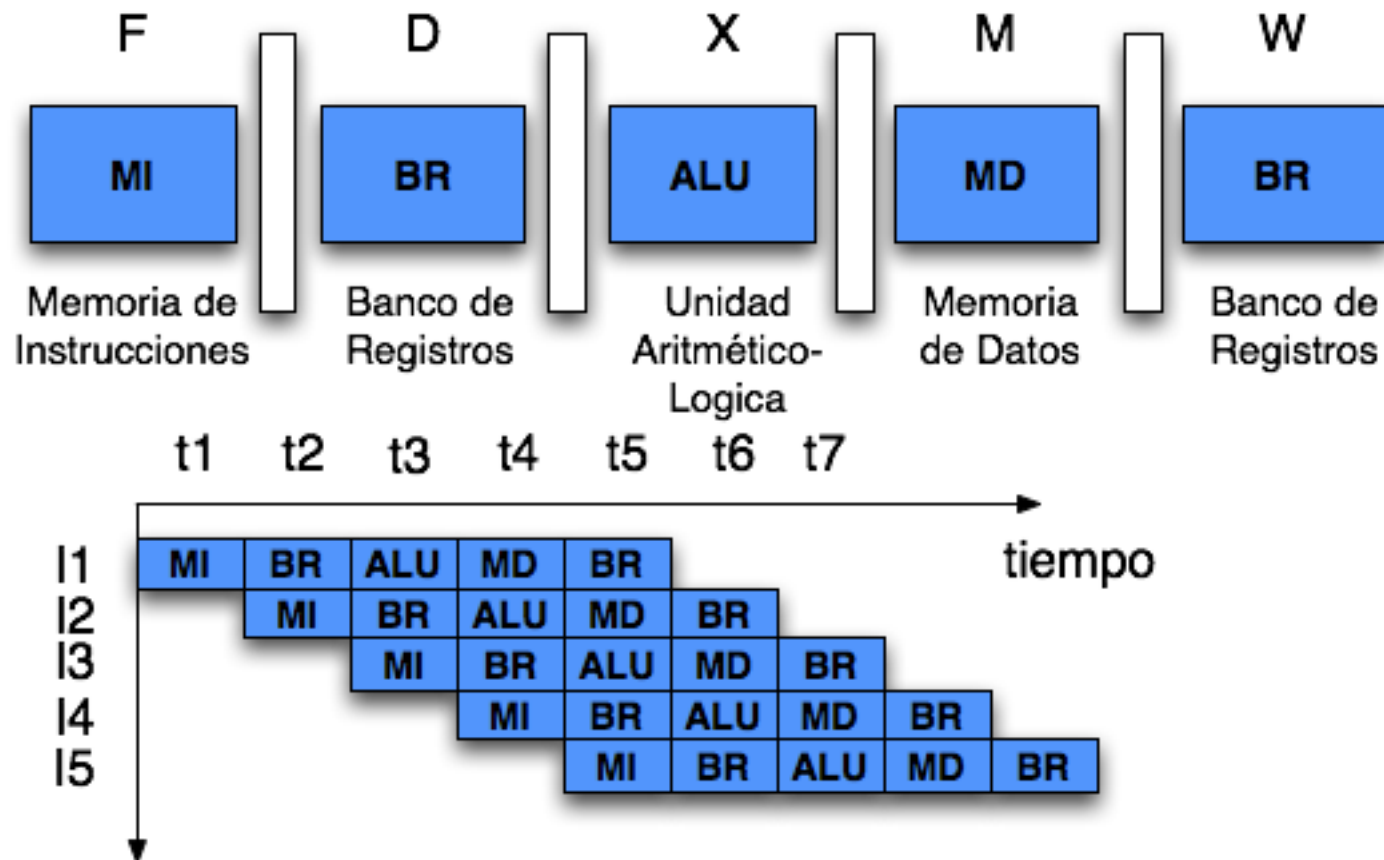


Conceptos básicos de segmentación

- ☐ ¿Qué es lo deseable?
 - ☐ $CPI = 1$
 - ☐ Periodo de reloj limitado por la etapa más lenta
- ☐ La segmentación surgió como una técnica para el aumento de prestaciones
- ☐ Ejecutando una única instrucción cada vez, el hardware está desaprovechado la mayor parte del tiempo
- ☐ Para segmentar el nanoMIPS multicycle basta con comenzar la ejecución de una nueva instrucción en cada ciclo
- ☐ En cada etapa las diferentes instrucciones tienen que utilizar diferentes recursos para evitar conflictos

Concepto de segmentación

Conceptos básicos de segmentación



Conceptos básicos de segmentación

Problemas

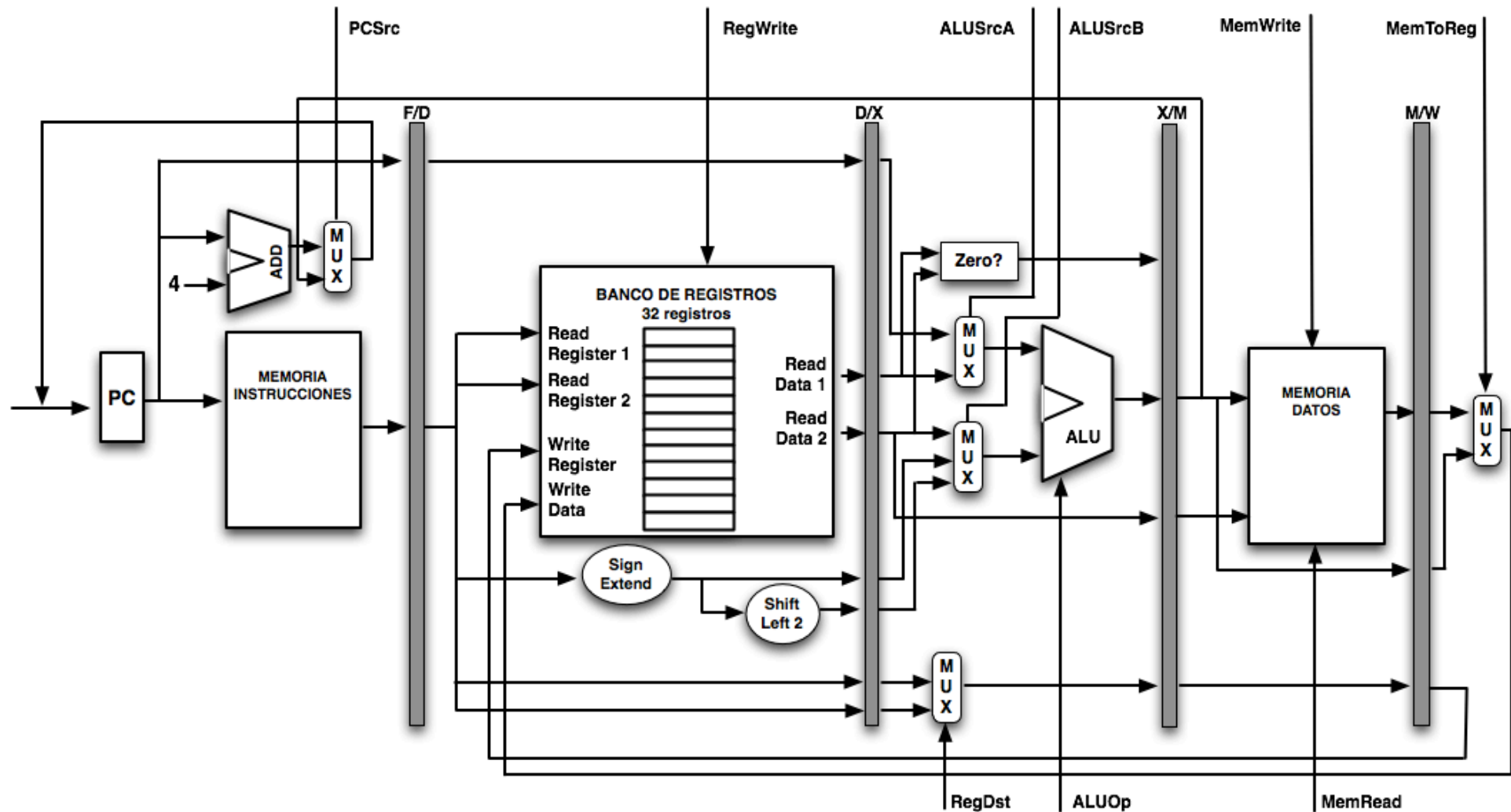
- ☐ Se realizan accesos a memoria en las etapas F y M
- ☐ Accesos al banco de registros en las etapas D y W
- ☐ El PC cambia en F, pero las instrucciones de salto lo pueden modificar en M

Soluciones

- ☐ Se separan las memorias de instrucciones y datos
- ☐ La escritura en los registros se realiza en la primera mitad del ciclo, y la lectura en la segunda ¿por qué no hacemos lo mismo con la memoria?
- ☐ Multiplexor etapa F
- ☐ Restador etapa X para realizar al mismo tiempo la suma del desplazamiento al PC en la ALU y la comparación de los registros en el caso de las instrucciones BEQ
- ☐ **Registros de segmentación**, almacenan el resultado de cada etapa al final del ciclo de reloj. Cada uno tendrá una longitud diferente dependiendo de la información que tenga que almacenar

Concepto de segmentación

Ruta de datos del nanoMIPS segmentado

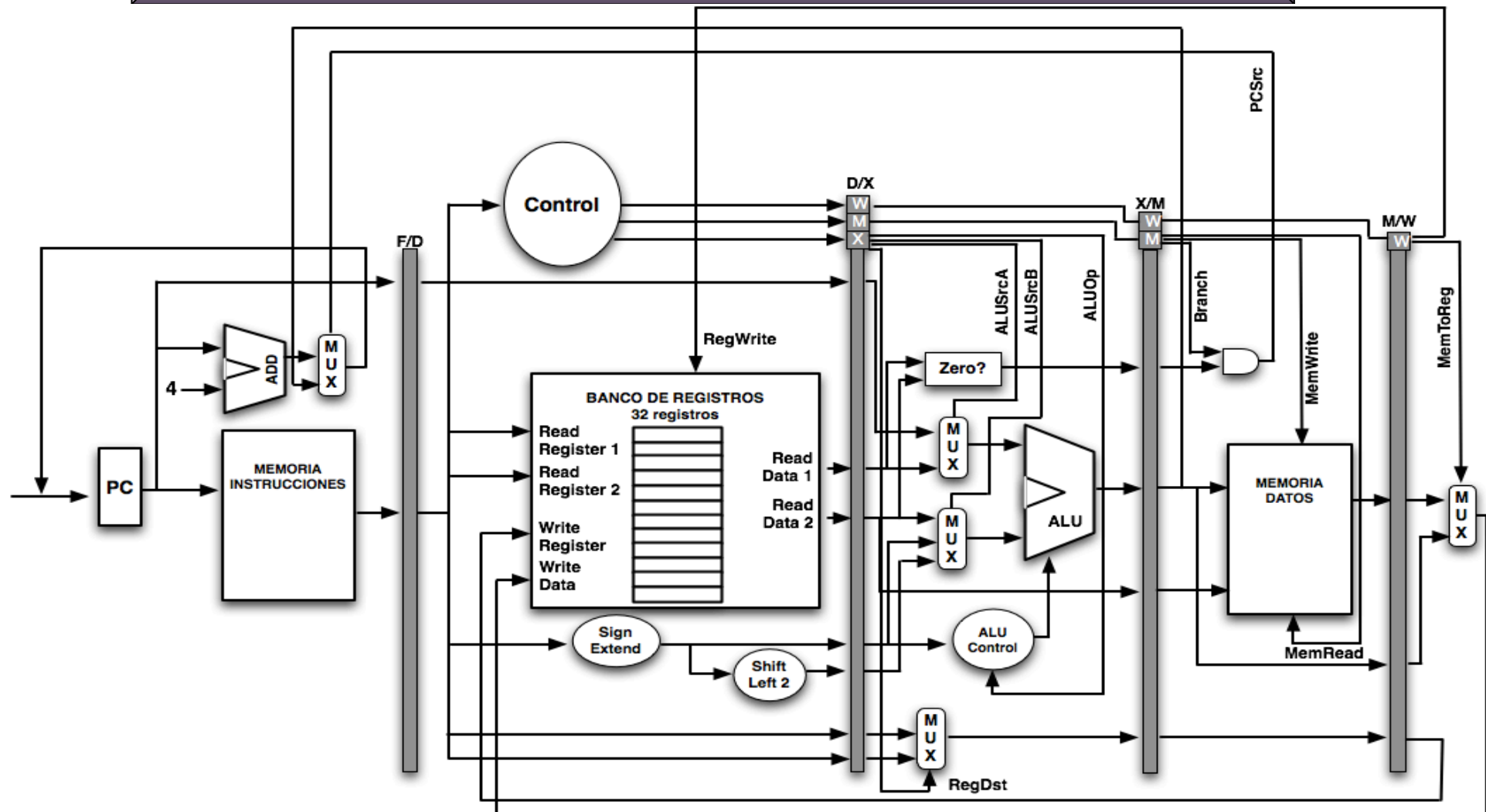


Unidad de control del procesador segmentado

- ☐ En la etapa D, el opcode permite generar todas las señales de control necesarias
- ☐ Las señales se propagan por los registros de segmentación
- ☐ De nuevo, tenemos un circuito combinacional que genera las señales de control para cada tipo de instrucción a partir del opcode
- ☐ Dimensionar los registros de segmentación
 - ☐ $X \rightarrow \text{ALUSrcA, ALUSrcB, ALUOp y RegDst}$
 - ☐ $M \rightarrow \text{Branch, MemRead y MemWrite}$
 - ☐ $W \rightarrow \text{MemToReg y RegWrite}$

Concepto de segmentación

Unidad de control del procesador segmentado



Rendimiento de un procesador segmentado

- ❑ **La productividad aumenta** ya que idealmente se completa una nueva instrucción por ciclo, $CPI = 1$
 - ❑ CPI de un procesador secuencial monociclo
- ❑ **El tiempo de ejecución de una única instrucción empeora** debido al hardware adicional que se introduce en la ruta de datos
 - ❑ Periodo de reloj de un procesador multiciclo
- ❑ **Speedup** máximo al segmentar un procesador multiciclo

$$S = t_{\text{multiciclo}} / t_{\text{segmentado}}$$

$$S = (I \cdot CPI \cdot T)_{\text{multiciclo}} / (I \cdot CPI \cdot T)_{\text{segmentado}}$$

$$S = 5 / 1 = 5$$

Ejemplo

- ❑ Comparación de una versión multiciclo del nanoMIPS con una versión segmentada
- ❑ Latencias componentes hardware multiciclo:
 - ❑ Lectura de la memoria: 0,3 ns
 - ❑ Escritura en la memoria: 0,45 ns
 - ❑ Lectura y escritura en el banco de registros: 0,05 ns
 - ❑ Operación aritmético-lógica en la ALU: 0,25 ns
 - ❑ Suma para preparar el siguiente PC: 0,1 ns

$$T = 0,45 \text{ ns}, \text{CPI} = 3,85$$

- ❑ Retardo de los registros de segmentación de 0,01 ns

$$T = 0,45 \text{ ns} + 2 \cdot 0,01 = 0,47 \text{ ns}$$

- ❑ **Speedup**

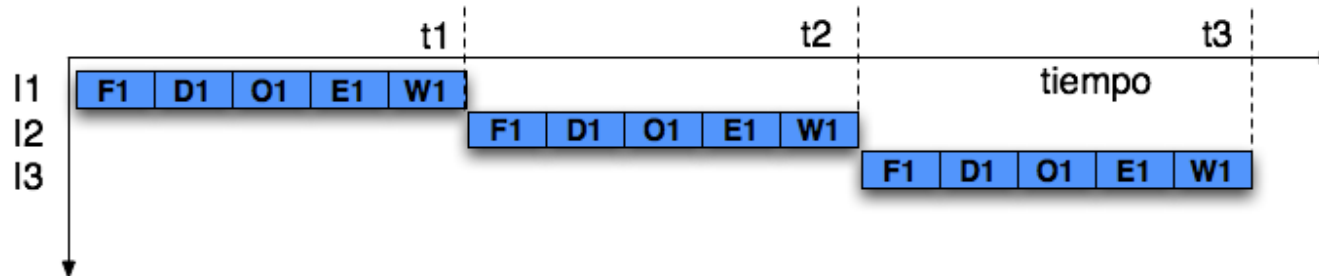
$$S = 3,85 \cdot 0,45 / 1 \cdot 0,47 = 3,68$$

Pipelining Vs Paralelismo

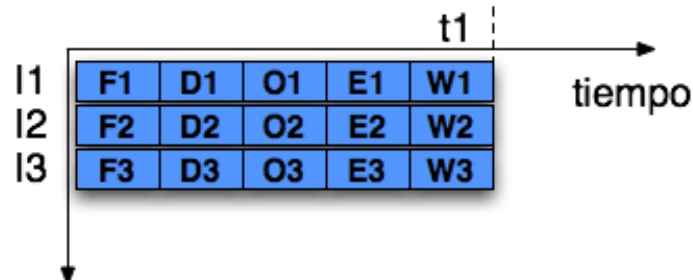
- ❑ Ambas técnicas están orientadas a mejorar el rendimiento (número de instrucciones por unidad de tiempo) incrementando el número de módulos hardware que operan simultáneamente
- ❑ ***Pipelining***
 - ❑ **El HW no está replicado**, sólo está dividido en varias etapas distintas especializadas
- ❑ **Arquitecturas paralelas**
 - ❑ **El HW sí está replicado**, por lo que varias operaciones pueden ejecutarse de forma simultánea

Pipelining Vs Paralelismo

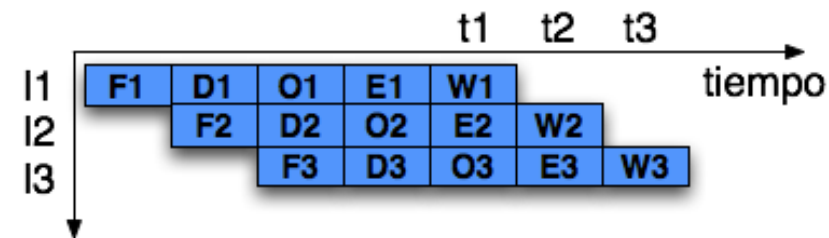
- ❑ **Secuencial pura:** un resultado cada 5 ciclos



- ❑ **Paralela pura:** N resultados cada 5 ciclos



- ❑ **Pipeline:** un resultado por ciclo



MIPS: ejemplo de juego de instrucciones para pipelining

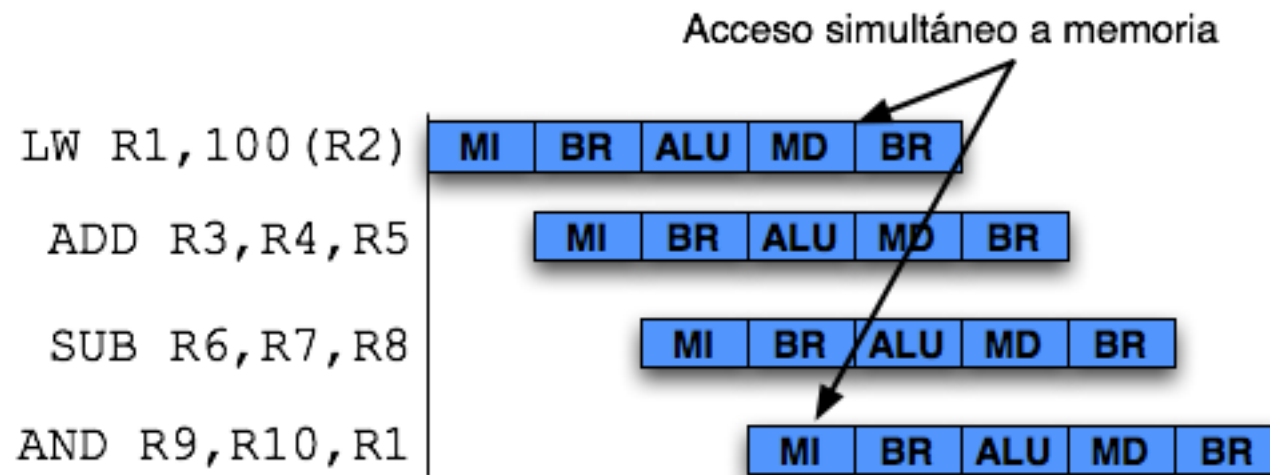
- ❑ El juego de instrucciones MIPS se diseñó para ser ejecutado en un pipeline
 1. Todas las instrucciones tienen la misma longitud, lo que facilita las etapas F y D
 2. MIPS solo tiene unos pocos formatos de instrucciones: encontrar los registros fuente en los mismos campos permite la lectura de los operandos al mismo tiempo que se decodifica la instrucción. En caso contrario necesitaríamos partir en dos la etapa D
 3. Operandos en memoria solo existen en instrucciones load/store, por lo que podemos utilizar la etapa de ejecución para calcular la dirección de memoria
 4. Los operandos deben estar alineados en memoria, por lo que solo se necesita un acceso a memoria
- ❑ En general, el repertorio MIPS está diseñado para evitar riesgos en el cauce

Resolución de riesgos en procesadores segmentados

- ❑ Lo ideal sería mantener todas las etapas del *pipeline* siempre llenas, pero esto no siempre es así
 - ❑ **Riesgos estructurales**
 - ❑ **Riesgos de datos**
 - ❑ **Riesgos de control**

Riesgos estructurales

- ❑ Se producen cuando **dos o más instrucciones necesitan utilizar el mismo recurso hardware al mismo tiempo**



Resolución parones estructurales

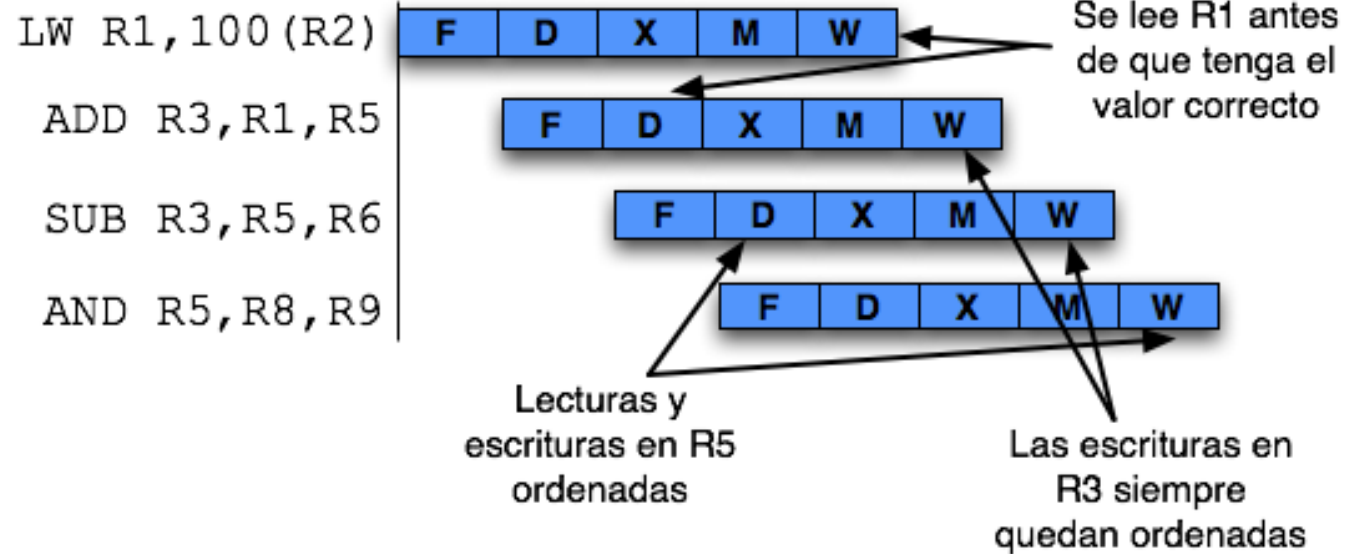
- ☐ Basta con **duplicar los recursos hardware** que provocan los conflictos, segmentarlos o realizar turnos para acceder a ellos
- ☐ Memoria
 - ☐ Separar entre memorias de instrucciones y datos
- ☐ Banco de registros
 - ☐ Realizar turnos para leer y escribir
 - ☐ Escrituras en la primera mitad de los ciclos de reloj
 - ☐ Lecturas en la segunda mitad

Riesgos de datos

- ☐ **Dos o más instrucciones presentan dependencias de datos entre sí** que podrían llevar a la obtención de **resultados erróneos** debido a una alteración en la secuencia de instrucciones
- ☐ Tipos de dependencias
 - ☐ **RAW**, *Read After Write* (RIESGO REAL!)
 - ☐ **WAR**, *Write After Read*
 - ☐ **WAW**, *Write After Write*

Ralentización del *pipeline*

Riesgos de datos



Riesgos de datos

- ☐ Solución software
 - ☐ **Prevención**
- ☐ Mecanismo hardware
 - ☐ **Detener el *pipeline***
 - ☐ **Anticipación, *data forwarding***

Riesgos de datos: Solución software, *Prevención*

☐ La solución software es responsabilidad del compilador

- ☐ No resuelve el riesgo, sino que lo evita
- ☐ Se trata de **reordenar el código**

```
I1  ld R7,200(R0)
I2  add  R3,R1,R2
I3  mul  R5,R3,R4
I4  ld  R1,200(R0)
I5  ld  R2,300(R0)
```



```
I2  add  R3,R1,R2
I1  ld  R7,200(R0)
I4  ld  R1,200(R0)
I3  mul  R5,R3,R4
I5  ld  R2,300(R0)
```

☐ Retrasar la ejecución de la instrucción dependiente **un número K de etapas** hasta que desaparezca el problema

- ☐ Encontrar K instrucciones que se pueden ejecutar después de la instrucción que genera la dependencia **sin variar la estructura lógica del programa**

Riesgos de datos: Solución software, *Prevención*

- ❑ Cuando no se pueden reordenar las instrucciones sin alterar la lógica del programa, entonces se deben **insertar instrucciones NOP**!

```
I1  ld    R1, 200(R0)
I2  add   R3, R1, R2
I3  mul   R5, R3, R4
I4  addi  R3, R3, 1
```

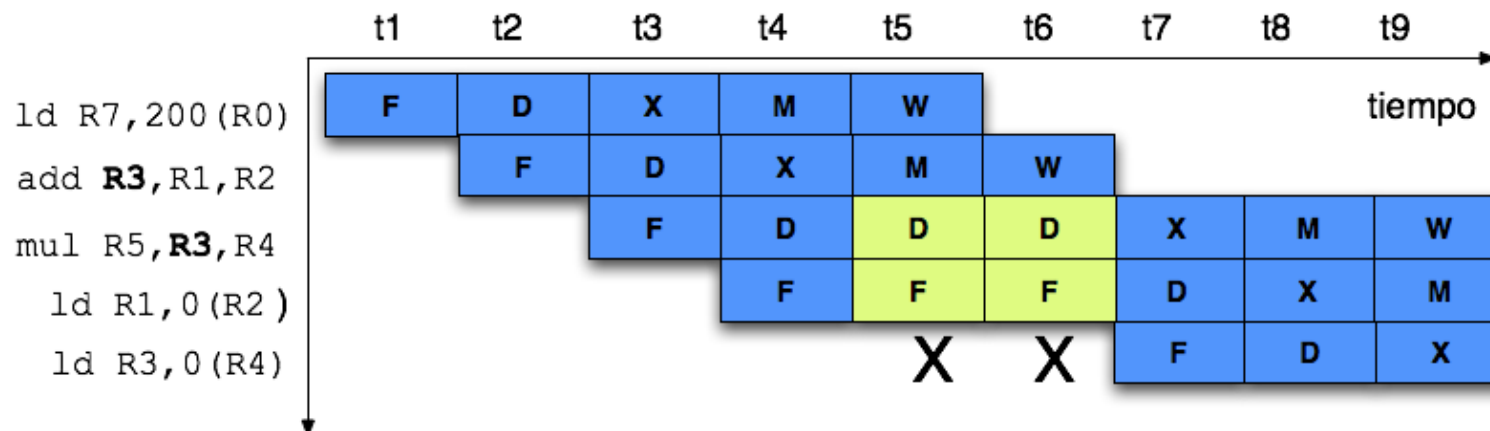


```
I1  ld    R1, 200(R0)
     NOP
     NOP
I2  add   R3, R1, R2
     NOP
     NOP
I3  mul   R5, R3, R4
I4  add   R3, R3, 1
```

- ❑ **La prevención no requiere de hardware adicional** pero a expensas de un compilador más complejo y una pérdida de tiempo si es necesario insertar instrucciones NOP

Riesgos de datos: Solución hardware, *Detener el pipeline*

- ❑ **Hardware adicional para detener la actividad en las etapas necesarias del *pipeline*** hasta que desaparezca la dependencia
- ❑ Primero se detecta la dependencia y después se detiene el *pipeline* a partir de la instrucción dependiente hasta que desaparece la dependencia

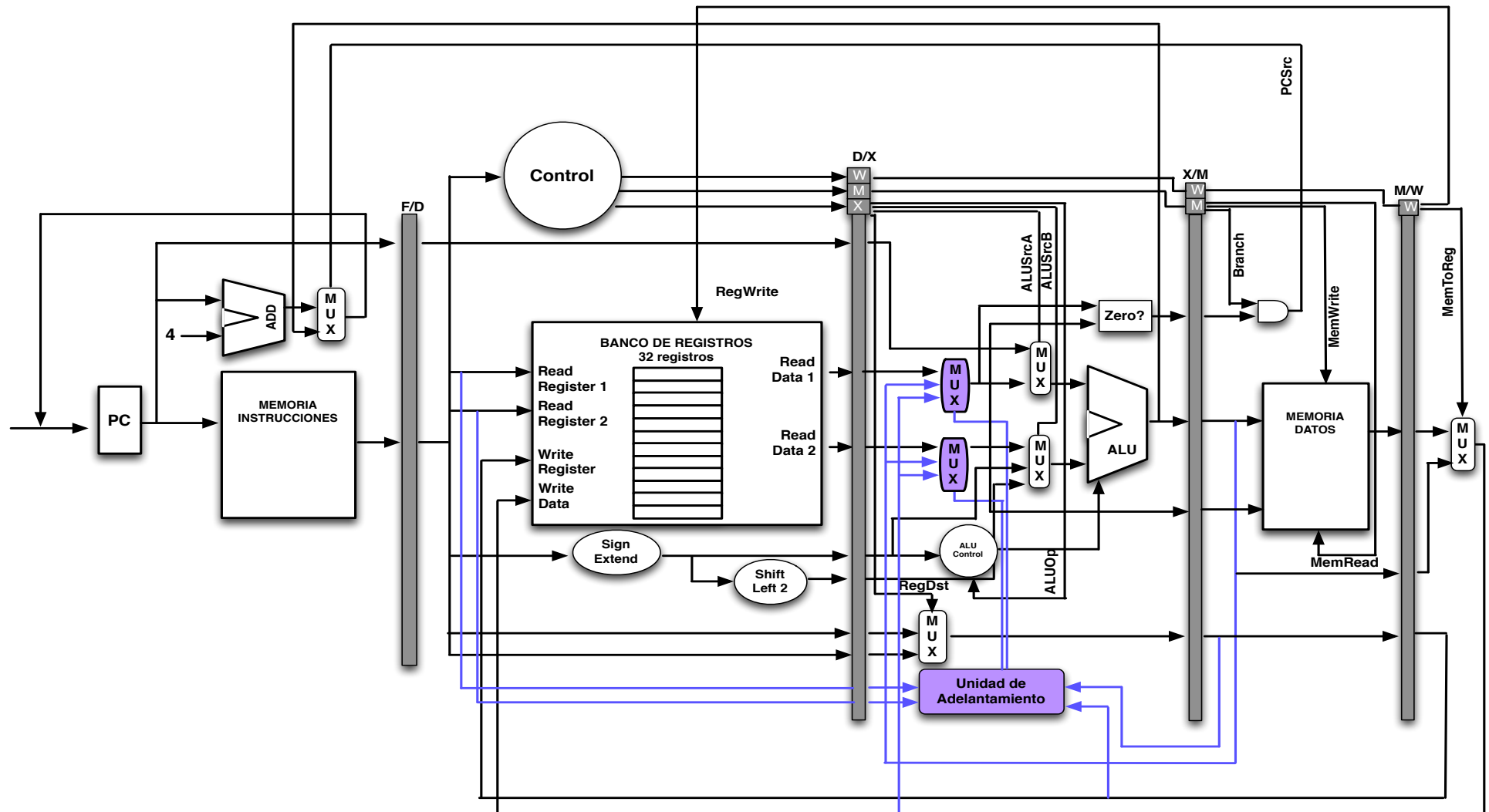


Riesgos de datos: Solución hardware, *Adelantamiento*

- ❑ **Técnica hardware de adelantamiento (*data forwarding*)**
 - ❑ Pasar el resultado obtenido en una instrucción a las instrucciones que lo necesitan como operando
 - ❑ Las instrucciones los reciben en cuanto está disponible
- ❑ Fácil de implementar, aunque no siempre se puede aplicar
- ❑ Identificar todos los posibles adelantamientos necesarios para el repertorio en cuestión
- ❑ En la siguiente transparencia se muestra una ruta de datos del nanoMIPS simplificada con las modificaciones necesarias para poder adelantar operandos a la etapa X
- ❑ Las mismas modificaciones se deberían realizar para adelantar operandos a la etapa M o a cualquier otra que pudiera necesitar un adelantamiento

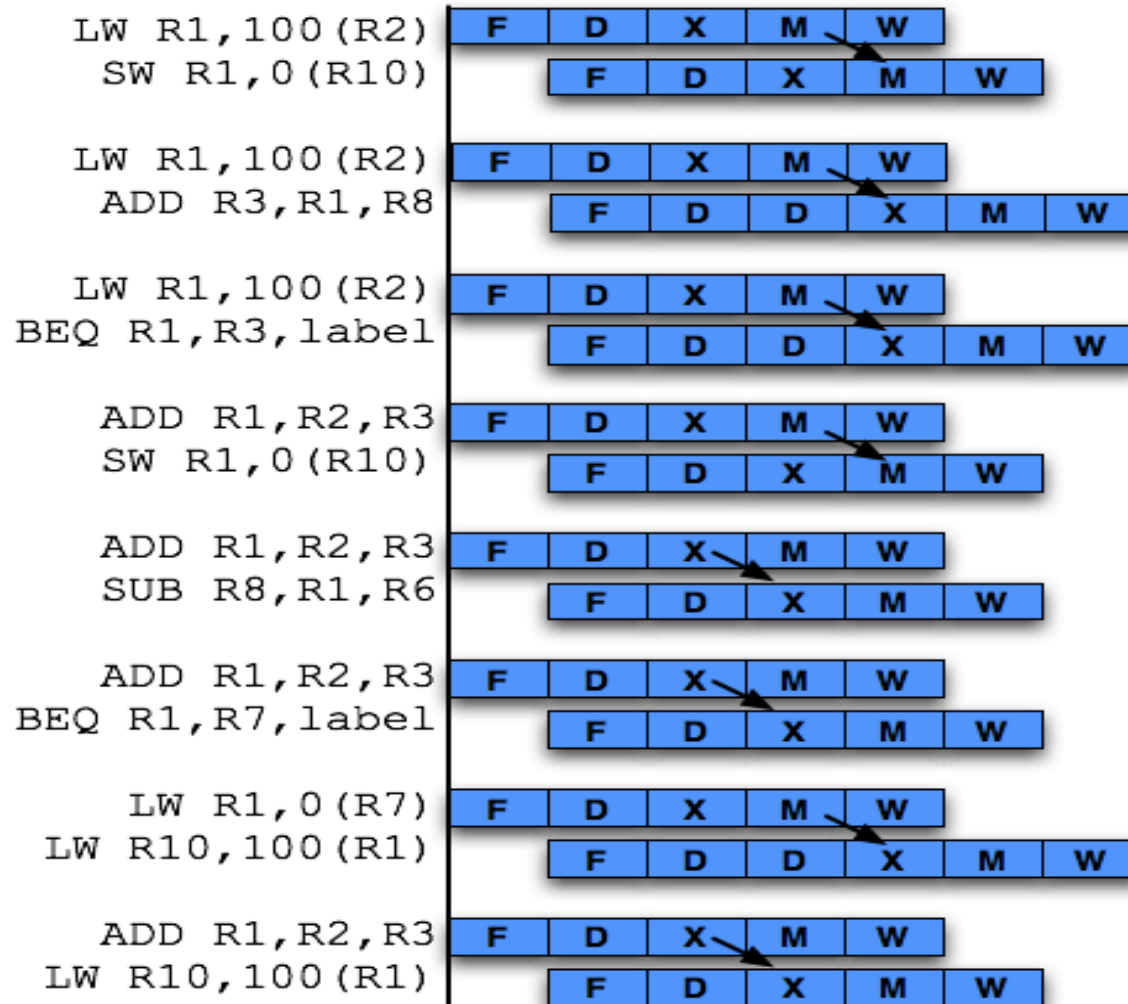
Ralentización del *pipeline*

Riesgos de datos: Solución hardware, *Adelantamiento*



Ralentización del *pipeline*

Riesgos de datos: Solución hardware, *Adelantamiento*

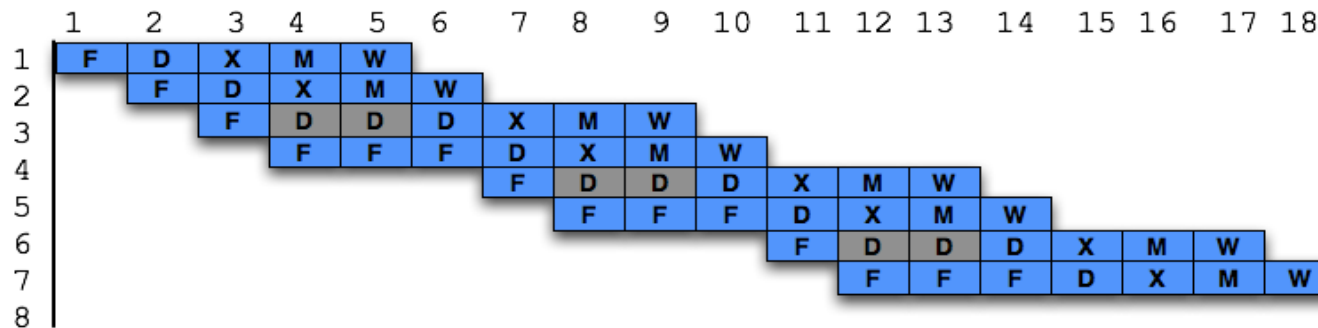


Ralentización del *pipeline*

Ejemplo adelantamiento

- ❑ Comparación de una versión nanoMIPS segmentado sin adelantamiento y con adelantamiento

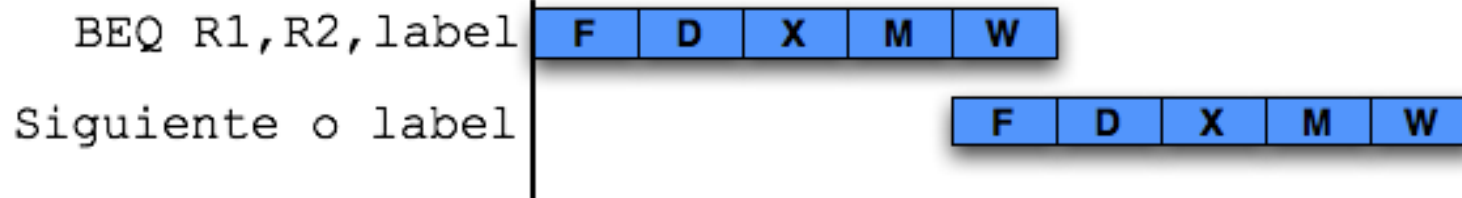
```
1  LW   R1,0(R0)
2  LW   R2,100(R0)
3  ADD  R3,R1,R2
4  SUB  R4,R1,R2
5  AND  R5,R3,R4
6  ADD  R6,R1,R4
7  SW   R6,200(R0)
8  BEQ  R6,R0,salto
```



- ❑ $CPI_{\text{sin adelantamiento}} = \text{ciclos}/I = 18/8 = 2,25$
- ❑ $CPI_{\text{con adelantamiento}} = \text{ciclos}/I = 13/8 = 1,625$
- ❑ Speedup de 1,38

Riesgos de control

- ❑ Una instrucción que modifica el valor del PC todavía no lo ha hecho cuando se tiene que comenzar la ejecución de la siguiente instrucción
- ❑ BEQ, hasta la fase M no carga el valor adecuado para el PC!

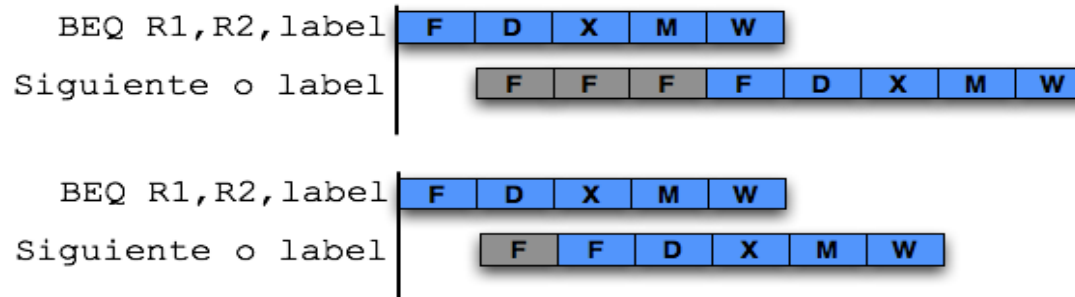


Riesgos de control

- ☐ Una instrucción que modifica el valor del PC todavía no lo ha hecho cuando se tiene que comenzar la ejecución de la siguiente instrucción
- ☐ Soluciones
 - ☐ **Hardware adicional**
 - ☐ **Predicción de salto estática**
 - ☐ No tomado
 - ☐ Tomado
 - ☐ **Software (compilador)**
 - ☐ Salto retardado o relleno de ranura

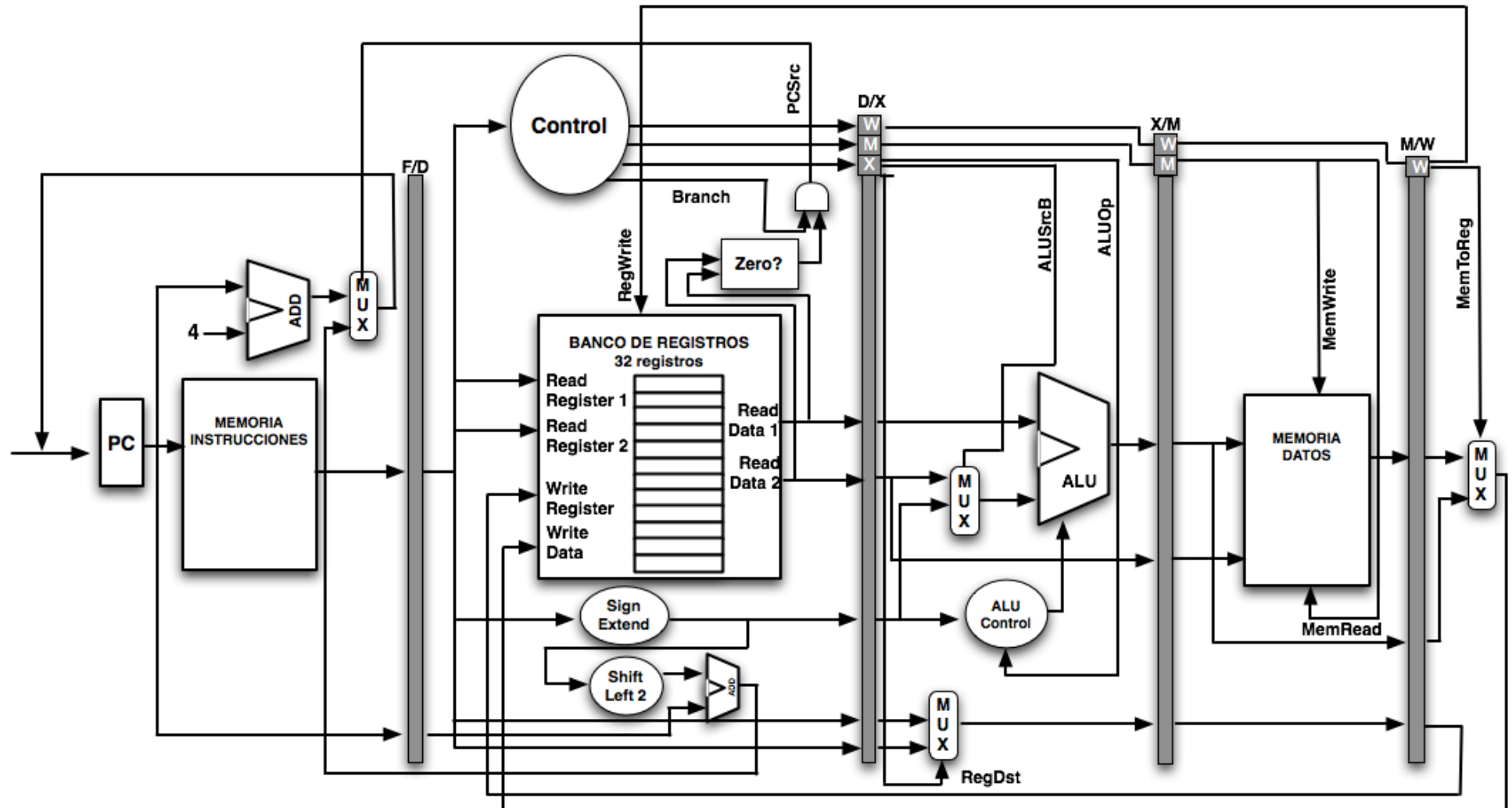
Riesgos de control: Hardware adicional, Adelantamiento

- ☐ Resolver un riesgo de control equivale a 3 ciclos de espera
- ☐ Podemos reducir esta espera a 1 ciclo adelantando la resolución de los saltos
- ☐ En la etapa D debemos incluir el hardware necesario para
 - ☐ Evaluar la condición de salto (restador, Zero?)
 - ☐ Calcular la dirección destino del salto (sumador)
- ☐ Seguimos teniendo una parada
 - ☐ Las instrucciones de salto constituyen el 20% de las instrucciones ejecutadas
 - ☐ Estas paradas empeoran significativamente el rendimiento de la segmentación



Ralentización del *pipeline*

Riesgos de control: Hardware adicional, Adelantamiento



Riesgos de control: *adelantamiento + predicción de salto estática*

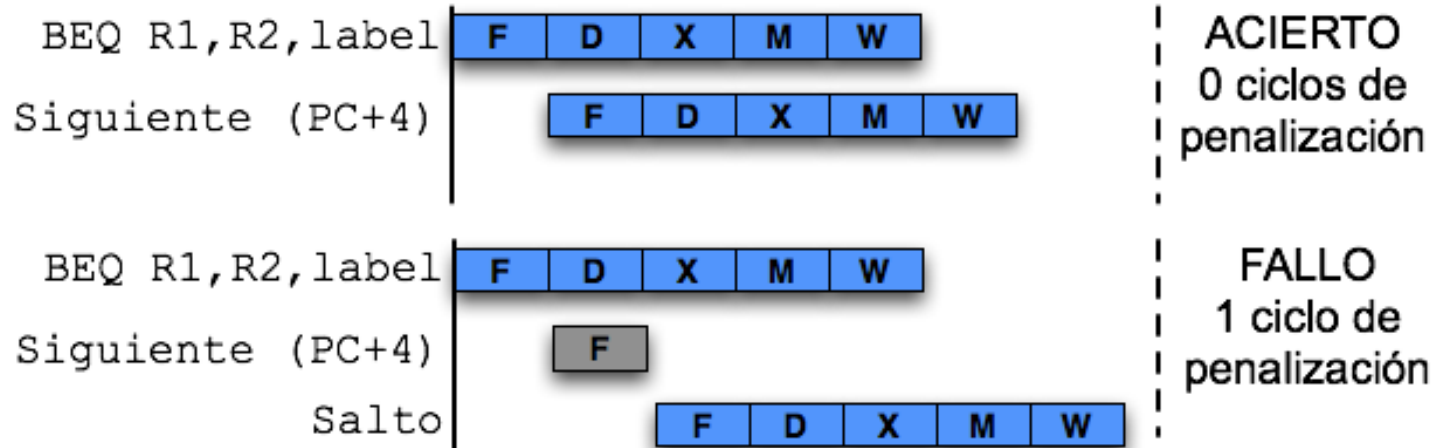
- ☐ Intenta evitar esta parada realizando una predicción
- ☐ Sólo se reduce la penalización cuando se acierta
- ☐ Predicciones estáticas, el procesador está diseñado para predecir en todos los casos que el salto se toma o que no se toma
 - ☐ **La predicción es siempre la misma**

Riesgos de control:

Predicción de *salto no tomado* en el nanoMIPS

- ☐ Penalización de un salto para saltos tomados y de ningún ciclo para saltos no tomados
- ☐ Mientras se decodifica se busca la siguiente instrucción
- ☐ Si el salto no se toma:
 - ☐ **Se continúa normalmente**
- ☐ Si el salto se toma:
 - ☐ **Se busca la instrucción y se pierde un ciclo que se hubiera perdido igualmente**

Riesgos de control: Predicción de *salto no tomado* en el nanoMIPS



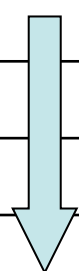
- ☐ Añadir a la unidad de control la lógica necesaria
- ☐ Parar ejecución antes de que lleguen a las etapas en las que se escribe en memoria o en los registros ¿peligro en el nanoMIPS?
- ☐ En nanoMIPS y otros micros similares no tiene sentido utilizar la predicción de salto tomado ¿por qué?

Ralentización del *pipeline*

Ejemplo: predicción de salto estática

- ❑ Etapas: F, D, IS, X, M, W
- ❑ Cálculo dirección de salto: D
- ❑ Evaluación condición y carga del nuevo PC: X
- ❑ Comparar rendimiento predicción estática de salto no tomado con la de salto tomado

Salto NT	F	D	IS	X	M	W
PC+4		F	D	IS		
PC+8			F	D		
PC+12				F		



Salto T	F	D	IS	X	M	W
PC+4		F	D	IS		
PC+8			F	D		
Destino de salto				F	F	

Ejemplo: predicción de salto estática

- ❑ Si se hace predicción de salto tomado, en la etapa D ya conocemos la dirección destino del salto, por lo que podemos ir llenando el cauce con instrucciones de dicha dirección. Sabemos si hemos acertado o no en la etapa X, es entonces cuando sabremos si podemos continuar ejecutando las instrucciones destino o si tenemos que cargar el cauce con instrucciones de PC+4.

Salto NT	F	D	IS	X	M	W
Destino de salto		—	F	D		
Destino + 4				F		
PC + 4					F	

Salto T	F	D	IS	X	M	W
Destino de salto		—	F	D		
Destino + 4				F		
Destino + 8						

Ejemplo: predicción de salto estática

Tipo de predicción	Penalización en acierto	Penalización en fallo
Salto no tomado	0	3
Salto tomado	1	3

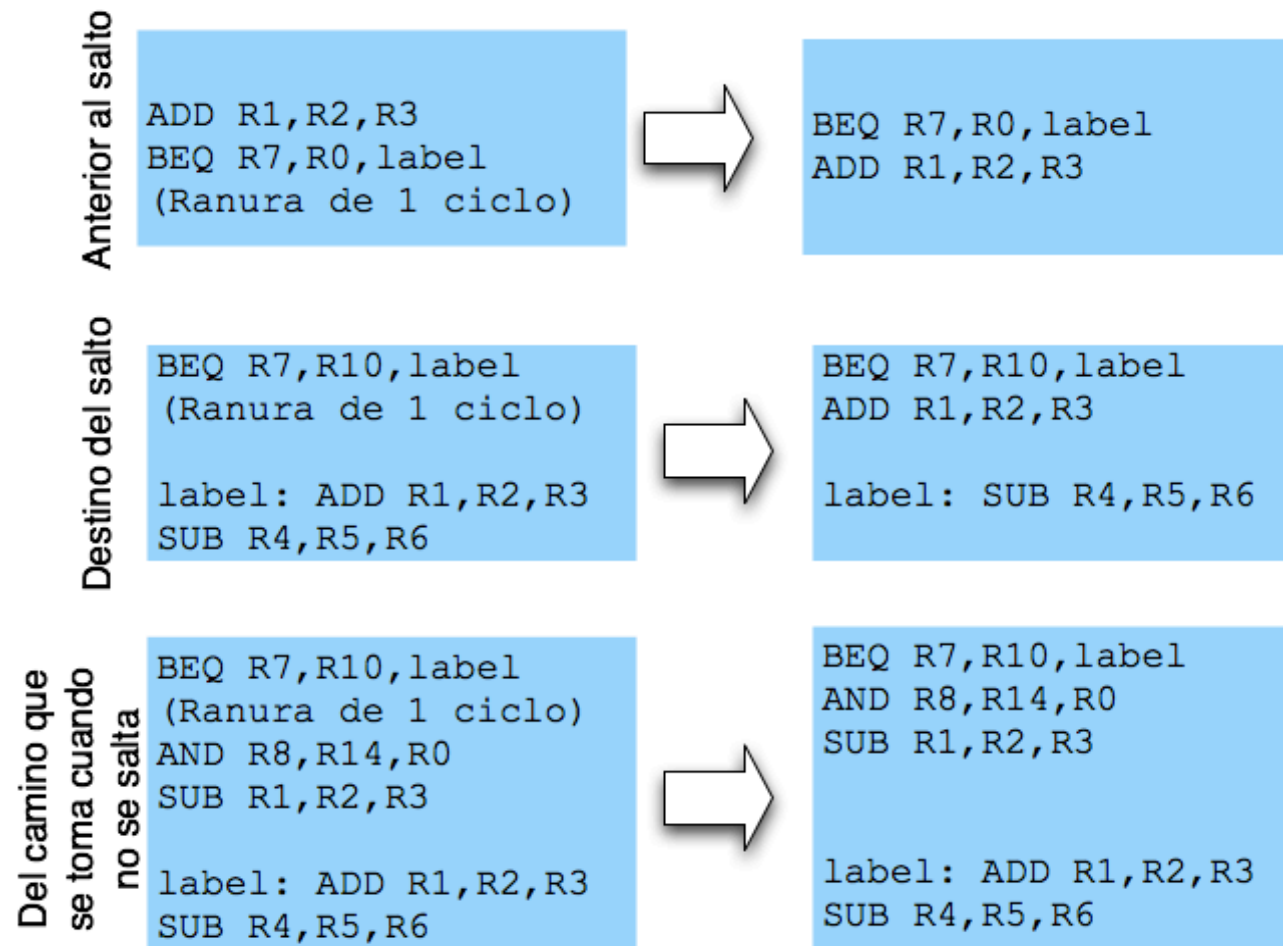
- ❑ Si el 75% de los saltos se toma, el 25% no se toma
- ❑ Penalización media por riesgos de control utilizando salto no tomado $0,75 \cdot 3 = \mathbf{2,25 \text{ ciclos}}$
- ❑ Penalización media por riesgos de control utilizando salto tomado $0,75 \cdot 1 + 0,25 \cdot 3 = \mathbf{1,5 \text{ ciclos}}$

Solución software: salto retardado, relleno de ranura

- ☐ Conocido el retardo en las bifurcaciones (**ranuras**), se emplea dicho tiempo en la ejecución de instrucciones que se van a ejecutar en cualquier caso
- ☐ nanoMIPS no modificado, ranura de 3 ciclos
- ☐ nanoMIPS modificado, ranura de 1 ciclo
 - ☐ Instrucción anterior al salto
 - ☐ Instrucción destino del salto
 - ☐ Instrucción del camino que se sigue cuando el salto no se toma

Ralentización del *pipeline*

Riesgos de control: Solución software, salto retardado, relleno de ranura



Ejemplo: relleno de ranura

- ❑ Utilización de la técnica de salto retardado
- ❑ nanoMIPS con adelantamiento y con ranura de salto de 1 ciclo

```
1          AND    R6,R4,R5
2          LW     R1,0(R10)
3          LW     R2,100(R10)
4          BEQ    R2,R0,salto
ranura de salto
5          SUB     R8,R1,R2
6          ADD     R2,R8,R2
7 salto:    SUB     R9,R11,R1
8          ADD     R12,R9,R1
```

Ejemplo: relleno de ranura

❑ Instrucción anterior al salto: AND R6,R4,R5

2		LW	R1, 0(R10)
3		LW	R2, 100(R10)
4		BEQ	R2, R0, salto
1		AND	R6, R4, R5
5		SUB	R8, R1, R2
6		ADD	R2, R8, R2
7	salto:	SUB	R9, R11, R1
8		ADD	R12, R9, R1

Ejemplo: relleno de ranura

❑ Instrucción destino del salto: SUB R9,R11,R1

```
1      AND    R6,R4,R5
2      LW     R1,0(R10)
3      LW     R2,100(R10)
4      BEQ    R2,R0,salto
7      SUB    R9,R11,R1
5      SUB    R8,R1,R2
6      ADD    R2,R8,R2
8 salto+4:  ADD    R12,R9,R1
```

Ejemplo: relleno de ranura

- ❑ Instrucción del camino que se sigue cuando el salto no se toma: SUB R8,R1,R2

1		AND	R6,R4,R5
2		LW	R1,0(R10)
3		LW	R2,100(R10)
4		BEQ	R2,R0,salto
5		SUB	R8,R1,R2
6		ADD	R2,R8,R2
7	salto:	SUB	R9,R11,R1
8		ADD	R12,R9,R1

Ejemplo: relleno de ranura

- ❑ Rellenar la ranura del siguiente código

```
1          AND    R1,R4,R5
2          LW     R1,0(R10)
3          LW     R2,100(R10)
4          BEQ    R2,R1,salto
ranura de salto
5          SUB     R8,R1,R2
6          ADD     R2,R8,R9
7  salto:  SUB     R9,R8,R1
8          ADD     R12,R9,R1
```

Excepciones en el nanoMIPS segmentado

- ❑ Métodos utilizados para indicar la causa de una excepción:
 - ❑ **Registro de excepción**
 - ❑ **Vector de interrupciones:** la dirección a la cual se transfiere el control viene determinada por el tipo de excepción

Tipo de excepción	Dirección vector de excepción
Instrucción no definida	0X8000 0000
Desbordamiento aritmético	0X8000 0180

Excepciones en el nanoMIPS segmentado

- ❑ Las excepciones se tratan como otro tipo de riesgo de control. Por ejemplo, si se produce un desbordamiento en una instrucción de suma:
 1. Se debe vaciar el pipeline de instrucciones posteriores a la suma
 2. Se debe comenzar la carga de instrucciones de la nueva dirección
- ❑ La ruta de datos se debe adaptar para soportar excepciones
 - ❑ Señal para seleccionar el nuevo valor del PC
 - ❑ Registro de excepción para almacenar la causa de la excepción, *Cause Register*
 - ❑ Registro para almacenar la instrucción que provocó la excepción, *EPC*

Trabajo conjunto del SSOO y del hardware

- ☐ Parte correspondiente al hardware:
 - ☐ *Detener la instrucción que provocó el fallo*
 - ☐ *Completar las instrucciones anteriores*
 - ☐ *Vaciar el pipeline de instrucciones posteriores*
 - ☐ *Guardar la causa de la excepción*
 - ☐ *Salvar la dirección de la instrucción que provocó la excepción*
 - ☐ *Saltar a una dirección predeterminada*
- ☐ Al sistema operativo le corresponde analizar la causa de la excepción y actuar en consecuencia:
 - ☐ Cuando se trata de una instrucción indefinida, un fallo hardware o un desbordamiento, el ssoo *mata el proceso y retorna el motivo*
 - ☐ Cuando se trata de una petición de E/S o de una llamada al sistema, el ssoo *salva el estado del programa (cambio de contexto), realiza la tarea en cuestión y, en el futuro, restaura el proceso para que continúe su ejecución*
- ☐ Uno de los usos más frecuentes de excepciones es el manejo de fallos de página y excepciones del TLB