Tema 6. Introducción a la segmentación avanzada: Riesgos

Organización de Computadores

LUIS ENRIQUE MORENO LORENTE
RAÚL PÉRULA MARTÍNEZ
ALBERTO BRUNETE GONZALEZ
DOMINGO MIGUEL GUINEA GARCIA ALEGRE
CESAR AUGUSTO ARISMENDI GUTIERREZ
JOSE CARLOS CASTILLO MONTOYA

Departamento de Ingeniería de Sistemas y Automática

PRESTACIONES DE UNA UNIDAD SEGMENTADA

- El speedup ideal de una unidad segmentada es el número de etapas en el cauce
- Como se pueden alcanzar estas prestaciones?
  - Existen problemas que limitan las prestaciones alcanzables entre ellos tenemos:
    - Operaciones cuyos tiempos de ejecución difieren mucho: un add de una mult
    - Falta de recursos hardware
    - Datos que no están disponibles cuando se requieren

RIESGOS EN UNA UNIDAD SEGMENTADA

- Uno de los límites en la segmentación son los riesgos.

  Los riesgos impiden que la siguiente instrucción se ejecute en el ciclo de reloj que le corresponde. Pueden ser de tres tipos:

  - Riesgos de datos: la instrucción depende de resultados de instrucciones previas que aún están en el cauce
  - Riesgos estructurales: el HW no puede soportar esta combinación de instrucciones
  - Riesgos de control: la segmentación de instrucciones de salto condicional y de otros tipos de instrucciones que cambian el PC

- La solución más común a estos problemas consiste en detener el cauce hasta que el riesgo desaparece, esto provoca la inserción de burbujas en el funcionamiento del cauce

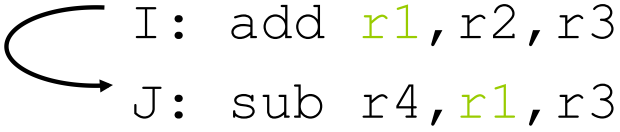
RIESGOS DE DATOS

Instr1 seguida de Instr2

- Riesgo Read After Write (RAW)
  - Instr2 trata de leer un operando fuente antes de que la instr1 lo escriba
- Riesgo Write After Write (WAW)
  - Instr2 trata de escribir un operando antes de que la instr1 lo escriba
  - Si se escribe en más de una etapa, cuando se detiene la primera escritura y se permite la segunda
- Riesgo Write After Read (WAR)
  - Instr2 trata de escribir un resultado antes de que la instr1 lo lea
  - Instrucciones que escriben resultados al comienzo del cauce con otras que leen un operando fuente tarde dentro del cauce

RIESGOS DE DATOS: READ AFTER WRITE (RAW)

- Instr J trata de leer un operando antes de que la Instr I lo escriba

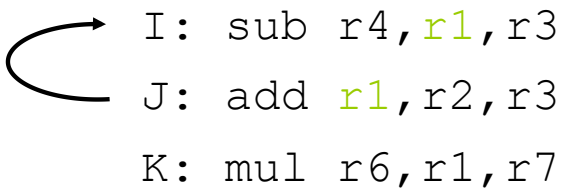


- Este problema se produce por una "Dependencia" (en terminología del compilador). Este riesgo es el resultado de de una necesidad real de comunicación de un dato.

- "True" dependence

RIESGOS DE DATOS: WRITE AFTER READ (WAR)

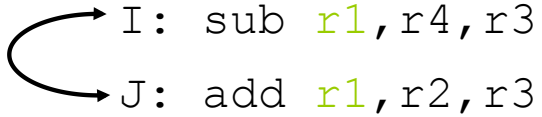
- La Instr J escribe el operando antes de que la Instr I lo lea



- Los diseñadores de compiladores denominan a este tipo de dependencia, "anti-dependencia" . Se produce como resultado de la reutilización del registro "r1".
- En el caso de la arquitectura DLX con 5 etapas en el cauce no se puede producir ya que:
  - Todas las instrucciones tienen 5 etapas,
  - Las lecturas son siempre en la etapa 2 (ID),
  - Las escrituras en la etapa 5 (WB)
  - También llamada "Name dependence"

RIESGOS DE DATOS: WRITE AFTER WRITE (WAW)

- $Instr_J$ escribe el operando antes de que la $Instr_I$ escribe en reg.



```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Los diseñadores de compiladores denominan a este tipo de dependencia, "dependencia de salida" .
  Se produce también como resultado de la reutilización del nombre "r1".
- En el caso de la arquitectura DLX con 5 etapas en el cauce no se puede producir ya que :
  - Todas las instrucciones son de 5 etapas, y
  - Las escrituras se realizan siempre en la etapa 5

- También llamada "Name dependence"

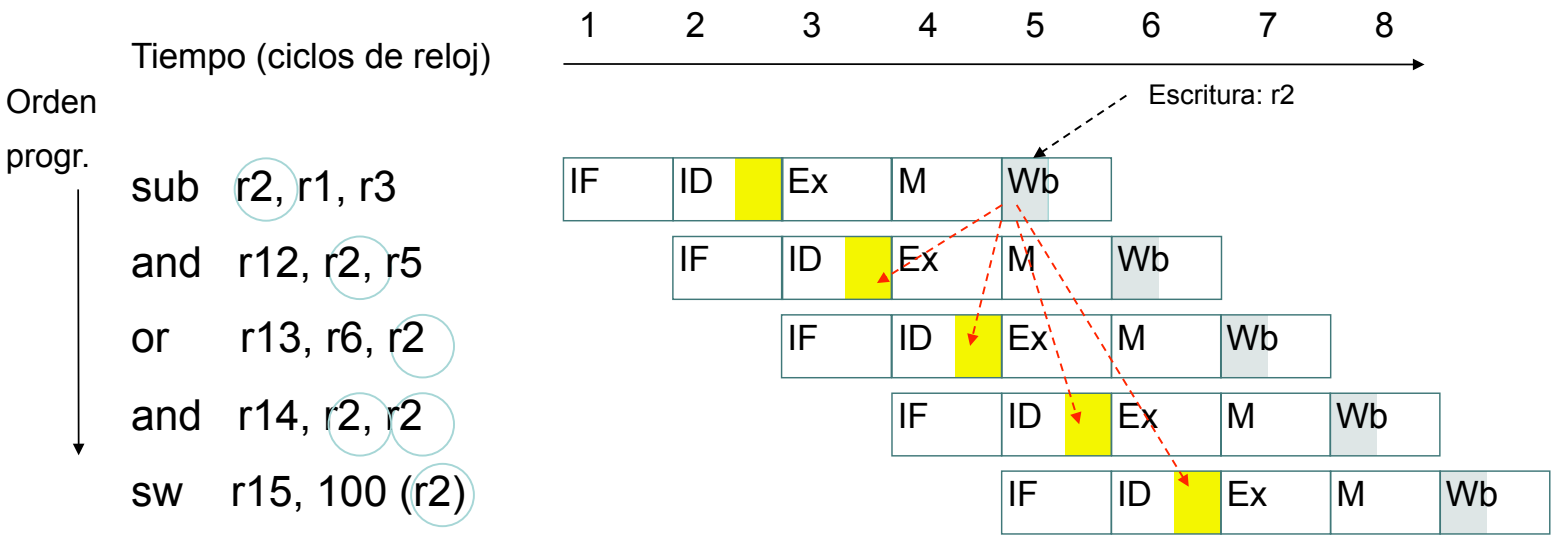
ILP (PARALELISMO A NIVEL DE INSTRUCCIÓN) Y RIESGOS DE DATOS

- EL HW/SW deben preservar el orden del programa:
  el orden en el que las instrucciones deben ejecutarse si se ejecutan secuencialmente una cada vez, viene determinado por su posición en el programa original
- Objetivo del HW/SW: explotar el paralelismo preservando el orden del programa, solo donde este afecte a la salida del programa.
- Las instrucciones implicadas en dependencias de nombre pueden ejecutarse simultáneamente si el nombre usado en la instrucción es modificado para que no exista conflicto
  - Renombrado de registros resuelve la dependencias de nombre producidas por la reutilización de los registros
  - Esto puede hacerse por compilador o por HW

RIESGOS DE DATOS

Problemas con el comienzo de las siguientes instrucciones antes de que termine la primera

Las dependencias que retroceden en el tiempo son de “datos”



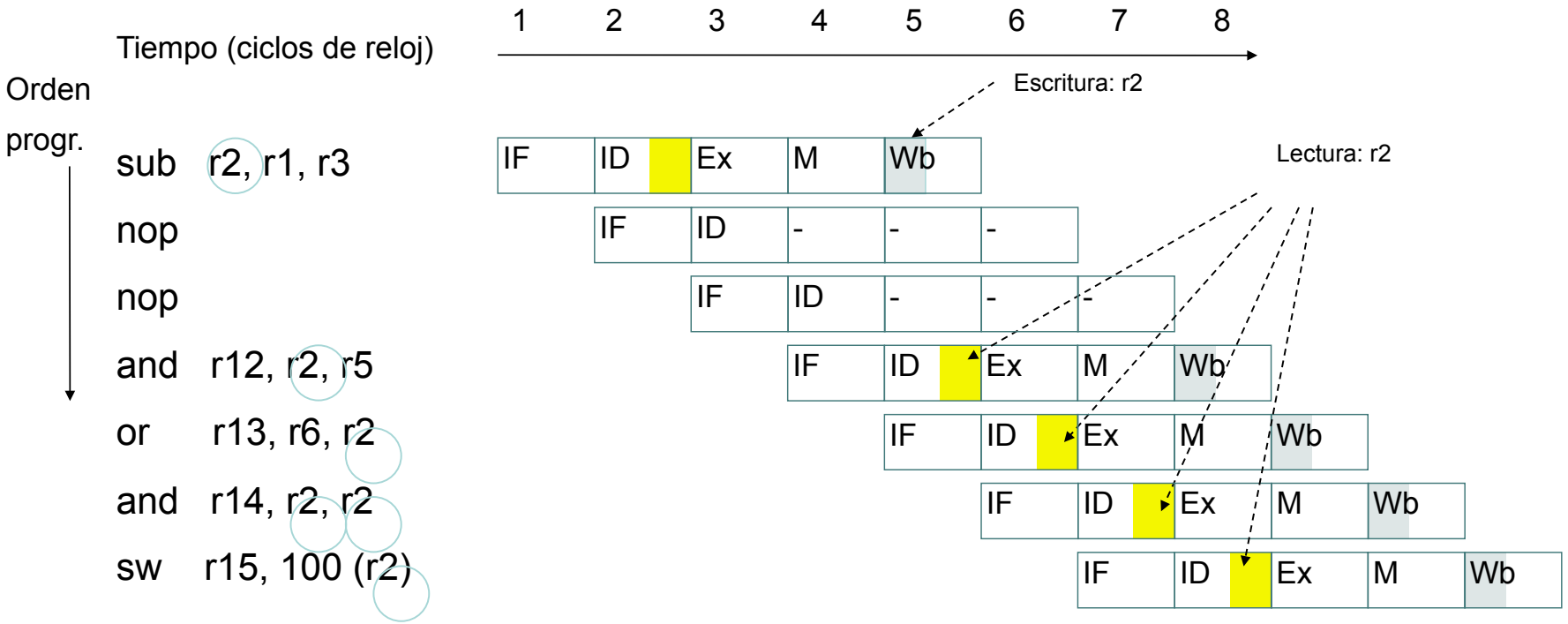
¿Cuales son lecturas incorrectas del reg r2?

SOLUCIONES A LOS RIESGOS DE DATOS

- Los riesgos de datos pueden tratarse de diversas formas:
  - Software: si el HW del proc. segmentado no es capaz de detectar y resolver los riesgos, esto tiene que resolverse en el compilador.
    - El compilador introduce instr de no-op después de cada instrucción que cause o pueda causar riesgos. Evita el malfuncionamiento, pero no reduce los retardos.
    - El compilador puede reordenar el código del programa (planificación de instr) para mejorar el funcionamiento (se uso este enfoque en los primeros RISC).
  - Hardware: Se pueden distinguir tres soluciones:
    - Interbloqueo. Es la forma más simple, y consiste en detectar el riesgo y detener el cauce hasta que este desaparezca. Esta solución provoca burbujas en el cauce.
    - Forwarding (anticipación). Es más sofisticada, requiere hardware adicional. Consiste en no esperar a leer los datos hasta el momento en que se escriban en estos, sino leerlo en cuanto que esta disponible.
    - Como el forwarding no es capaz de resolver todos los riesgos de datos, hay que combinarlo con el interbloqueo.

RIESGOS RAW: SOLUCIÓN SW

- Introducir instrucciones de NOP (el compilador)

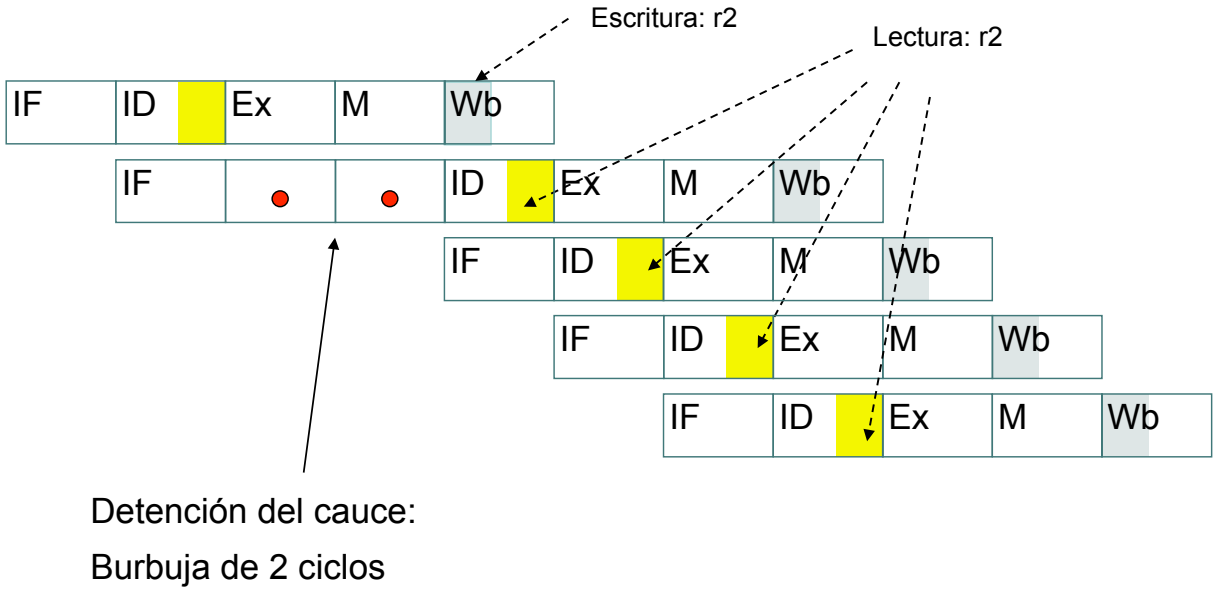


Problema: esto ralentiza mucho el funcionamiento

RIESGOS RAW: SOLUCIÓN HW INTERBLOQUEO

- Diseñar un HW que detecte los riesgos y detenga el cauce si es necesario

```
sub   r2, r1, r3
and   r12, r2, r5
or    r13, r6, r2
and   r14, r2, r2
sw    r15, 100 (r2)
```

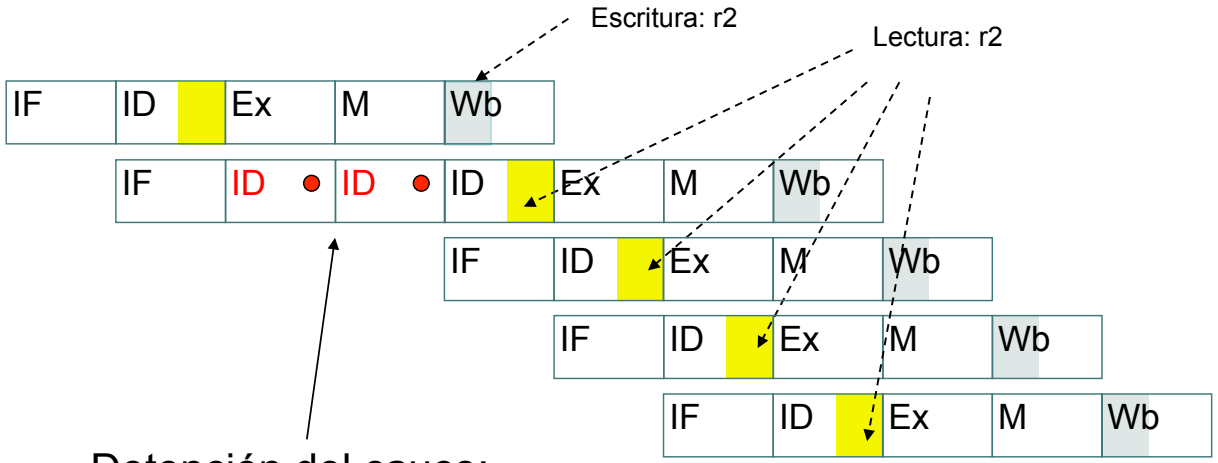


Problema: esto ralentiza mucho el funcionamiento

BURBUJAS, DETENCIONES (STALLINGS)

Es posible detener el cauce manteniendo una instrucción en la misma etapa

```
sub   r2, r1, r3
and   r12, r2, r5
or    r13, r6, r2
and   r14, r2, r2
sw    r15, 100 (r2)
```



Detención del cauce:

Burbuja de 2 ciclos

Como la etapa no puede finalizarse se mantiene la misma instrucción hasta que se completa

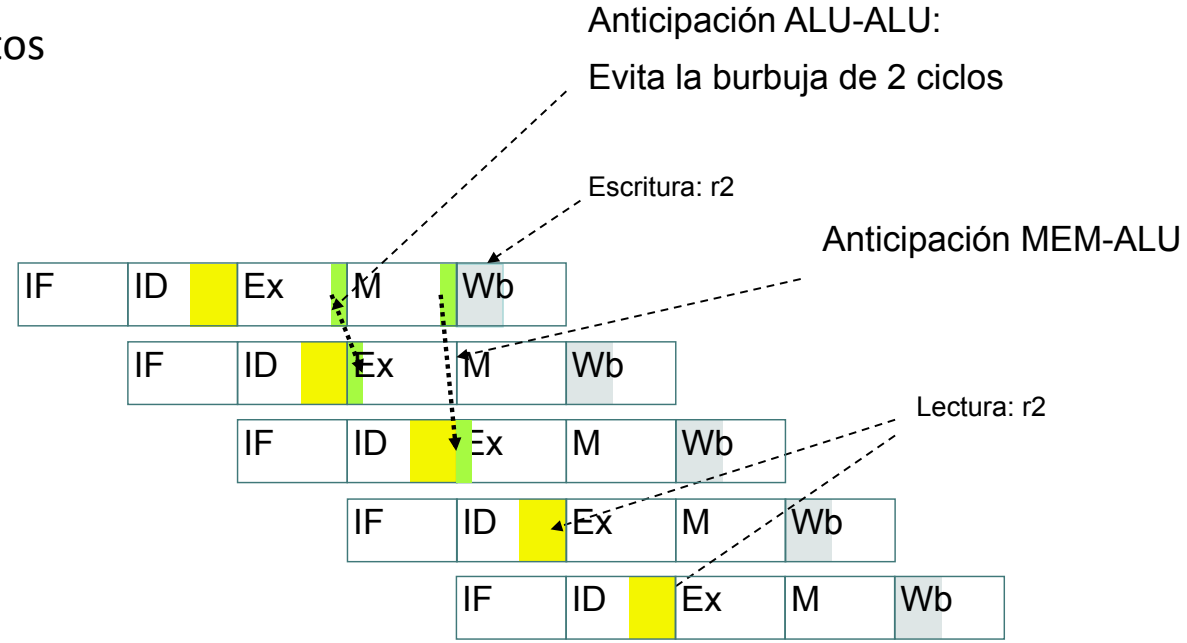
RIESGOS RAW: SOLUCIÓN HW FORWARDING (ANTICIPACIÓN)

- Utilizar los resultados cuando están disponibles, antes de que se escriban en el banco de registros
- Anticipación al banco de registros para permitir leer/escribir en el mismo registro
- Anticipación a la ALU, para adelantar una operación en la ALU que de otro modo tendría que esperar a que el dato se escribiera en el banco de registros y luego leerlo.
- Anticipación ALU-ALU: desde la salida de la ALU a alguno de los registros de entrada de la ALU.
- Anticipación MEM-ALU: desde la etapa MEM se anticipa el dato leído de la cache de datos a la entrada de la ALU.

ANTICIPACIÓN: EJEMPLO

- Anticipación de datos

```
sub   r2, r1, r3
and   r12, r2, r5
or    r13, r6, r2
and   r14, r2, r2
sw    r15, 100 (r2)
```



HW PARA FORWARDING (ANTICIPACIÓN)

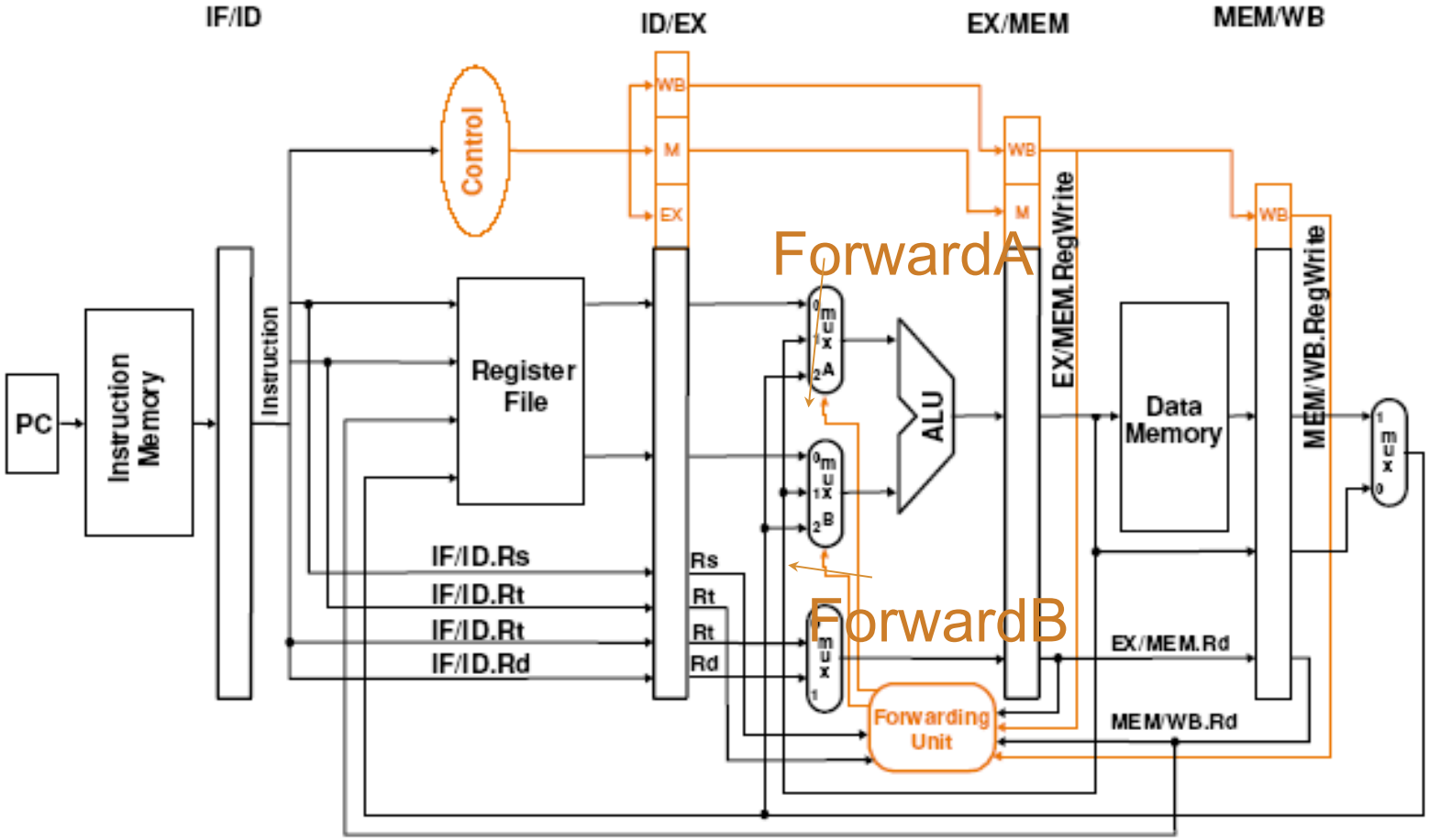


Imagen de Patterson, 1997

HW PARA FORWARDING (ANTICIPACIÓN)

- Riesgo en EX
  - If (EX/MEM.RegWrite and
    (EX/MEM.Rd = ID/EX.Rs)) then ForwardA=01
  - If (EX/MEM.RegWrite and
    (EX/MEM.Rd = ID/EX.Rt)) then ForwardB=01
- Riesgo en MEM
  - if (MEM/WB.RegWrite
    and !(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) then ForwardA=10
  - if (MEM/WB.RegWrite
    and !(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
    then ForwardB=10

EJEMPLO 1: FORWARDING POR HW

Forwarding: Ex -> Scr1 y WB -> Scr2

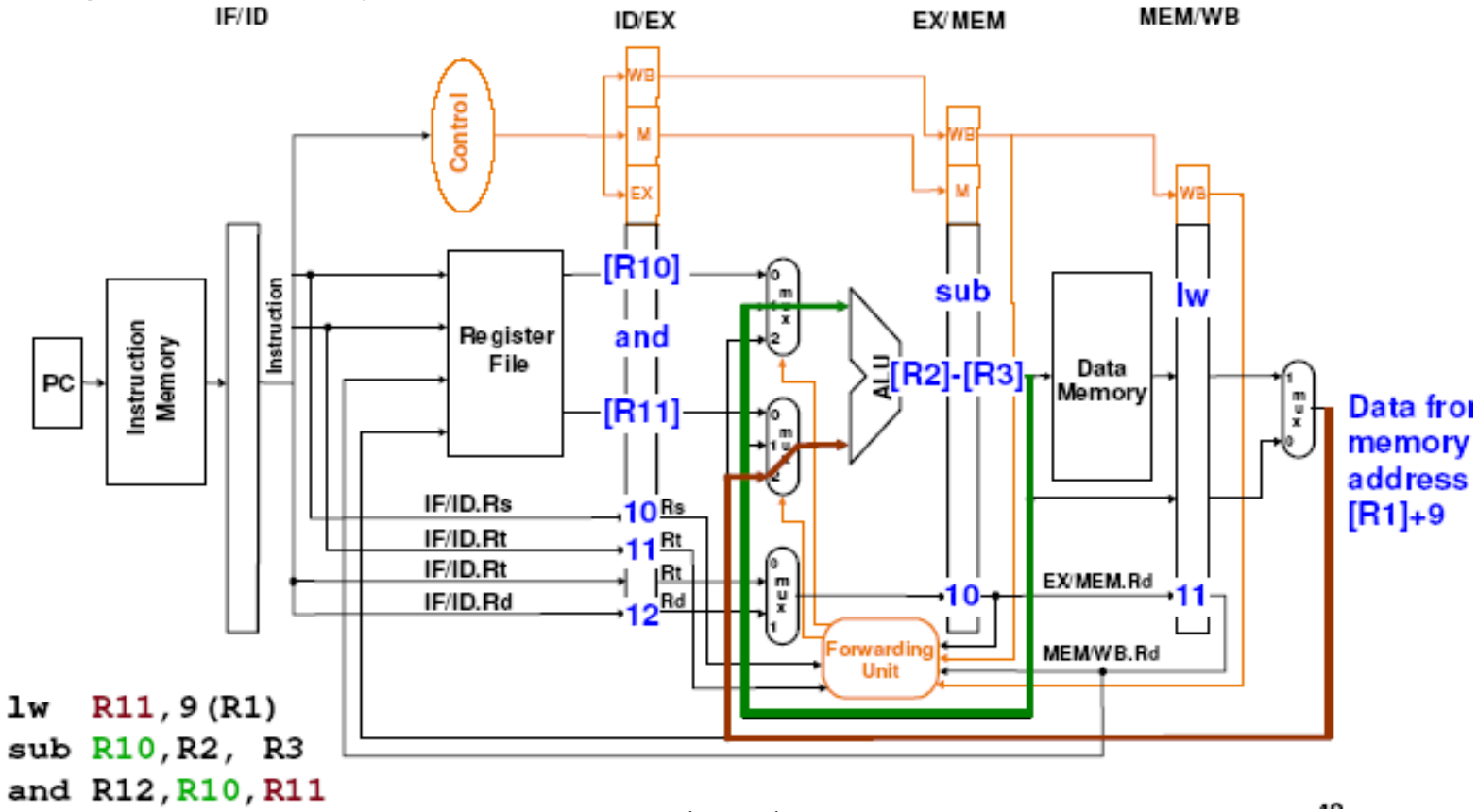


Imagen de Patterson, 1997

EJEMPLO 2: FORWARDING POR HW

Forwarding: WB -> Scr2

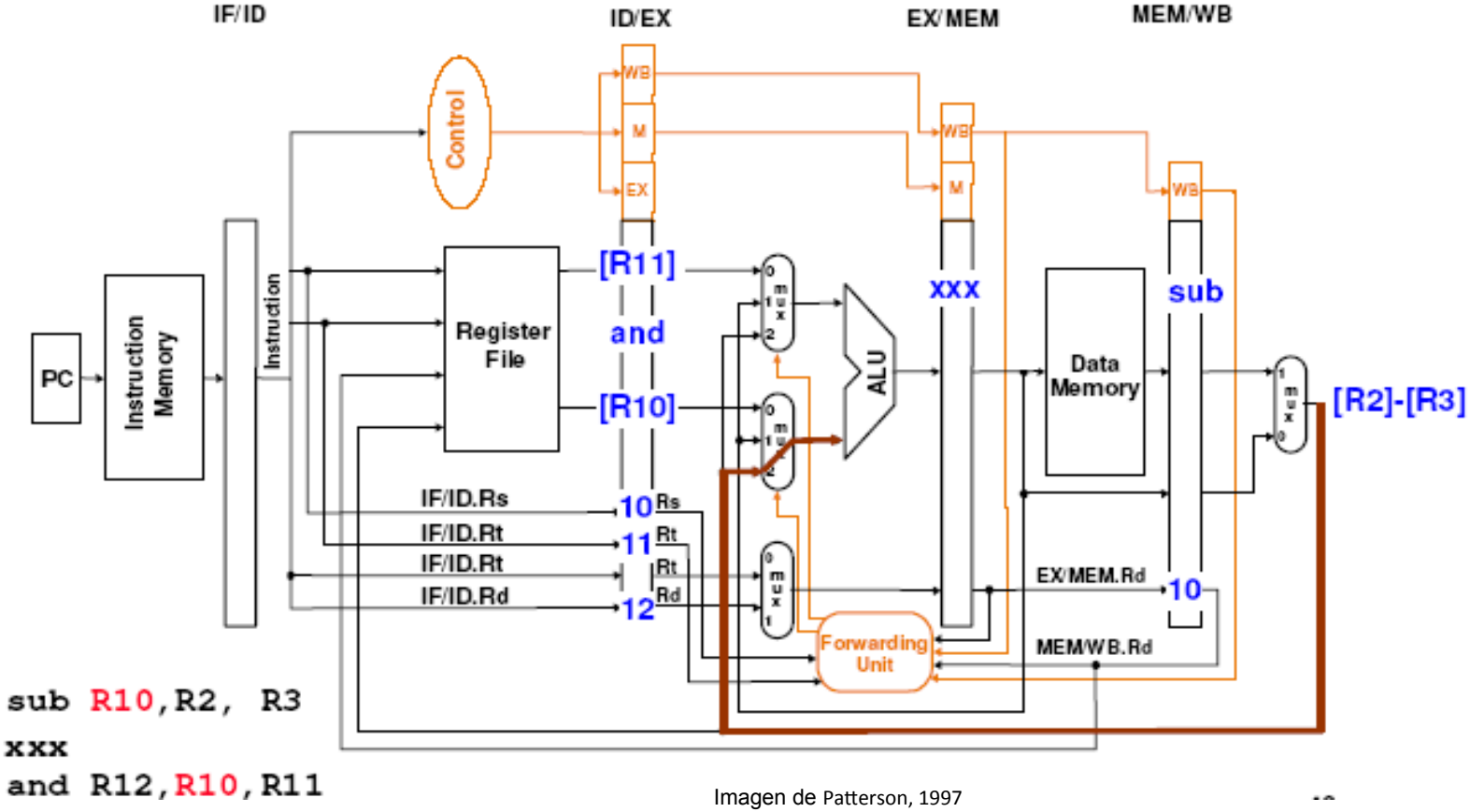


Imagen de Patterson, 1997

NO SIEMPRE ES POSIBLE EL FORWARDING

Una instr trata de leer un registro después de un load que va a escribir en ese mismo registro.

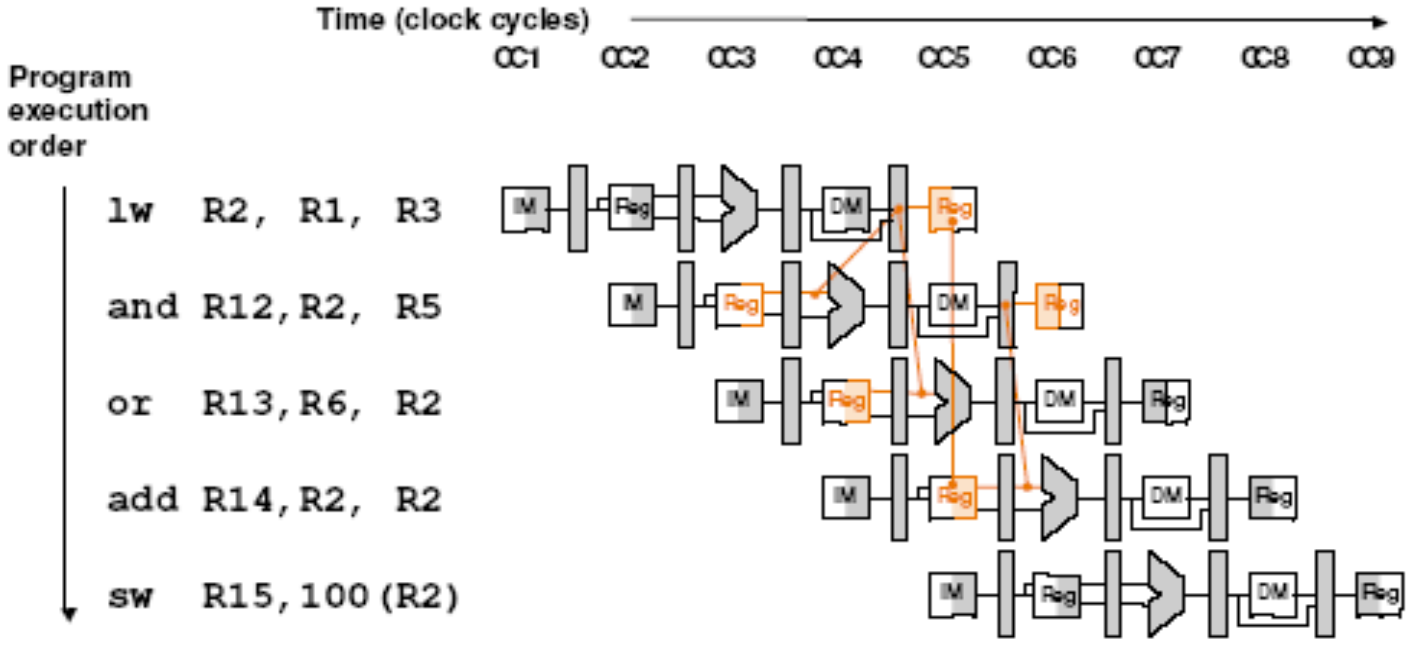


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

Es necesaria una unidad de detección de riesgos que detenga la instr posterior al load.

FORWARDING + UNIDAD DE DETECCIÓN DE RIESGOS

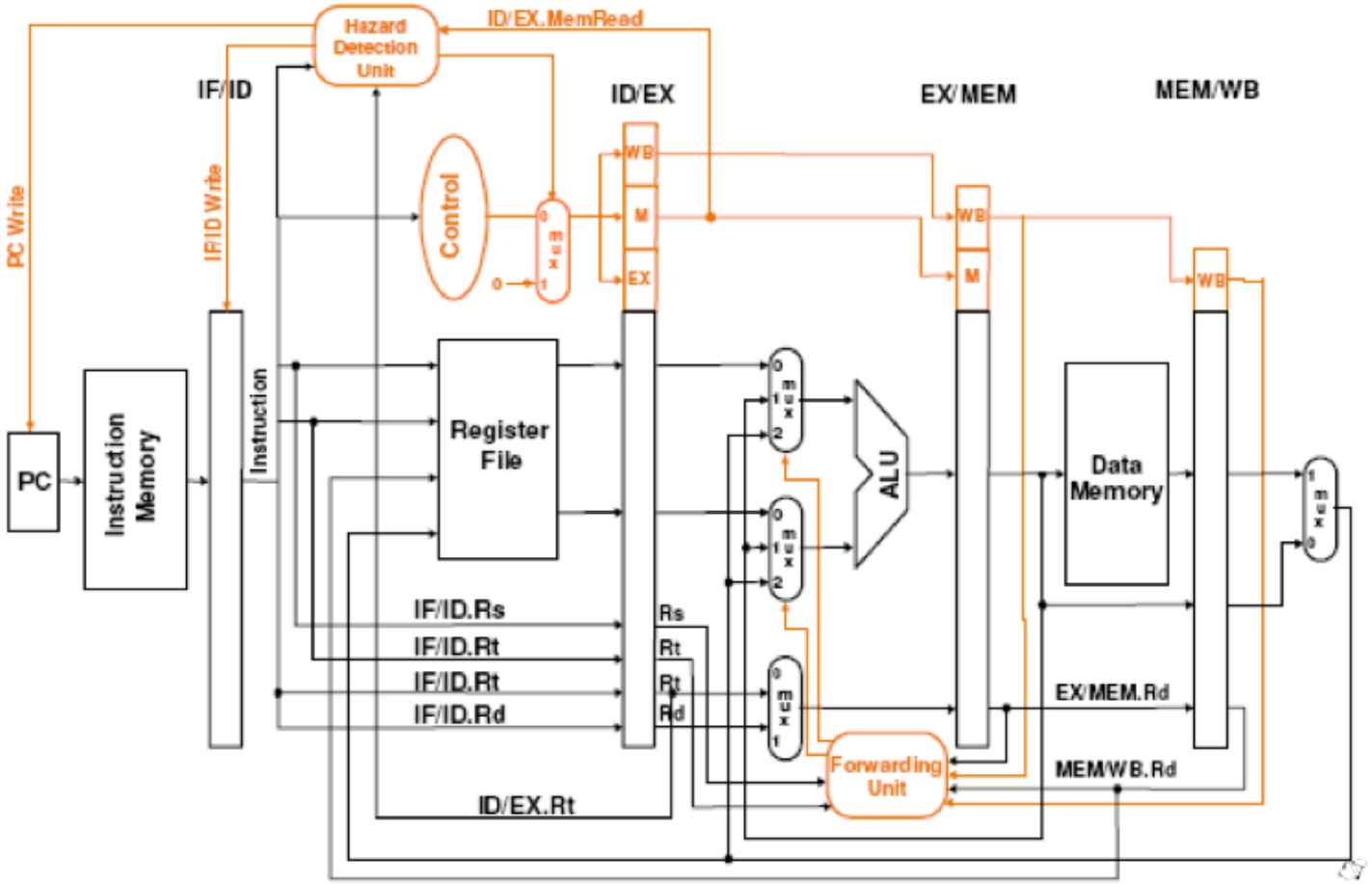


Imagen de Patterson, 1997

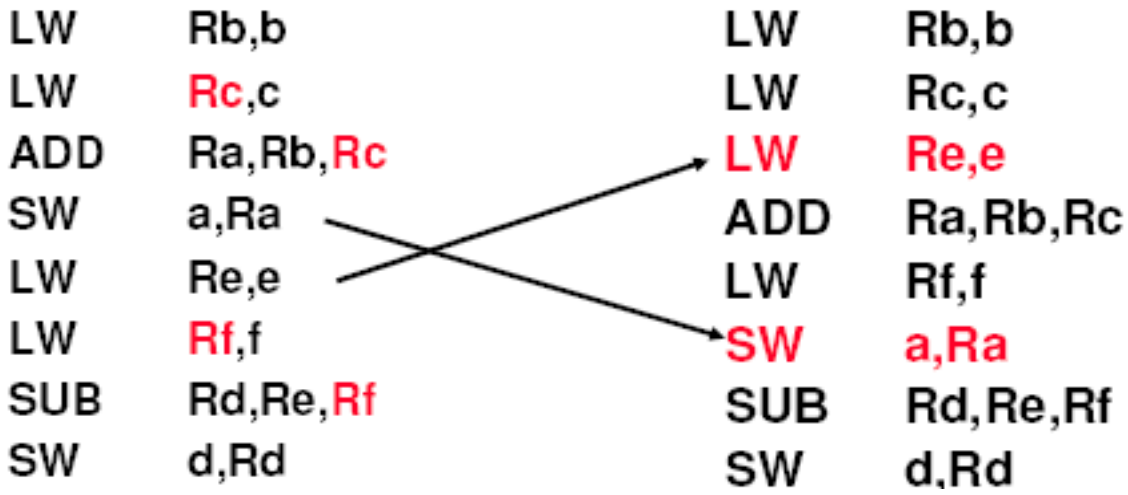
PLANIFICACIÓN DEL CÓDIGO PARA EVITAR RIESGOS POR LOAD

Tratemos de generar un código más rápido para la operación

a=b+c;

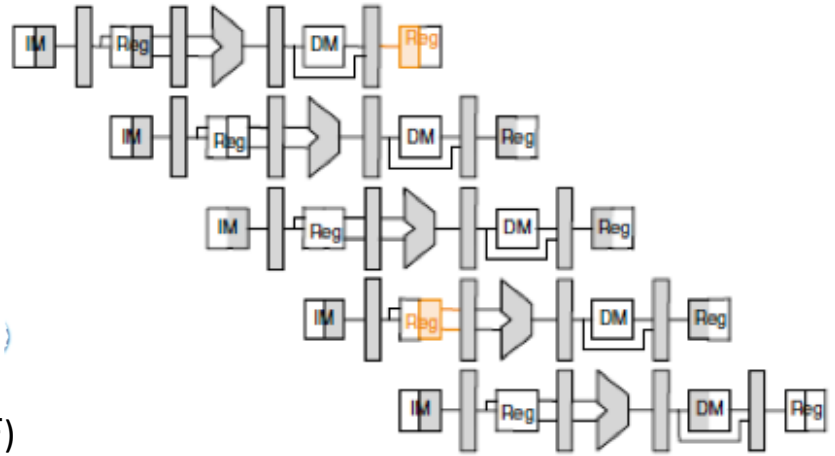
d=e-f;

suponiendo que a,b,c,d,e y f están en la memoria.



RIESGOS ESTRUCTURALES

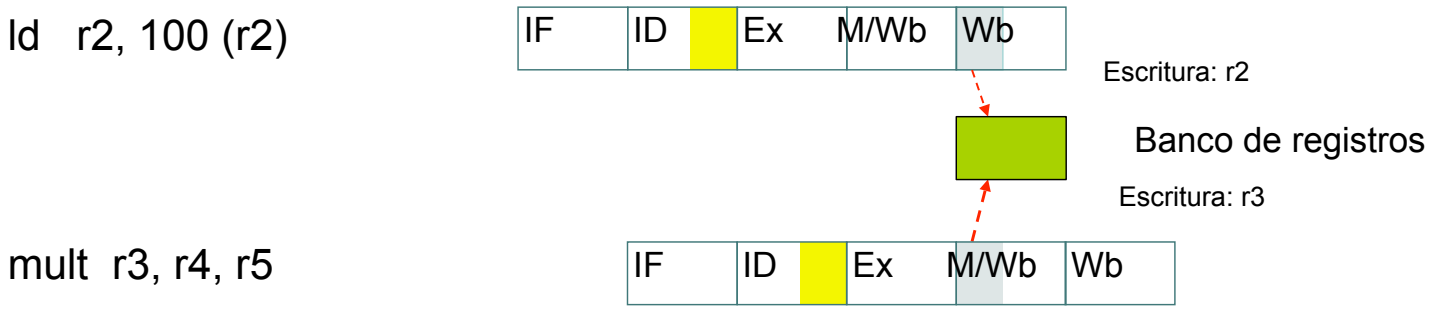
- Se producen por intentar utilizar simultáneamente un cierto recurso dos instrucciones en ejecución
- Banco de registros:
  - Acceso al banco de registros en 2 etapas:
    - Lectura durante la etapa 2 (ID)
    - Escritura durante la etapa 5 (Wb)
  - Solución: dos puertos de lectura y uno de escritura
- Memoria:
  - Se puede acceder a la memoria en dos etapas:
    - Búsqueda de instrucción durante la etapa 1 (IF)
    - Lectura/Escritura de datos durante la etapa 4 (Mem)
  - Solución: dos caches separadas para instrucciones y datos



- Cada unidad funcional puede ser usada una sola vez por instrucción
- Cada unidad funcional debe ser usada en la misma etapa por todas las instrucciones.

RIESGOS ESTRUCTURALES

- Se producen por un conflicto en el uso de los recursos
  - En el cauce planteado, si una instr reg-reg pudiese escribir el resultado de salida de la ALU en la etapa MEM y se produce una secuencia load seguida de una instr reg-reg entonces tendríamos dos accesos simultáneos al banco de registros

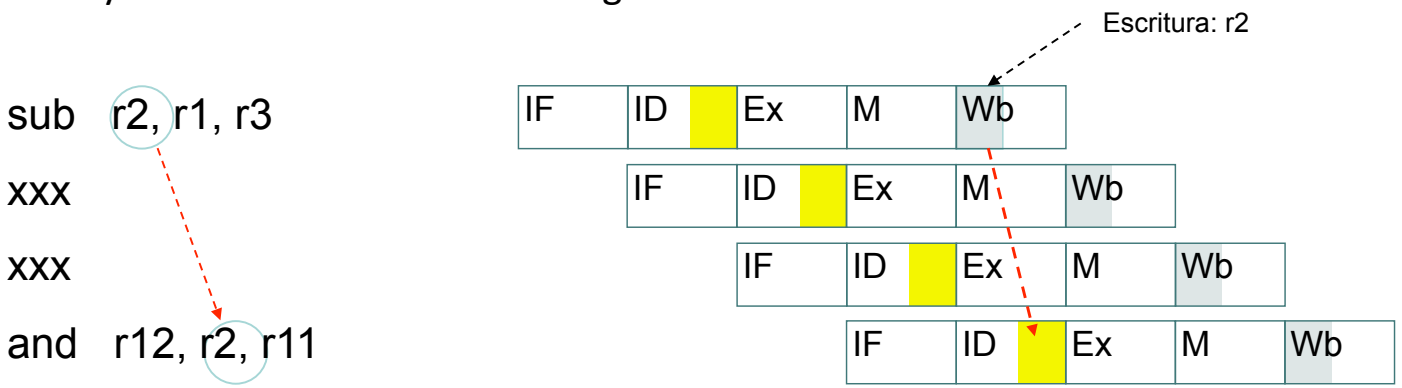


RIESGOS ESTRUCTURALES

- Soluciones hardware:
  - Arbitraje con interbloqueo: se añade hW que arbitre el conflicto e interbloquee a una de las instrucciones. Aparecen burbujas (en este caso de un ciclo).
  - Replicación de recursos hW: En nuestro ejemplo, si el banco de registros tuviese dos puertos de escritura el problema se resolvería (salvo que escriban exactamente en el mismo registro, pero este caso puede eliminarse en la compilación)

RIESGOS ESTRUCTURALES

- En el cauce que se ha diseñado no existen problemas de riesgo estructural.
  - Esto se debe a que el banco de registros se escribe en la primera mitad de los ciclos WB y la lectura se realiza en la segunda mitad de los ciclos ID.



• Las escrituras en el banco de registros se realizan en la primera mitad del ciclo

• Las lecturas en el banco de registros se realizan en la segunda mitad del ciclo

=> el banco es escrito antes que leído => devuelve el dato correcto

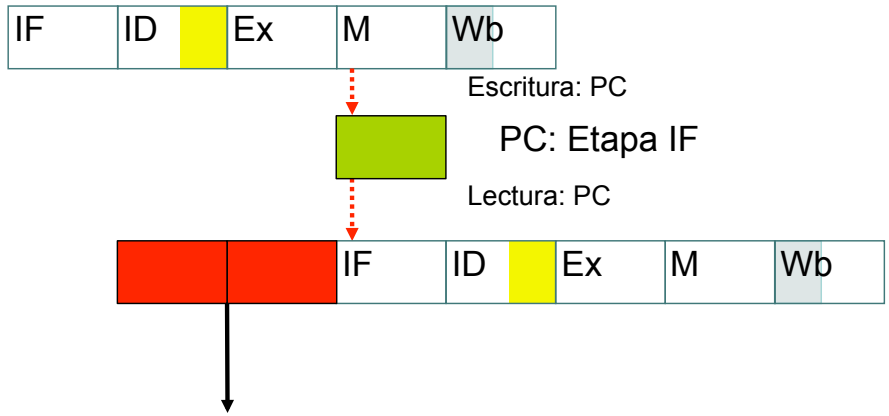
RIESGOS DE CONTROL

- Se originan en las instrucciones de salto condicional (branch) o incondicional (jmp).
  - Si suponemos que la instr 1 es un jmp, y la dirección de salto se calcula en la fase EX y se reemplaza el PC en la fase MEM, en el cauce han entrado una instr 2 (fase EX), una instr 3 (fase ID) y una instr 4 (fase IF).
    - Salto incondicional: las instrucciones deben ser canceladas y deberá buscarse la instr situada en la dirección de salto.
    - Salto condicional: sólo es necesario vaciar si el salto se efectúa.
  - A estas instrucciones que siguen a la instr 1 se les denomina delay slot o ventana de retardo. Que en este caso es de tres ciclos.

RIESGOS DE CONTROL: EJEMPLO

bnez r2, 100 (r3)

mult r3, r4, r5



Delay slot o ventana de retardo

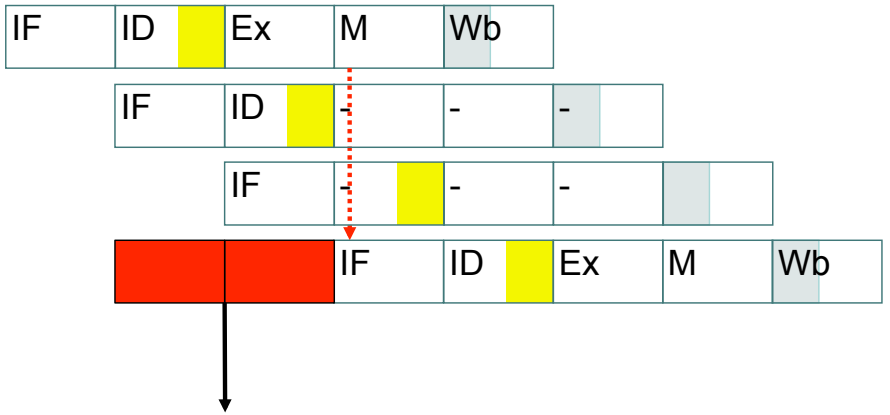
Burbuja de 2 o 3 ciclos según sea el diseño:

2→ si se escribe el PC al comienzo de la fase M y se empieza la búsqueda de la instr en el mismo ciclo.

3→ si se escribe el PC al final de la fase M y se empieza la búsqueda en el ciclo siguiente

RIESGOS DE CONTROL: VACIADO DEL CAUCE

```
bnez   r2, 100 (r3)
yyy1
yyy2
mult  r3, r4, r5
```



Delay slot o ventana de retardo

Si el cauce no se había detenido, y se estaba ejecutando yyy1 e yyy2 de forma especulativa, en caso de que el salto se efectúe es necesario vaciar el cauce.

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 1

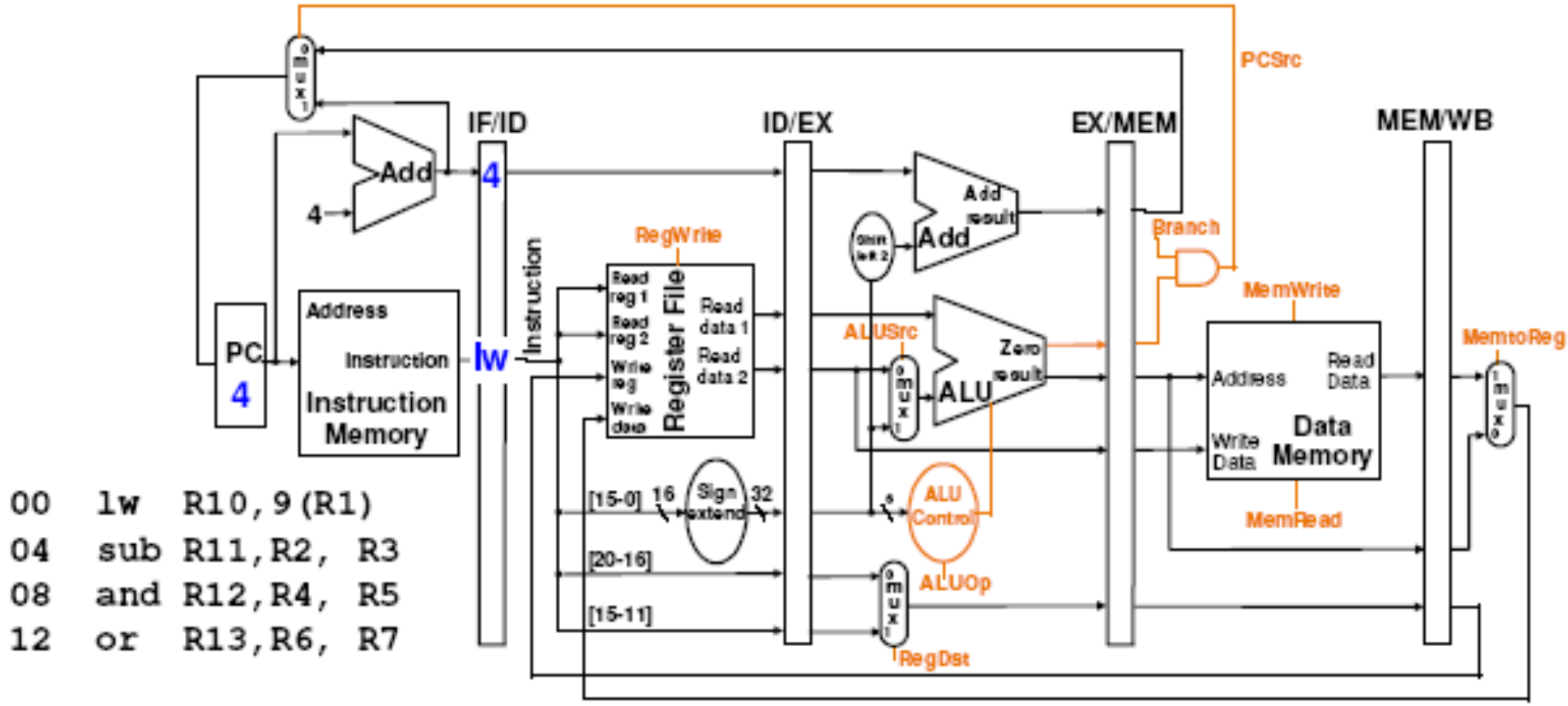


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 2

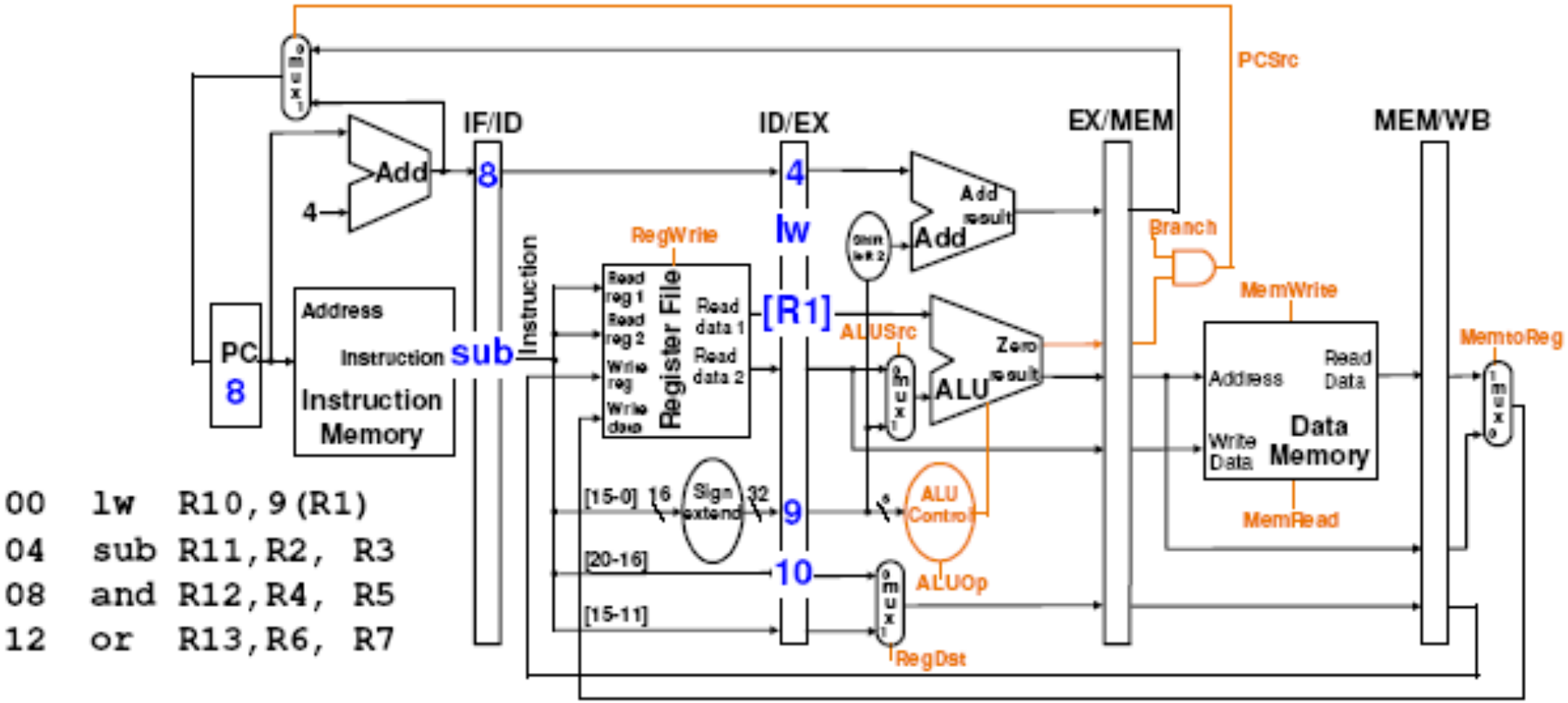


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 3

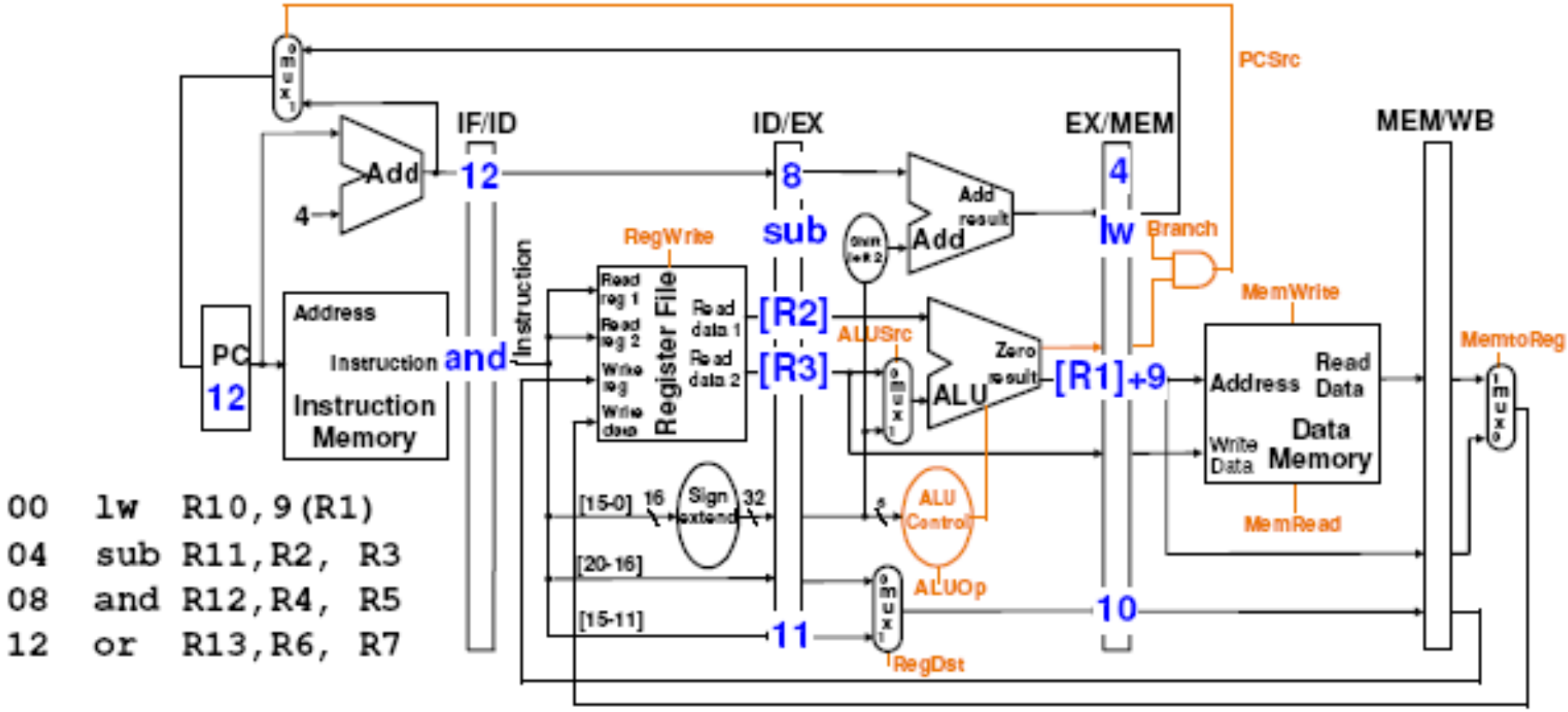


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 4

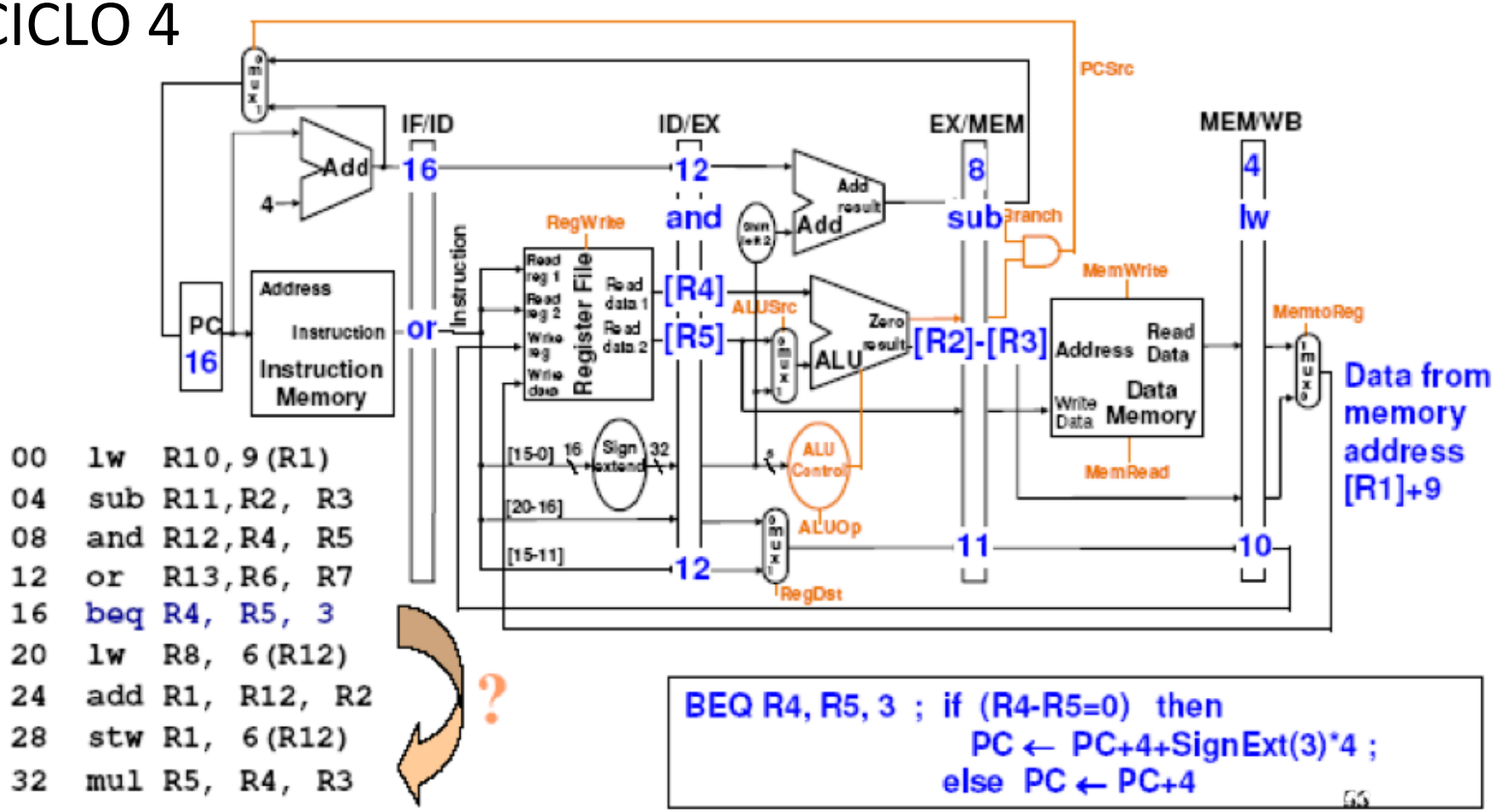


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 5

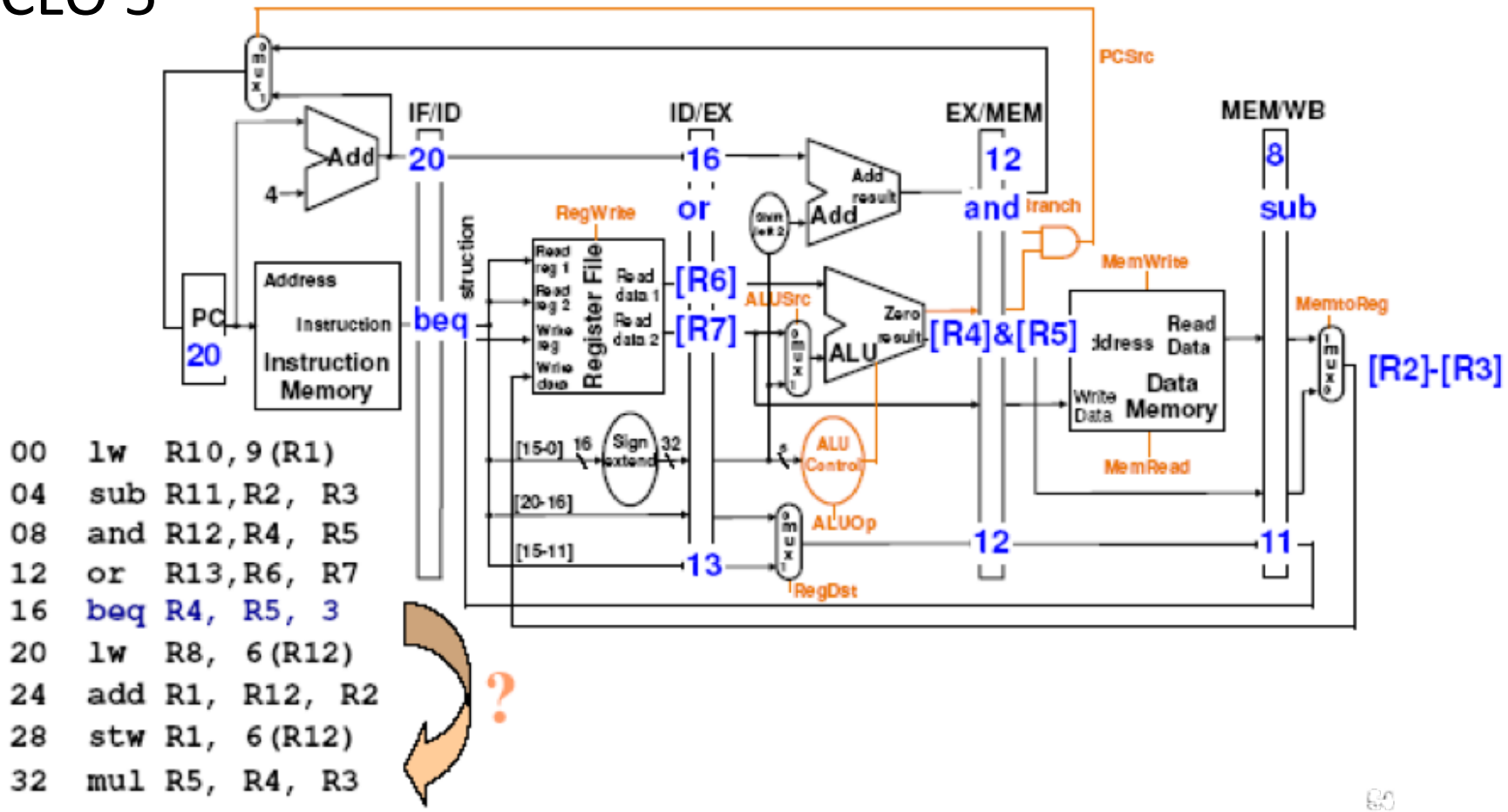


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 6

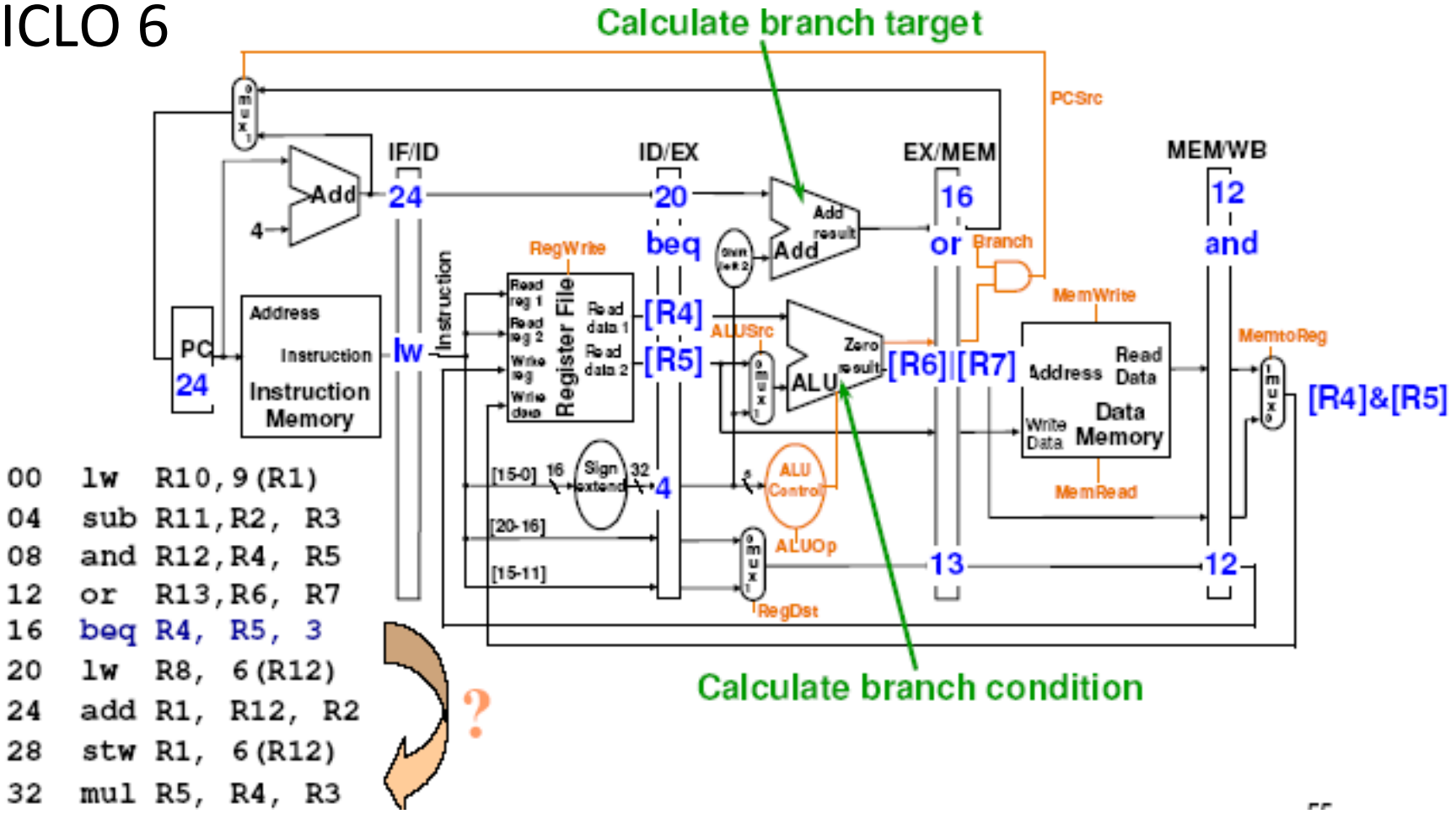


*Image from J. L. Hennesy and D. Patterson “Computer Organization and Design”

RIESGO PROVOCADO POR UN SALTO CONDICIONAL: CICLO 7

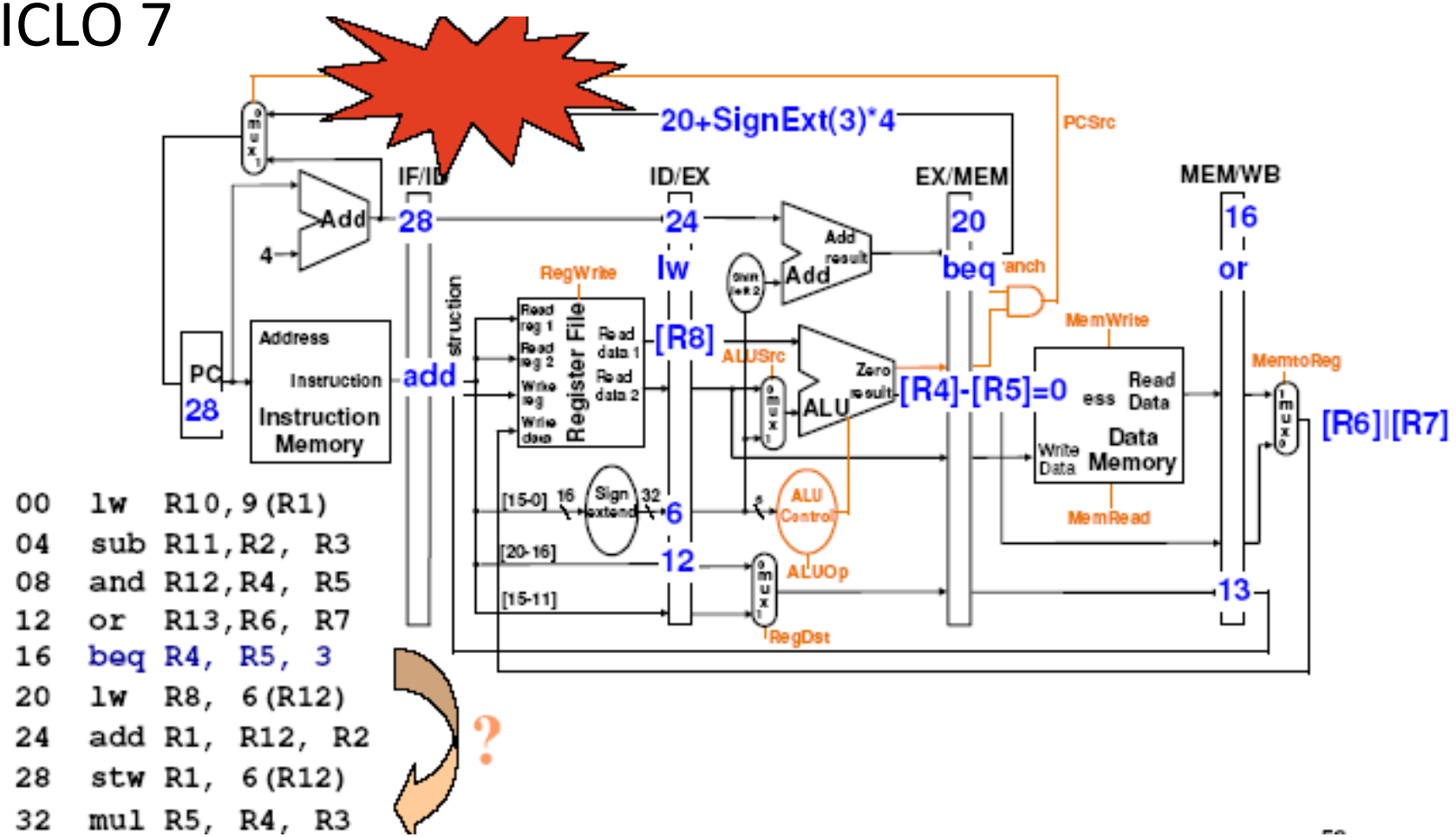


*Image from J. L. Hennesy and D. Patterson "Computer Organization and Design"

IMPACTO DE UN SALTO EN LAS DETENCIONES

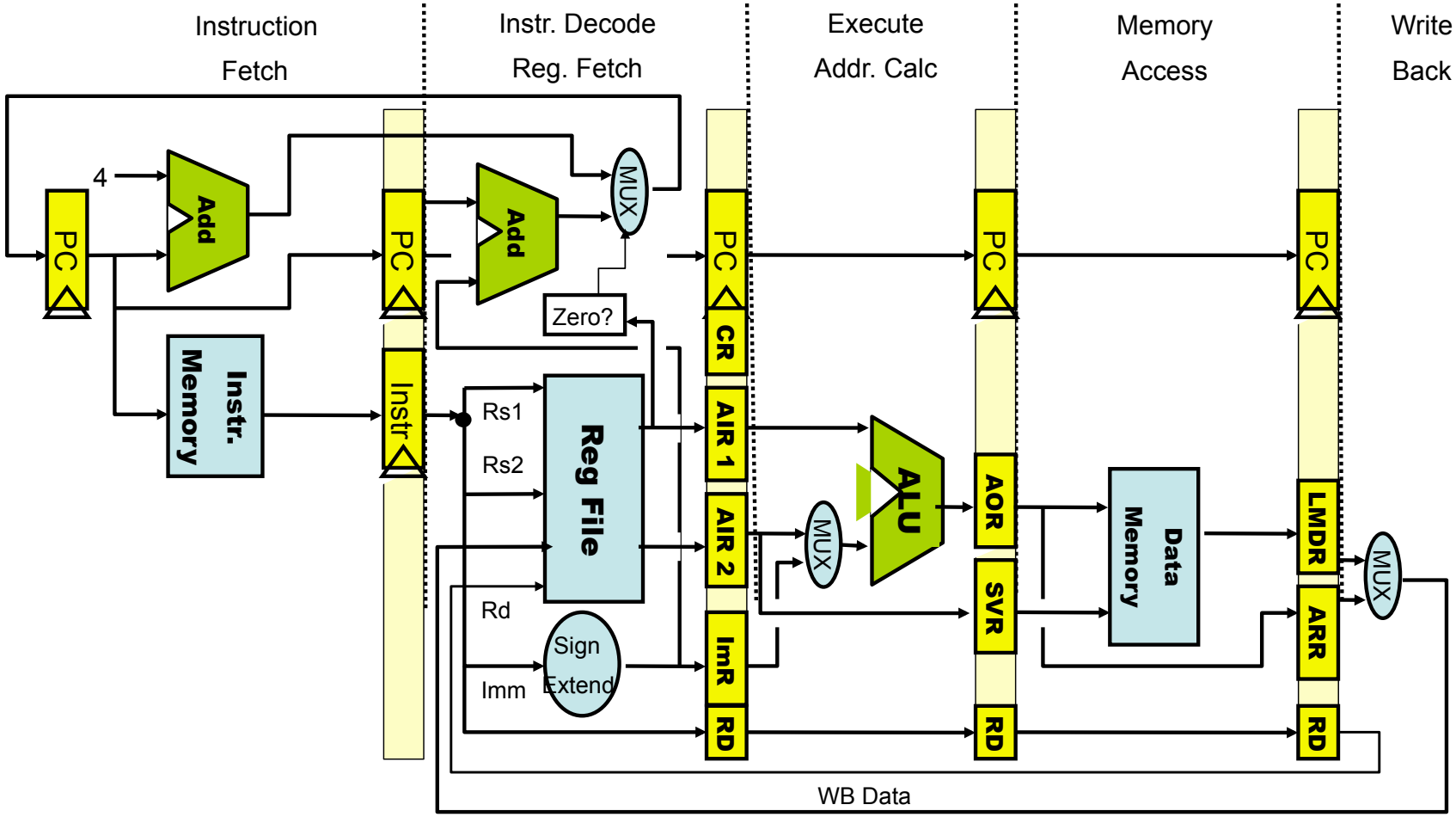
- Si el número de instr de salto es un 30%, detener 2 o 3 ciclos en cada uno tiene un impacto significativo en las prestaciones

- La solución tiene dos partes:
  - Determinar si el salto debe o no efectuarse tan pronto como sea posible , y
  - Calcular la dirección efectiva de salto antes

RIESGOS DE CONTROL: REDUCCIÓN DE LA VENTANA DE RETARDO

- Se produce debido a que la dirección objetivo del salto (branch y jmp) y la condición de salto (branch) se calulan en la fase Ex de la arquitectura DLX, y se reemplaza el PC en la fase MEM.
- Es posible reducir el tamaño de la ventana de retardo.
  - Para ello es necesario calcular la dirección efectiva de salto y evaluar la condición de salto lo antes posible.
  - Esto implica modificar el diseño de la arquitectura del cauce.
- MIPS: Solución utilizada:
  - Mover el test de Zero a la etapa ID/RF
  - Situar el sumador para calcular el nuevo PC en la etapa ID/RF
  - Se consigue 1 ciclo de reloj de penalización en los saltos en vez de 2 o 3

ESQUEMA MODIFICADO



JUMPS AND CALLS (JAL) (SALTOS INCONDICIONALES)

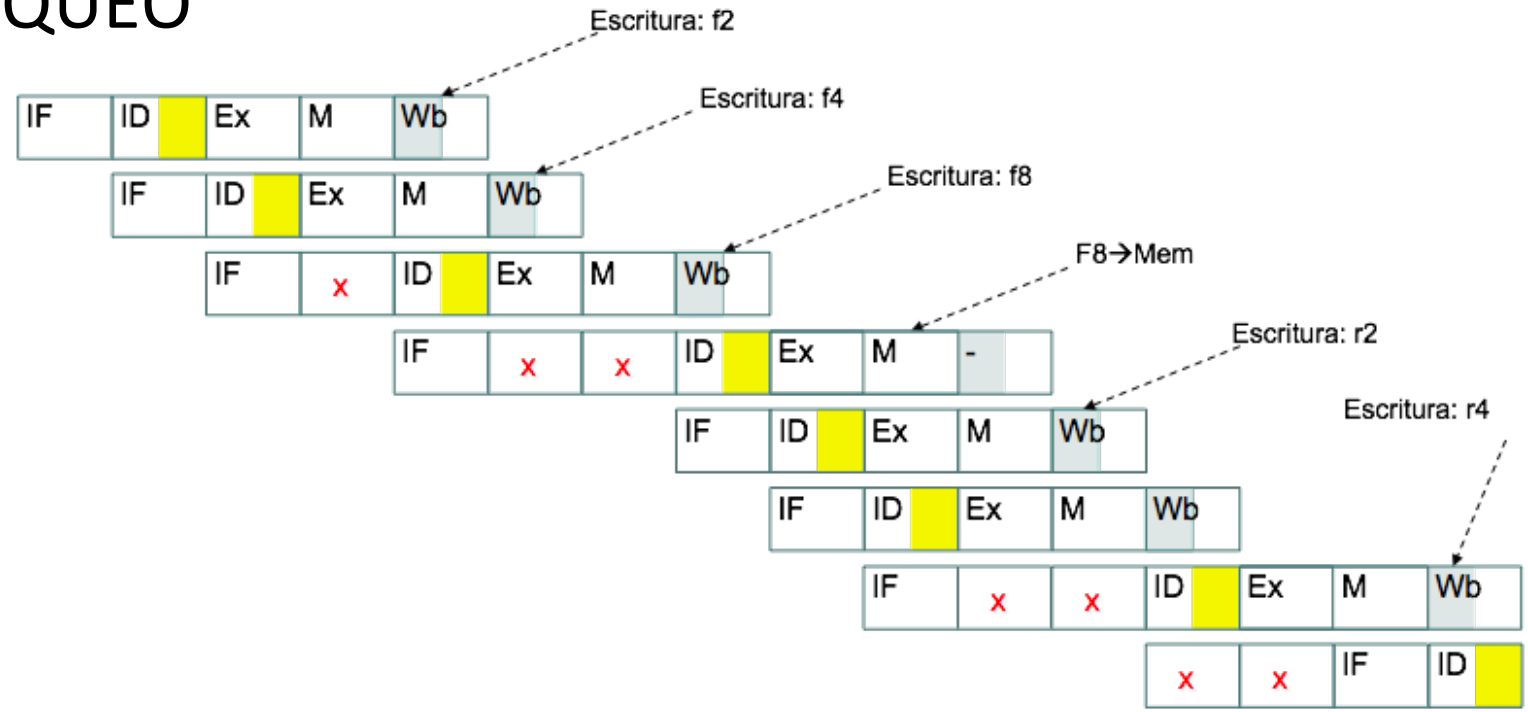
- Aunque sabemos que en estos tipos de instrucciones tenemos que cambiar el PC, aún es necesario un delay slot (ventana de retardo de 1 ciclo) ya que:
  - Descodificación de la instrucción
  - Cálculo del PC
- Básicamente, es una decisión que hay que realizar, lo cual toma su tiempo.
- Esto sugiere un único delay slot:
  - Es decir la siguiente instrucción después de un salto incondicional se ejecuta siempre.

RIESGOS DE CONTROL: SOLUCIONES

- Un rediseño de cauce, como el que se ha comentado antes reduce el retardo a un ciclo, pero puede producir nuevos riesgos.
- Son necesarias técnicas específicas:
- Hardware:
  - una es el interbloqueo.
  - utilizar una cache predescodificada.
  - predicción dinámica de salto (a esta le dedicaremos un tema entero por lo que no lo veremos aquí).
- Software:
  - la técnica de saltos retardados.
  - predicción estática de saltos (dirigida por el compilador).

SOLUCIONES HARDWARE A RIESGOS DE CONTROL: INTERBLOQUEO

```
loop: ld     f2, -8(r1)
      ld     f4, 0(r1)
      multd  f8, f0, f2
      sd     0(r2), f8
      addi   r2, r2, 8
      addi   r1, r1, 8
      slt    r4, r4, r3
      bnez   r4, loop
      yyy1
```

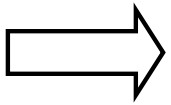


- El Hw interbloquea la siguiente instrucción a ejecutar
  - → hasta que se calcula la dirección efectiva y el sentido del salto.

COMO CONSEGUIR UN JUMP CON "RETARDO CERO"

- Que es lo que realmente hay que hacer en run time?
  - Una vez que se ha detectado que una instr es un jump o JAL, es posible recodificarla en la cache interna.
  - Esto es una forma muy limitada de compilación dinámica

```
and   r3,r1,r5
addi  r2,r3,#4
sub   r4,r2,r1
jal   doit
subi    r1,r1,#1
```

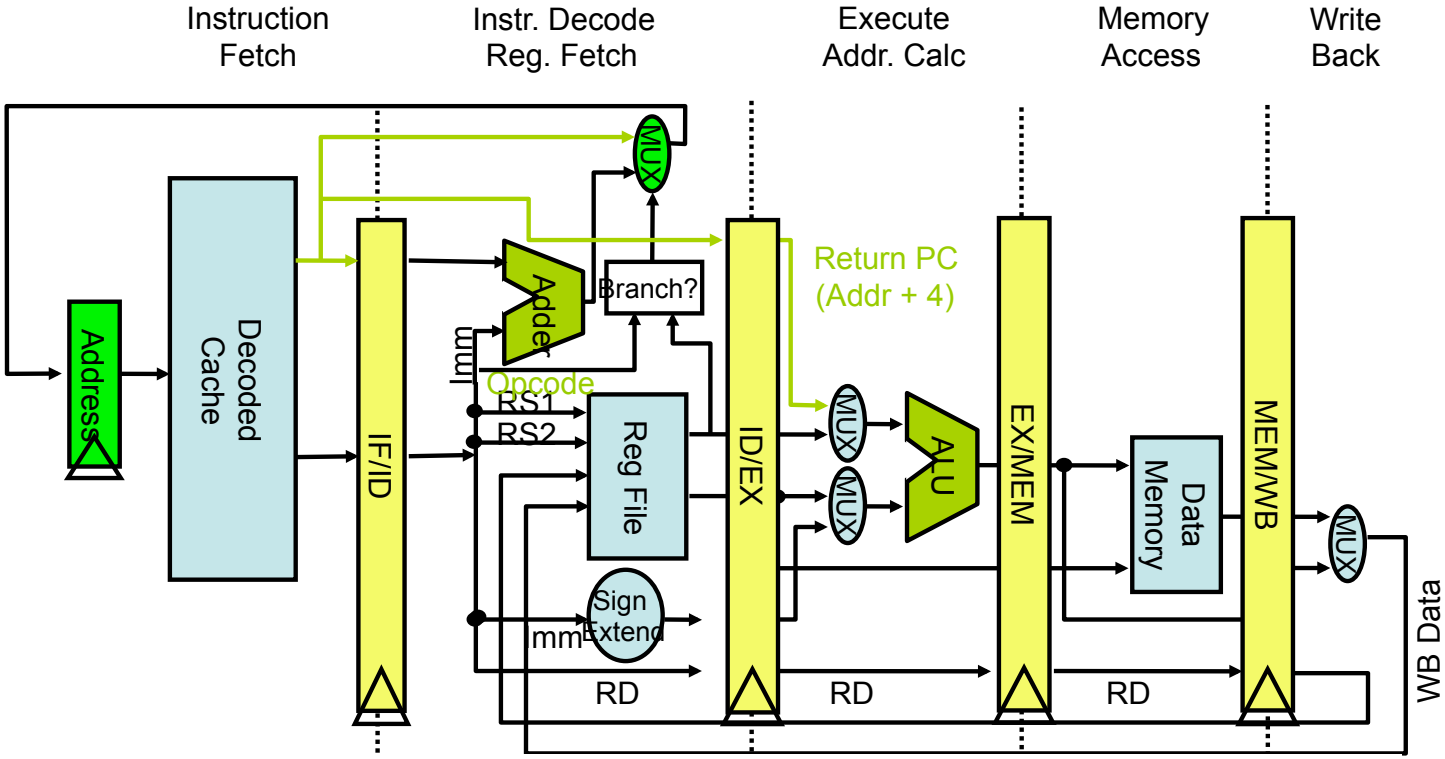


Internal Cache state:

A:	and r3,r1,r5	N	A+4
	addi r2,r3,#4	N	A+8
	sub r4,r2,r1	L	doit
	---	--	---
	subi r1,r1,#1	N	A+20

- Uso de una cache de instr "Pre-descodificadas"
  - Llamado "branch folding" en el CRISP processor de los Bell-Labs.
  - Nota: el JAL introduce un riesgo estructural sobre las escrituras

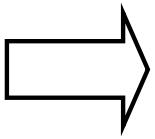
SOLUCIONES SOFTWARE RIESGOS DE CONTROL: ESQUEMA MODIFICADO (CACHE PREDESCODIFICADA)



- Incrementa el ciclo de reloj en no menos de un retardo por el MUX
- Si embargo introduce un riesgo estructural sobre las escrituras debido al JAL

POR QUÉ NO USARLO CON LOS SALTOS CONDICIONALES?

```
and   r3,r1,r5
addi  r2,r3,#4
sub   r4,r2,r1
bne   r4,loop
subi   r1,r1,#1
```

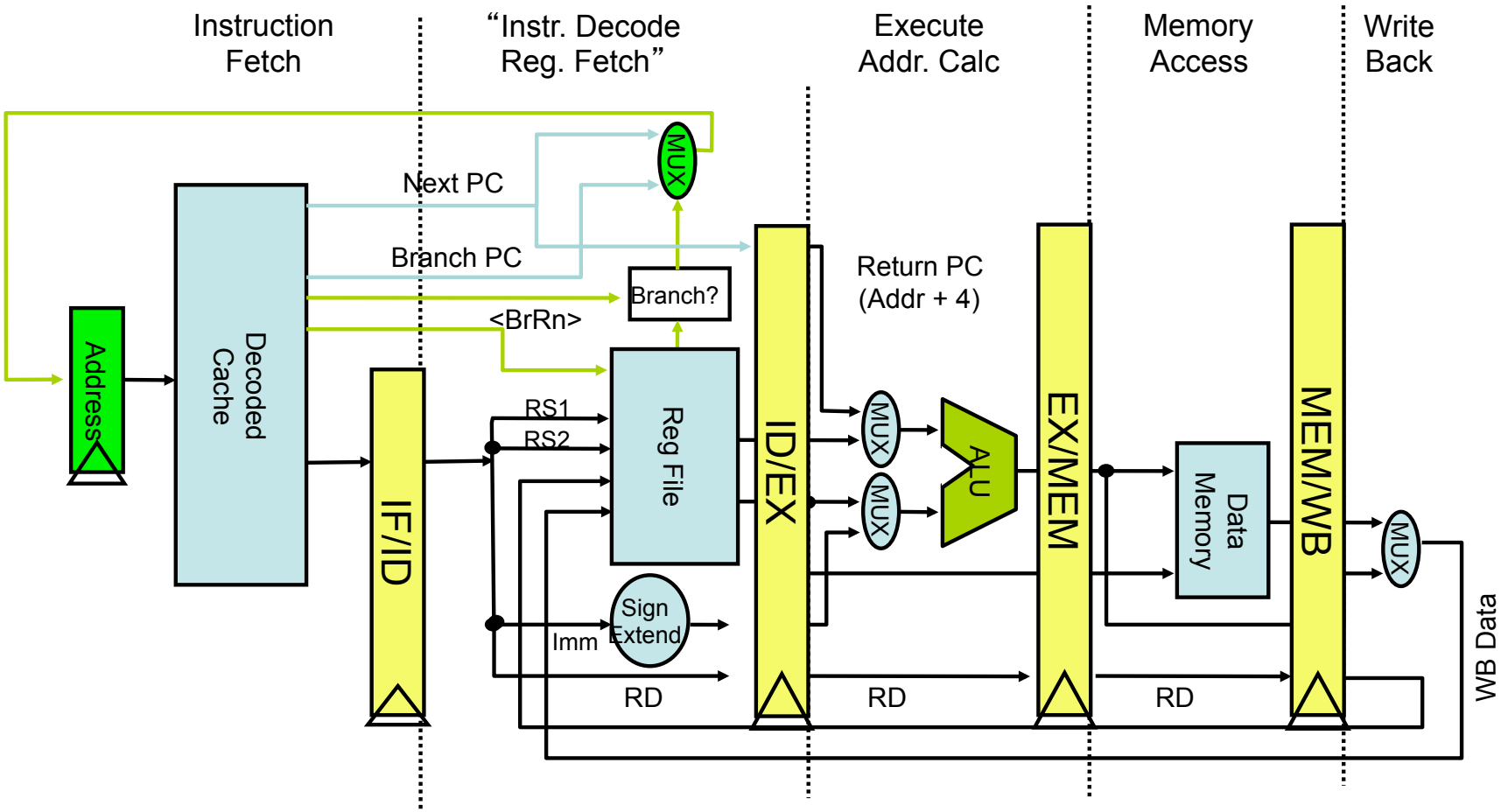


Internal Cache state:

			Next	Branch
A:	and r3,r1,r5	N	A+4	N/A
	addi r2,r3,#4	N	A+8	N/A
	sub r4,r2,r1	BnR4	A+16	loop
	---	--	---	---
A+16:	subi r1,r1,#1	N	A+20	N/A

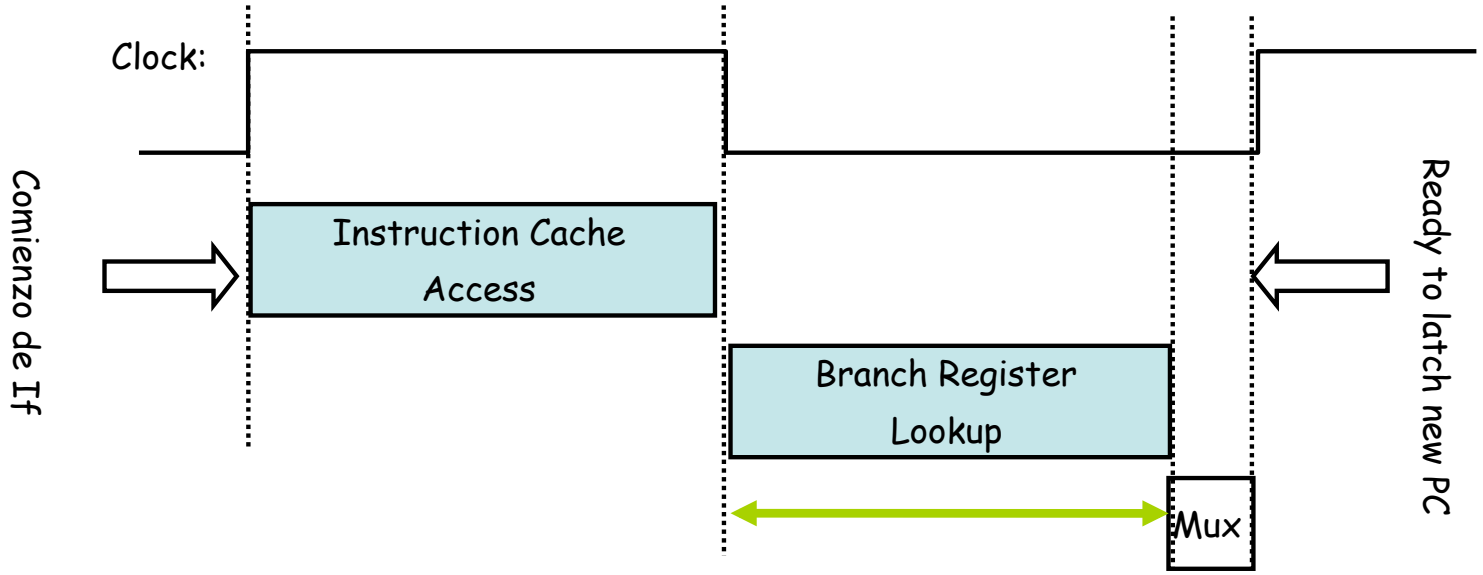
- Se elimina el delay slot (bueno)
- El branch ha sido solapado con la instrucción sub (bueno).
- Incrementa el tamaño de la cache de instrucciones (no tan bueno)
- Requiere otro puerto de lectura en el banco de registros (MALO)
- Dobla potencialmente el periodo de reloj (REALMENTE MALO)

ESQUEMA MODIFICADO



- Puede doblar el periodo de reloj – debe acceder a la cache y a los reg

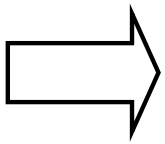
TEMPORIZACIÓN



El tiempo de acceso al banco de registros debe ser cercano al periodo de reloj original

SIN EMBARGO, PODEMOS USAR LA PRIMERA TÉCNICA PARA REFLEJAR PREDICCIONES Y ELIMINAR DELAY SLOTS

```
and  r3,r1,r5
addi      r2,r3,#4
sub  r4,r2,r1
bne  r4,loop
subi  r1,r1,#1
```



Internal Cache state:

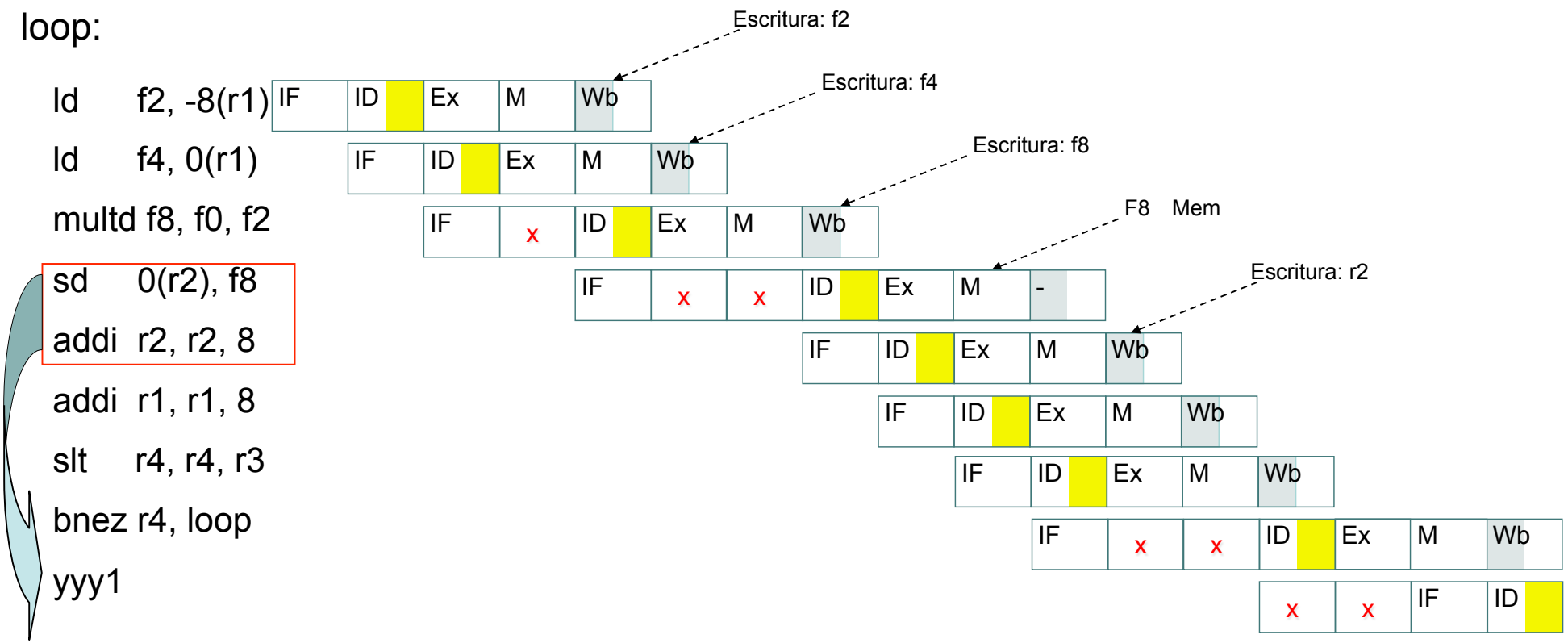
			Next
A:	and r3,r1,r5	N	A+4
	addi r2,r3,#4	N	A+8
	sub r4,r2,r1	N	A+12
	bne r4	N	loop
A+16:	subi r1,r1,#1	N	A+20

- Esto causa que la siguiente instrucción se busque inmediatamente en los destinos de los saltos (predict taken)
- Si el salto finalmente no es realizado, entonces se elimina la instrucción de destino y se continua por la dirección A+16

SOLUCIONES SOFTWARE RIESGOS DE CONTROL: TÉCNICA DE SALTOS RETARDADOS

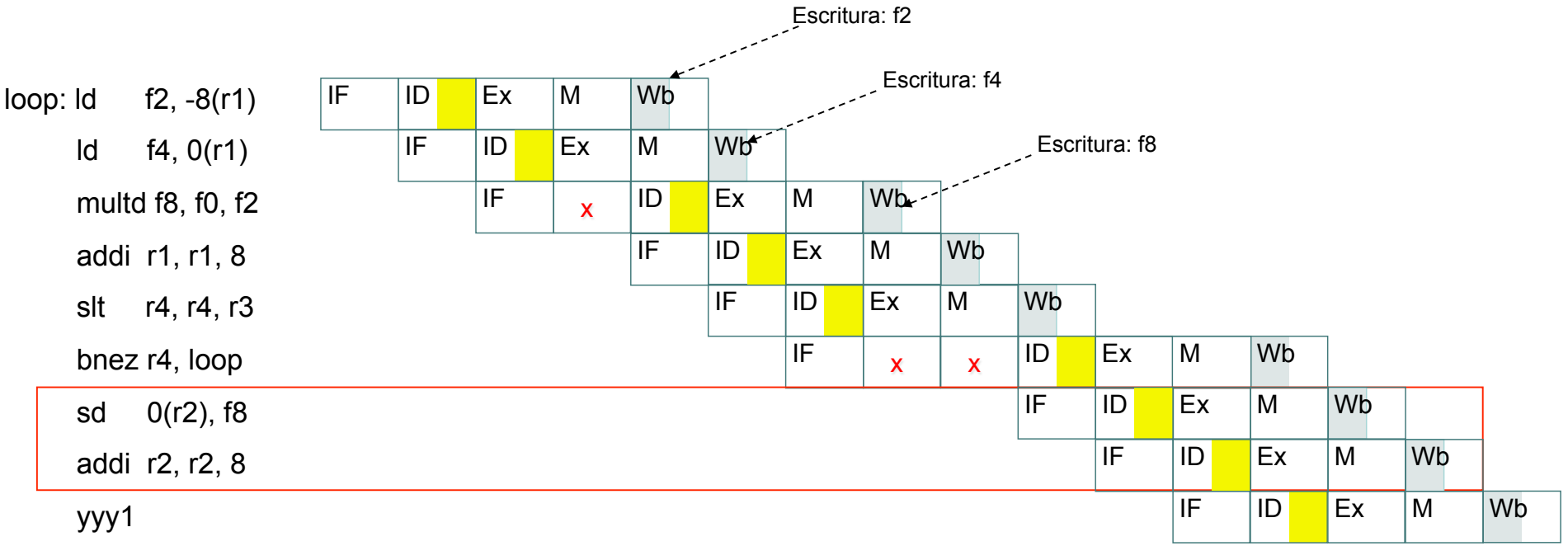
- Idea básica:
  - El compilador rellena el delay slot con instrucciones que no tengan dependencias con la de salto, y que estén situadas antes del jump o branch (según orden de programa).
  - Si no hay suficientes instrucciones para mover al delay slot este se rellena con instrucciones no-op.
  - Estas instrucciones se ejecutarán siempre.
- Se usó por vez primera en las primeras generaciones de proc RISC escalares: IBM 801, Berkeley RISC I y el MIPS de Stanford.
- Observación:
  - La probabilidad de encontrar una instrucción que mover a la ventana de retardo es 0.6.
  - La probabilidad de encontrar 2 instrucción es 0.2.
  - La probabilidad de encontrar 3 instrucción es 0.1 .

SALTO RETARDADO: EJEMPLO



- Bucle sin usar salto retardado
  - Existen instrucciones independientes que no afectan a la de salto.

SALTO RETARDADO: EJEMPLO



- Bucle usando la técnica de salto retardado
  - La ventana de retardo se ha completado con instrucciones independientes que no afectan a la de de salto

SOLUCIONES SOFTWARE RIESGOS DE CONTROL: PREDICCIÓN ESTÁTICA DEL SALTO

- Todos los saltos como tomados.
- Los saltos “hacia atrás” predecirlos como tomados y los saltos “hacia adelante” predecirlos como no tomados.
- Hacer una predicción sobre la dirección del salto:
  - La predicción es realizada por el compilador y se incluye en el formato de la instrucción branch.
  - Las instrucciones se comienzan a buscar en la dirección objetiva del salto predicha por este bit.
  - Si la predicción es errónea, las instrucciones que se han comenzado a ejecutar según ese path se eliminan del cauce y se busca en la dirección correcta.
- La predicción es estática por lo que no puede ser alterada, salvo en la compilación.

IDEAS PARA REDUCIR LAS DETENCIONES

Technique	Reduces
Dynamic scheduling	Data hazard stalls
Dynamic branch prediction	Control stalls
Issuing multiple instructions per cycle	Ideal CPI
Speculation	Data and control stalls
Dynamic memory disambiguation	Data hazard stalls involving memory
Loop unrolling	Control hazard stalls
Basic compiler pipeline scheduling	Data hazard stalls
Compiler dependence analysis	Ideal CPI and data hazard stalls
Software pipelining and trace scheduling	Ideal CPI and data hazard stalls
Compiler speculation	Ideal CPI, data and control stalls

REFERENCIAS

[Patterson, 1997]David A. Patterson, John L. Hennessy (1997) Computer Organization & Design: The Hardware/Software Interface, Second Edition. Morgan Kaufmann.