Segmentación

Segmentación

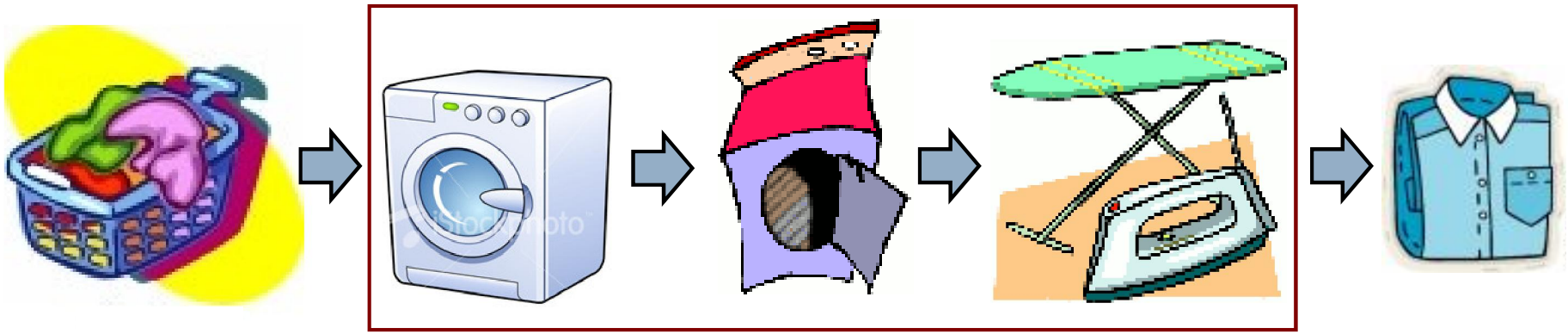
1.- Introducción
2.- Ruta de datos segmentada
3.- Control de la ruta de datos segmentada
4.- Riesgos
    4.1.- Riesgos de datos
        Solución software a los riesgos LDE
        Anticipación de operandos
        Detención del pipeline
        Reordenamiento de código
    4.2.- Riesgos de control
        Soluciones a los riesgos de control
        Predicción dinámica de saltos

Bibliografía:

"Estructura y diseño de computadores" David A. Patterson & John L. Hennessy, Editorial Reverté, 2000

"Computer architecture. A quantitative approach", J.L. Hennessy & D.A. Patterson, Morgan Kaufmann, 2ª edic.

1.- Introducción



Lavadora
1 hora

Secadora
45 minutos

Planchado
1 hora

Total 2 horas 45 minutos

1.-¿Cómo se puede hacer más rápido?

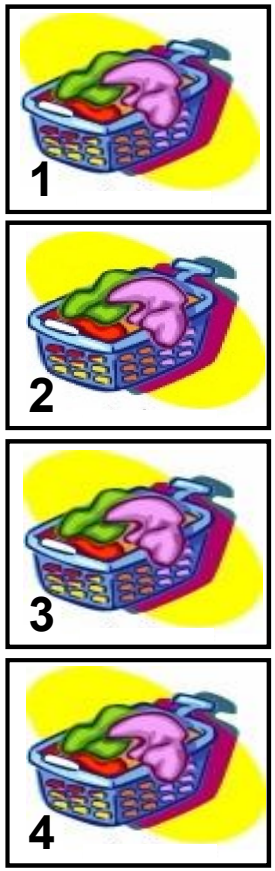
Lavadora
1 hora

Secadora
45 minutos

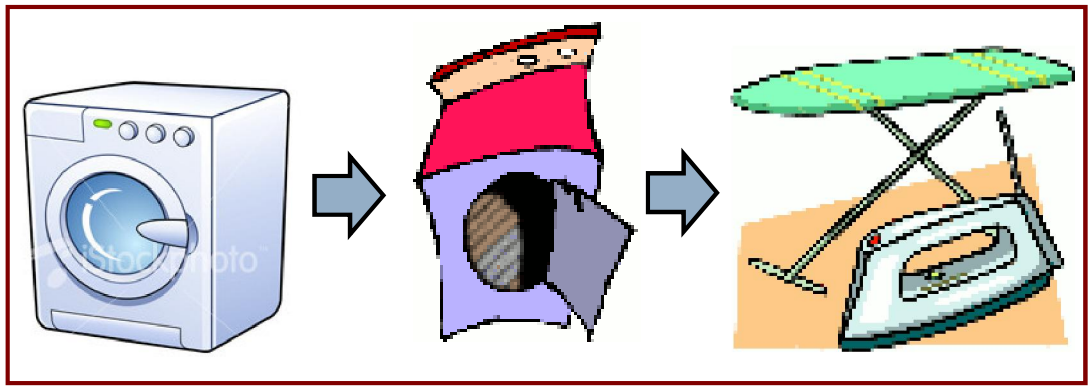
Planchado
1 hora

Total 11 horas

1.-¿Cómo se puede hacer más rápido?



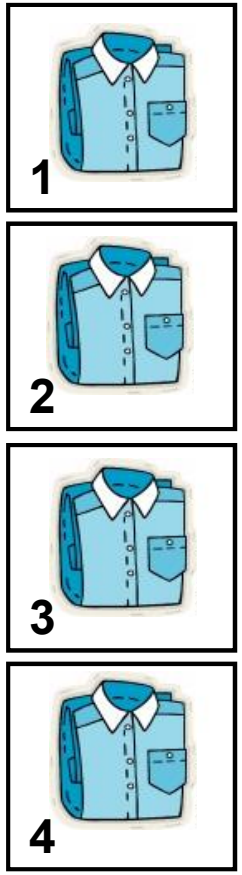
Solución 1: ejecución segmentada



Lavadora
1 hora

Secadora
45 minutos

Planchado
1 hora



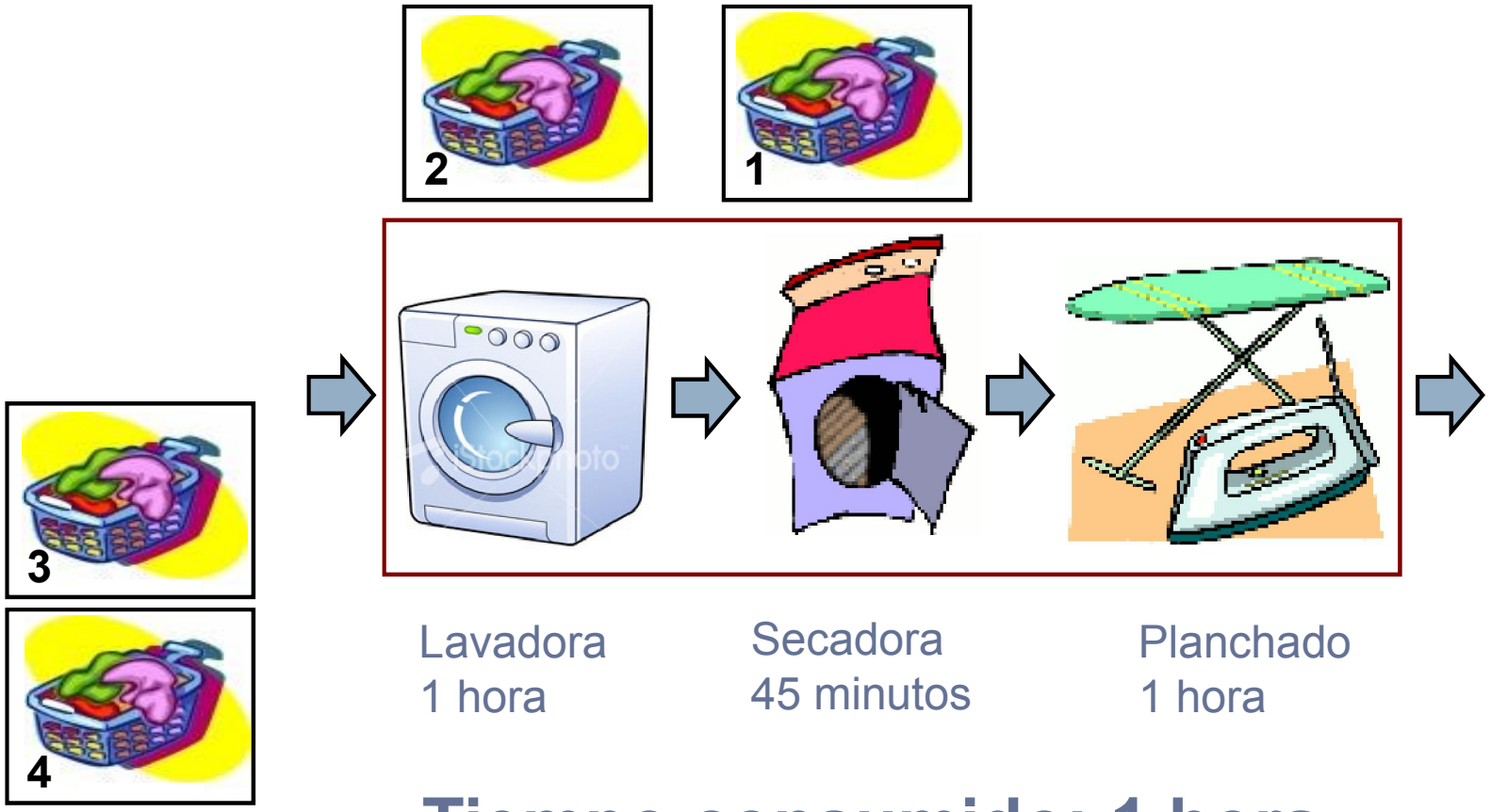
Tenemos tres recursos pero en cada instante sólo usamos uno

¿y si tratamos de usar los tres a la vez?

1.-¿Cómo se puede hacer más rápido?

Tiempo consumido: 0 horas

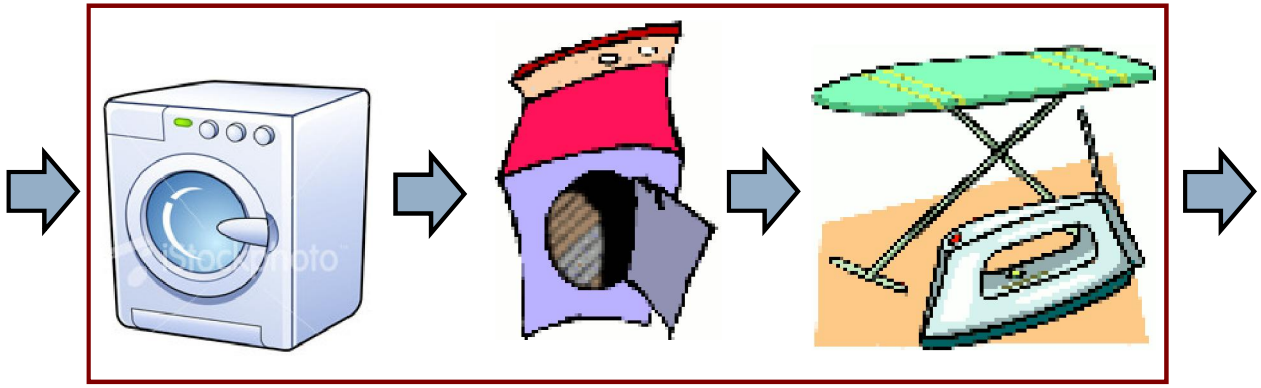
1.-¿Cómo se puede hacer más rápido?



Tiempo consumido: 1 hora

Usamos dos recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Lavadora
1 hora

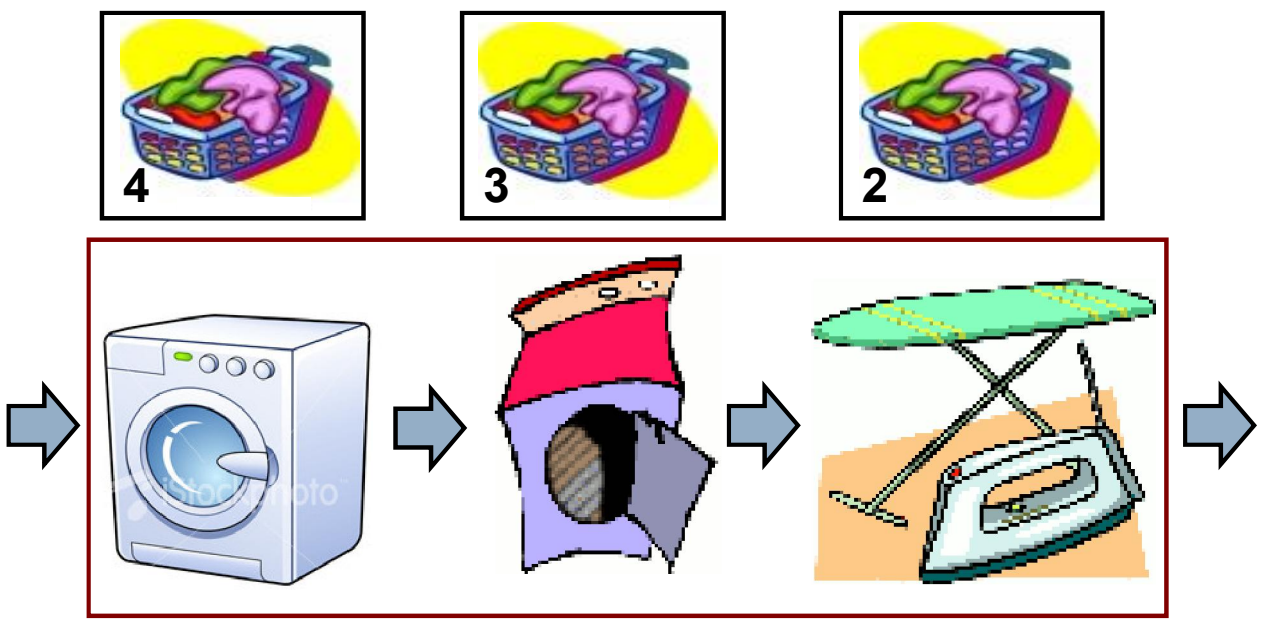
Secadora
45 minutos

Planchado
1 hora

Tiempo consumido: 2 horas

Usamos tres recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Lavadora
1 hora

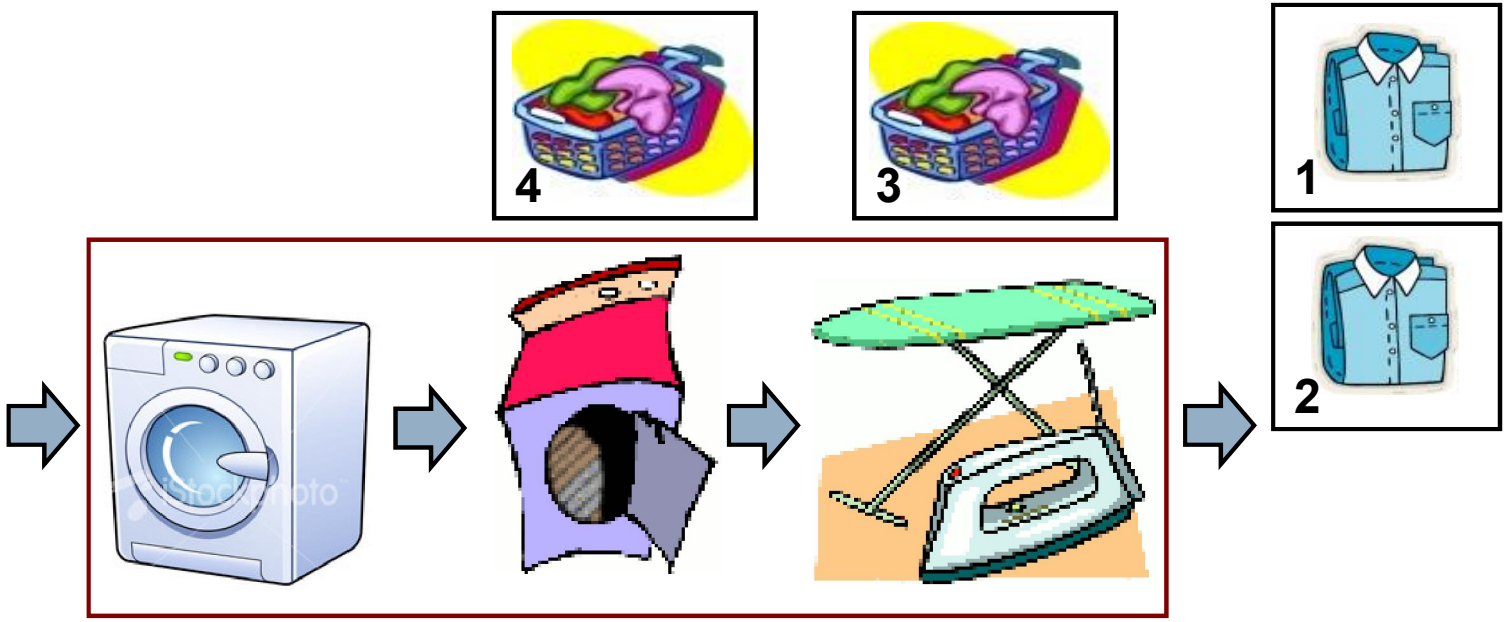
Secadora
45 minutos

Planchado
1 hora

Tiempo consumido: 3 horas

Usamos tres recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Lavadora
1 hora

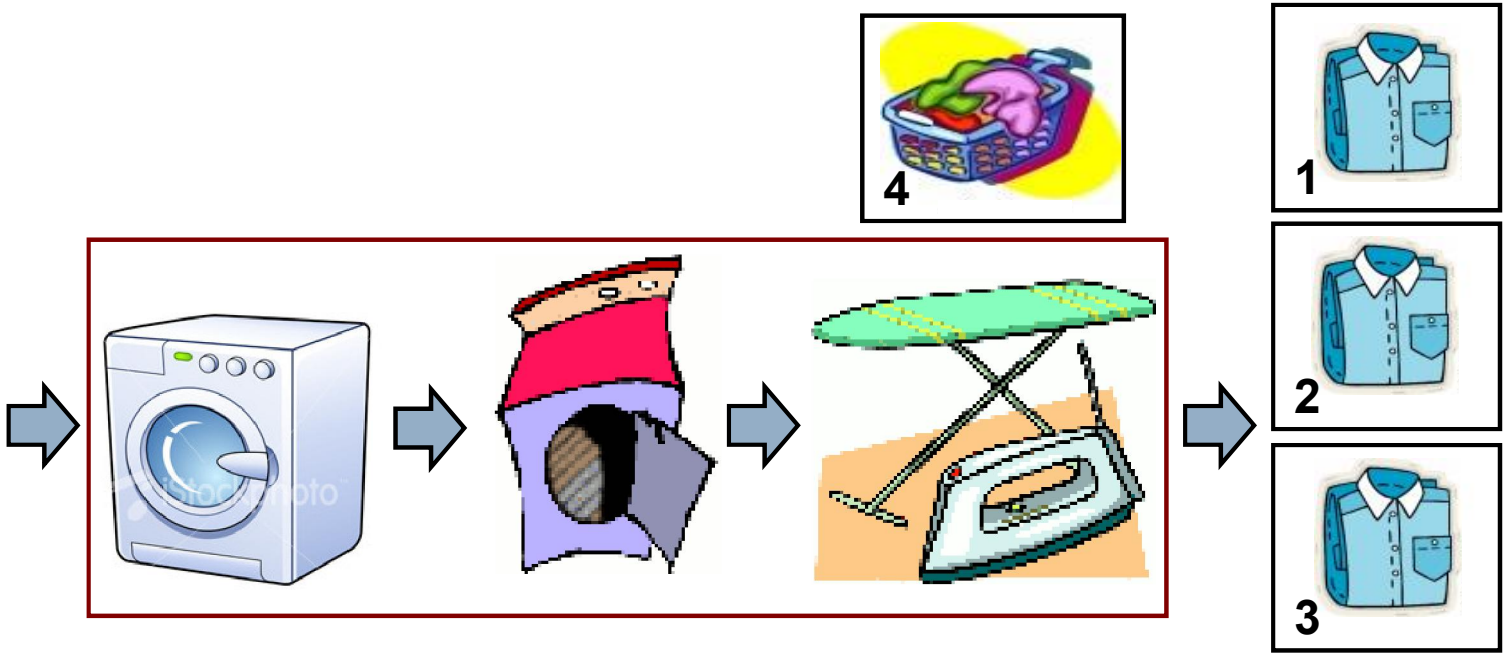
Secadora
45 minutos

Planchado
1 hora

Tiempo consumido: 4 horas

Usamos dos recursos en paralelo

1.-¿Cómo se puede hacer más rápido?



Lavadora
1 hora

Secadora
45 minutos

Planchado
1 hora

Tiempo consumido: 5 horas

1.-¿Cómo se puede hacer más rápido?

Lavadora
1 hora

Secadora
45 minutos

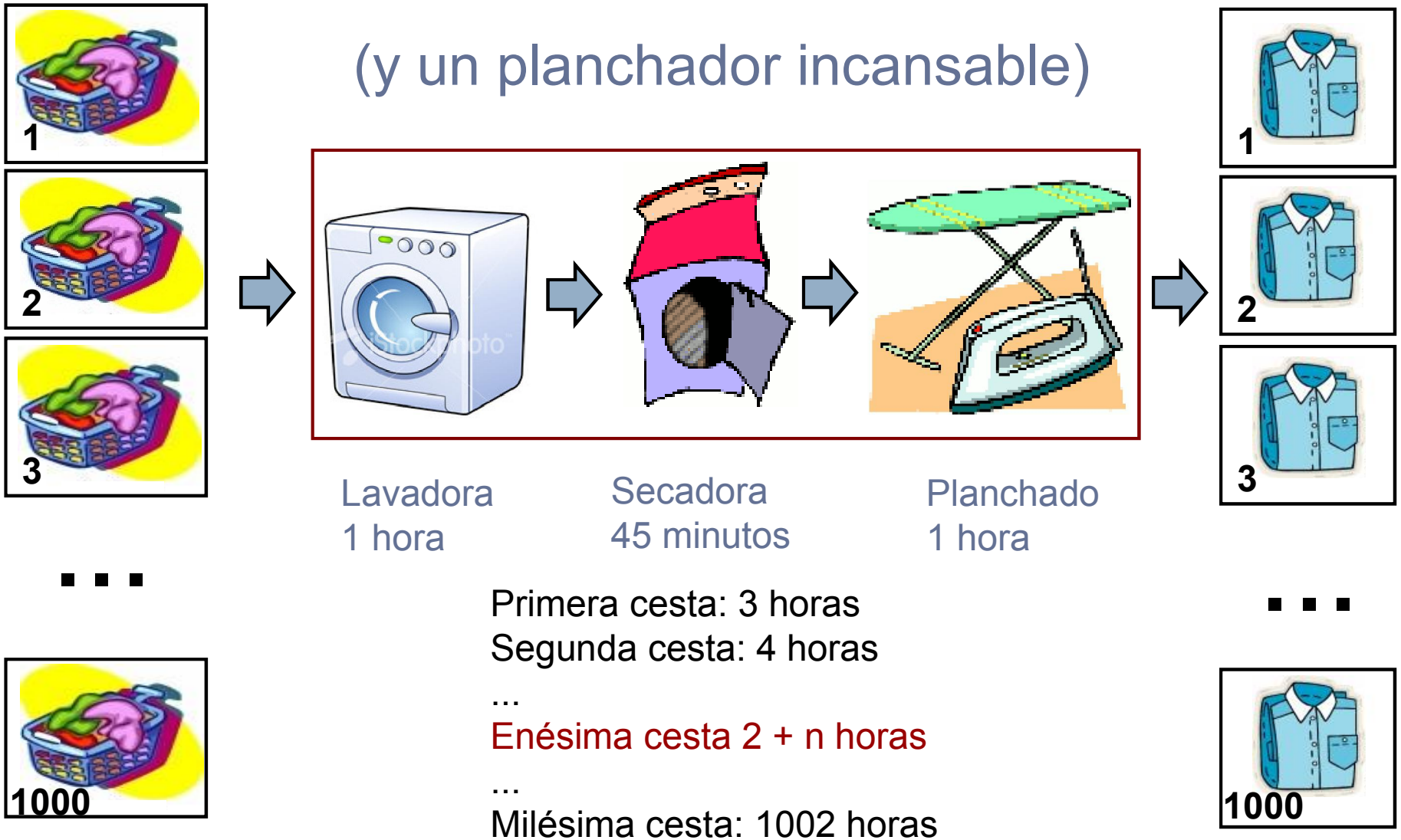
Planchado
1 hora

Tiempo consumido: 6 horas

¡Antes eran 11 horas!

1.-¿Y si tuviésemos 1000 cestas de ropa?

(y un planchador incansable)



Lavadora
1 hora

Secadora
45 minutos

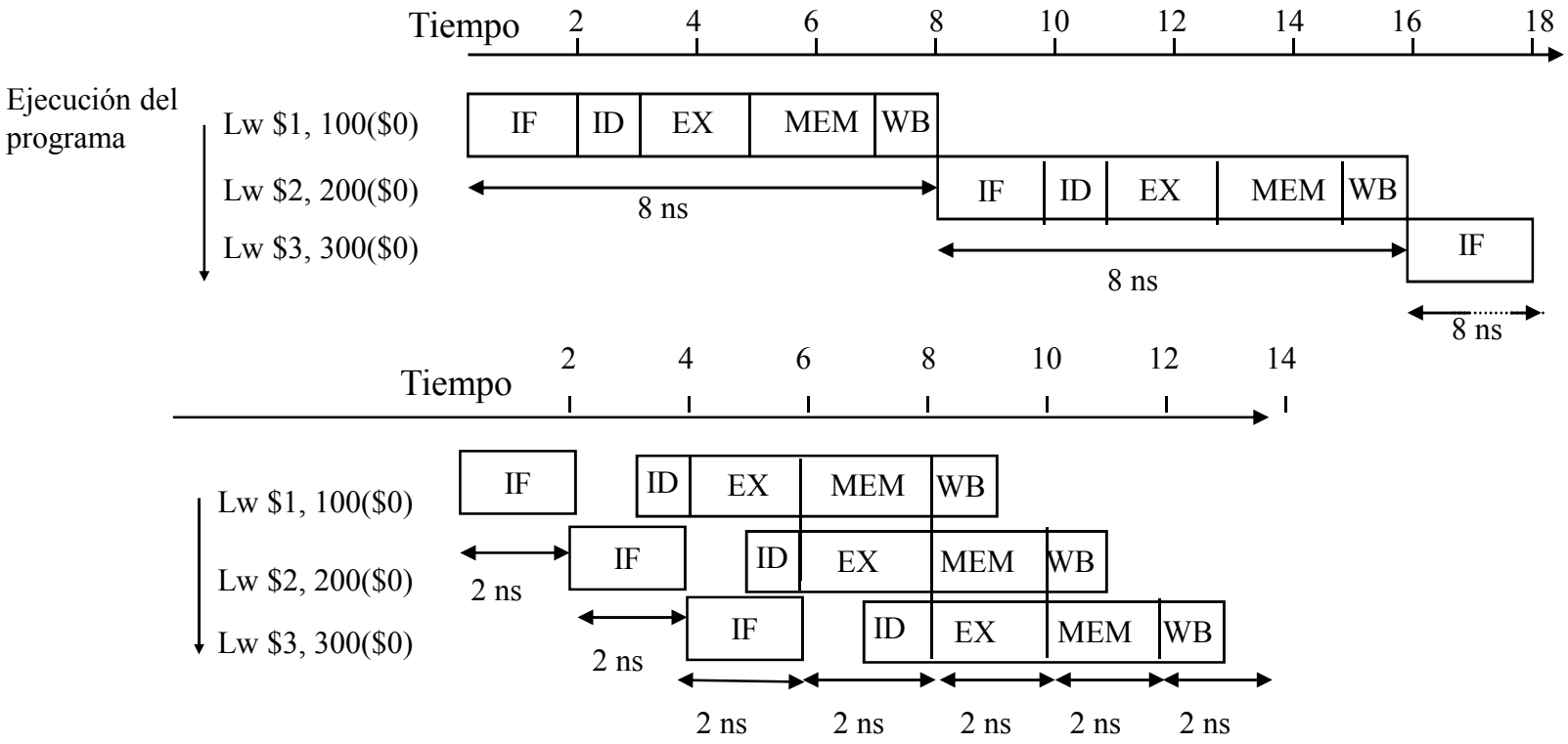
Planchado
1 hora

Primera cesta: 3 horas
Segunda cesta: 4 horas
...
Enésima cesta 2 + n horas
...
Milésima cesta: 1002 horas

¿Cómo se puede aplicar a un procesador?

- Dividimos la ejecución de la instrucción en cinco etapas fundamentales:
  - IF: Se lee la instrucción de la memoria de instrucciones
  - ID: se decodifica la instrucción y se leen los operandos del banco de registros. Se extiende el signo del operando inmediato
  - EX: Se utiliza la ALU para realizar los cálculos
  - MEM: Se lee o escribe en la memoria de datos
  - WB: Se escribe en banco de registros
- Cada etapa puede ejecutar una instrucción distinta
- Podemos ejecutar hasta cinco instrucciones en paralelo

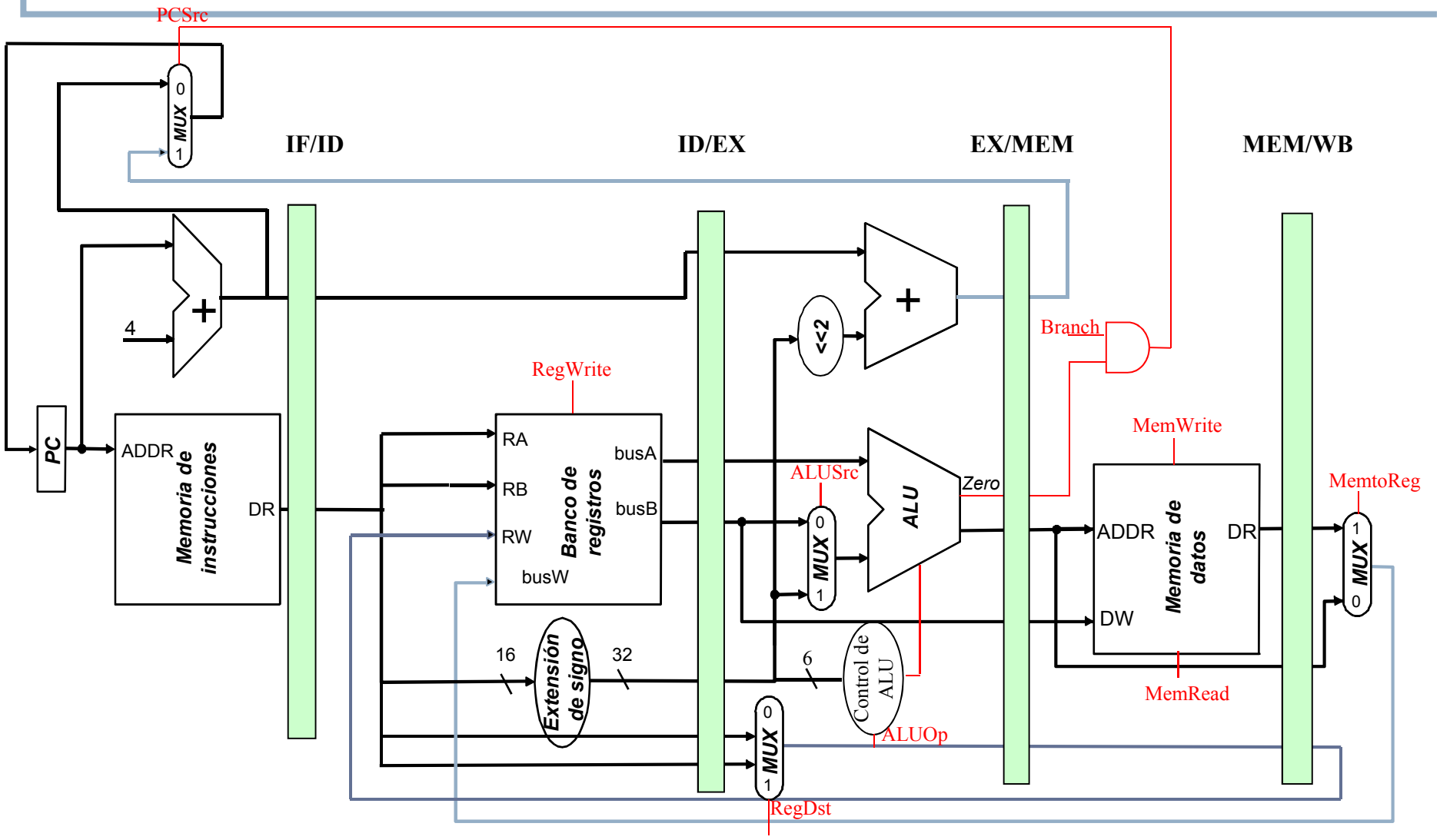
1.- ¿Cómo se puede aplicar a un procesador?



El speedup ideal = número de etapas

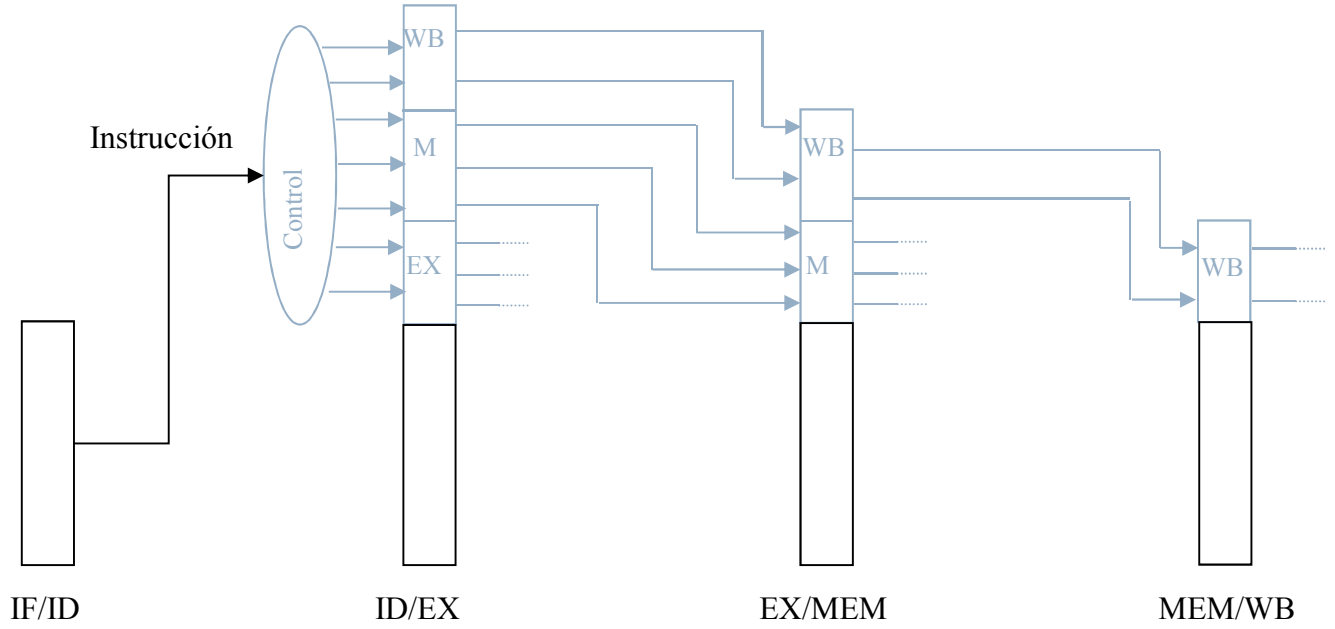
¿Es posible?

3.- Ruta de datos segmentada

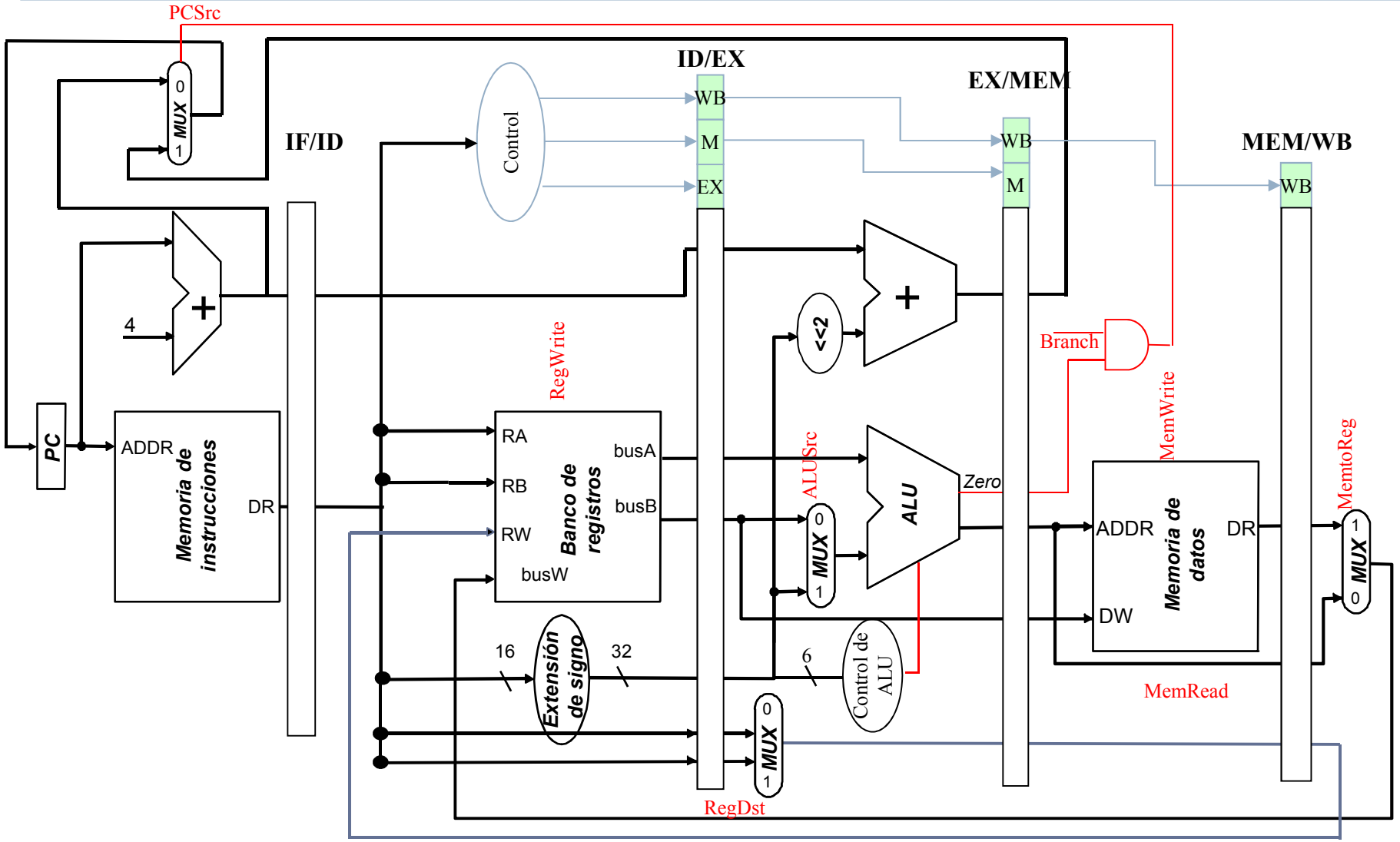


3.- Control de la ruta de datos segmentada

Las señales de control se generan en la U.C. y se van pasando de una etapa a otra como si fuesen datos.



3.- Control de la ruta de datos segmentada



4.- Riesgos

- Estructurales
  - No disponibilidad de un módulo cuando se necesita
  - No existen en nuestro MIPS porque tenemos:
    - Memoria de datos e instrucciones separadas
    - Arquitectura registro-registro
    - Sumadores adicionales para el cálculo del PC
  - En general se solucionan duplicando módulos y segmentando las ALUs
- De datos
  - No disponibilidad de un dato cuando se necesita
- De control
  - Desconocimiento de cuál es la próxima instrucción que debe ejecutarse

Dependencias vs. Riesgos

- Dependencia
  - Propiedad del código

```
subcc r1, r2, r3
addcc r5, r1, r4
```

- Riesgo:
  - Limitación hardware

orden de ejecución
≠ orden de programa

- Si hay dependencia …

… puede existir o no riesgo

	Dependencia	Riesgo
Dependencias de almacenamiento / nombre *(pueden eliminarse)*	Dependencia s.s. *True dependency*	RAW *Read After Write*
	Antidependencia *Antidependency*	WAR *Write After Read*
	Dep. de salida *Output Dependency*	WAW *Write After Write*

4.1.- Dependencias y riesgos

- **1.- Antidependencia**

  sub $2, $1, $3

  add $1, $4, $5

  - Posible riesgo: *escritura después de lectura*
  - No aparece en MIPS

- **2.- Dependencia de salida**

  sub $2, $1, $3

  add $2, $4, $5

  - Posible riesgo: *escritura después de escritura*
  - No causan riesgo en MIPS

- **3.- Dependencia** *(productor – consumidor)*

  sub $2, $1, $3 *; productor*

  add $4, $2, $5 *; consumidor a distancia 1*

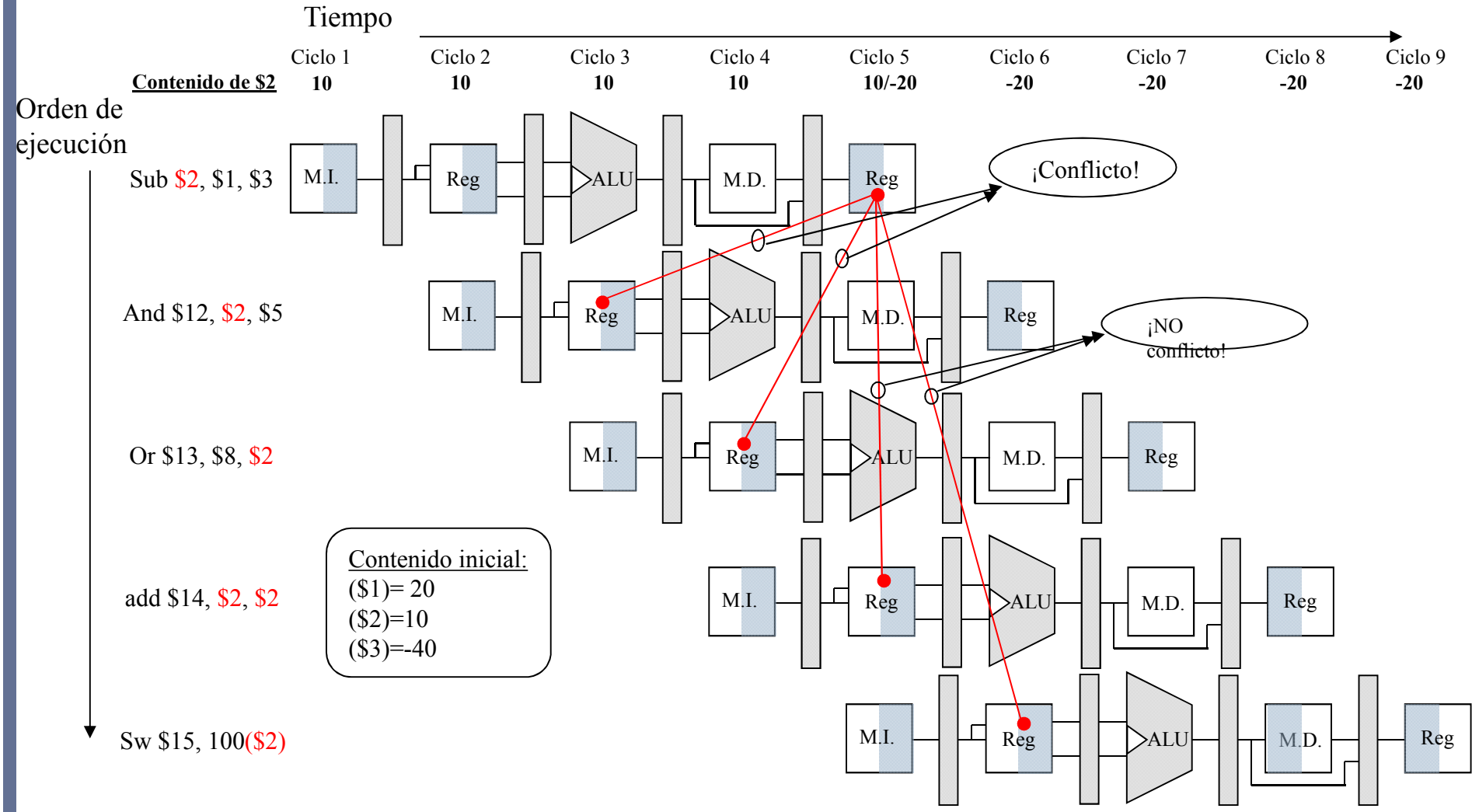
  or $6, $7, $2 *; consumidor a distancia 2*

  - Causan *riesgo de lectura después de escritura (LDE)* en MIPS

4.1.- Riesgos de datos LDE

La ejecución de una instrucción comienza antes de finalizar las anteriores

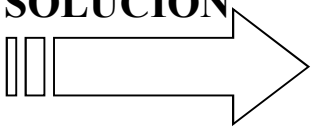


Solución software a los riesgos LDE

- El compilador se encarga de garantizar que no se produzcan riesgos
- Se *insertan instrucciones* NOP entre aquellas instrucciones que tengan dependencias de datos
- La ejecución se hace **más lenta**
    - **Tex = I x CPI x Tc ¿qué término aumenta?**

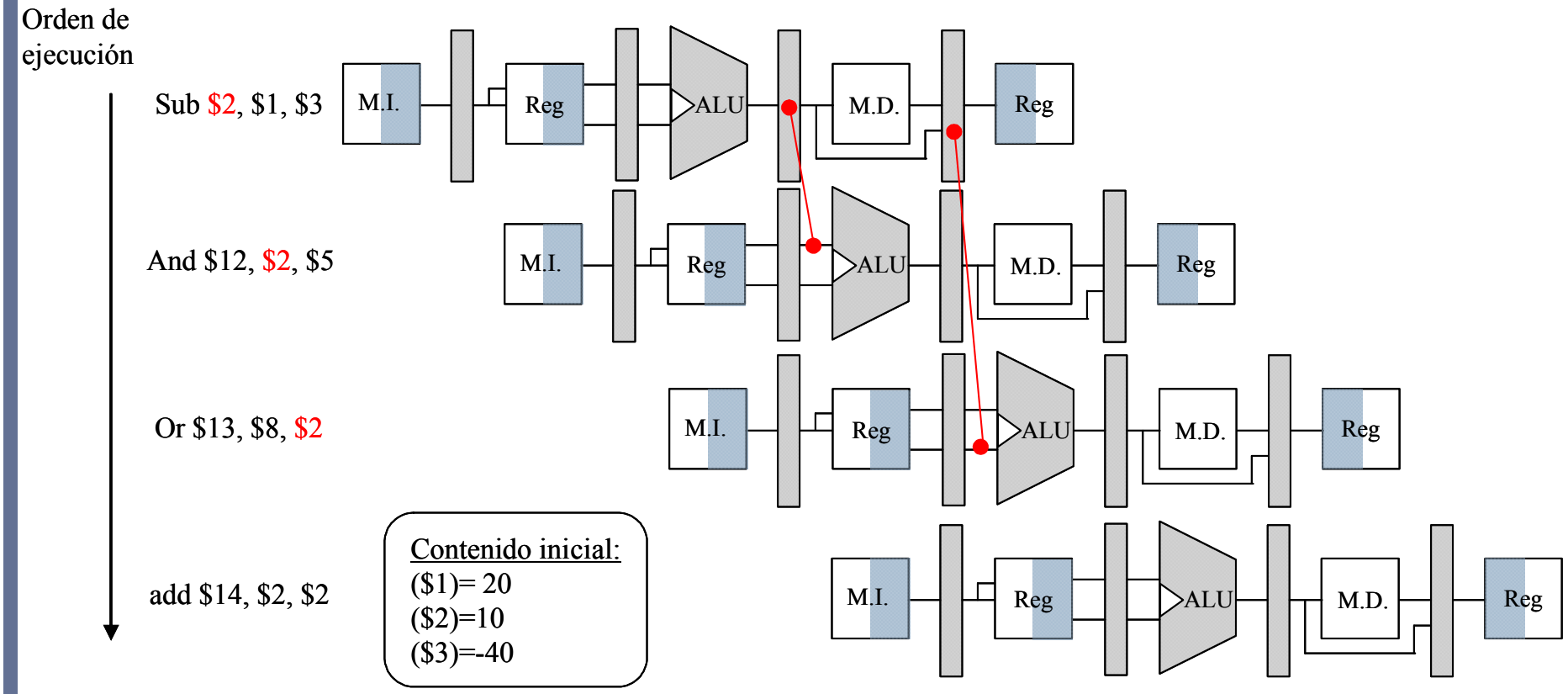
```
Sub     $2, $1, $3
and     $12, $2, $5
or      $13, $6, $2
add     $14, $2, $2
sw      $15, 100($2)
```

SOLUCIÓN



4.1.- Anticipación de operandos (forwarding)

- Solución hardware a riesgos LDE
- Usa resultados temporales sin esperar a que se escriban en el banco de registros



4.1.- Anticipación de operandos

¿Cómo detectar si hay que activar un cortocircuito?

F D **E** M W

etapa destino

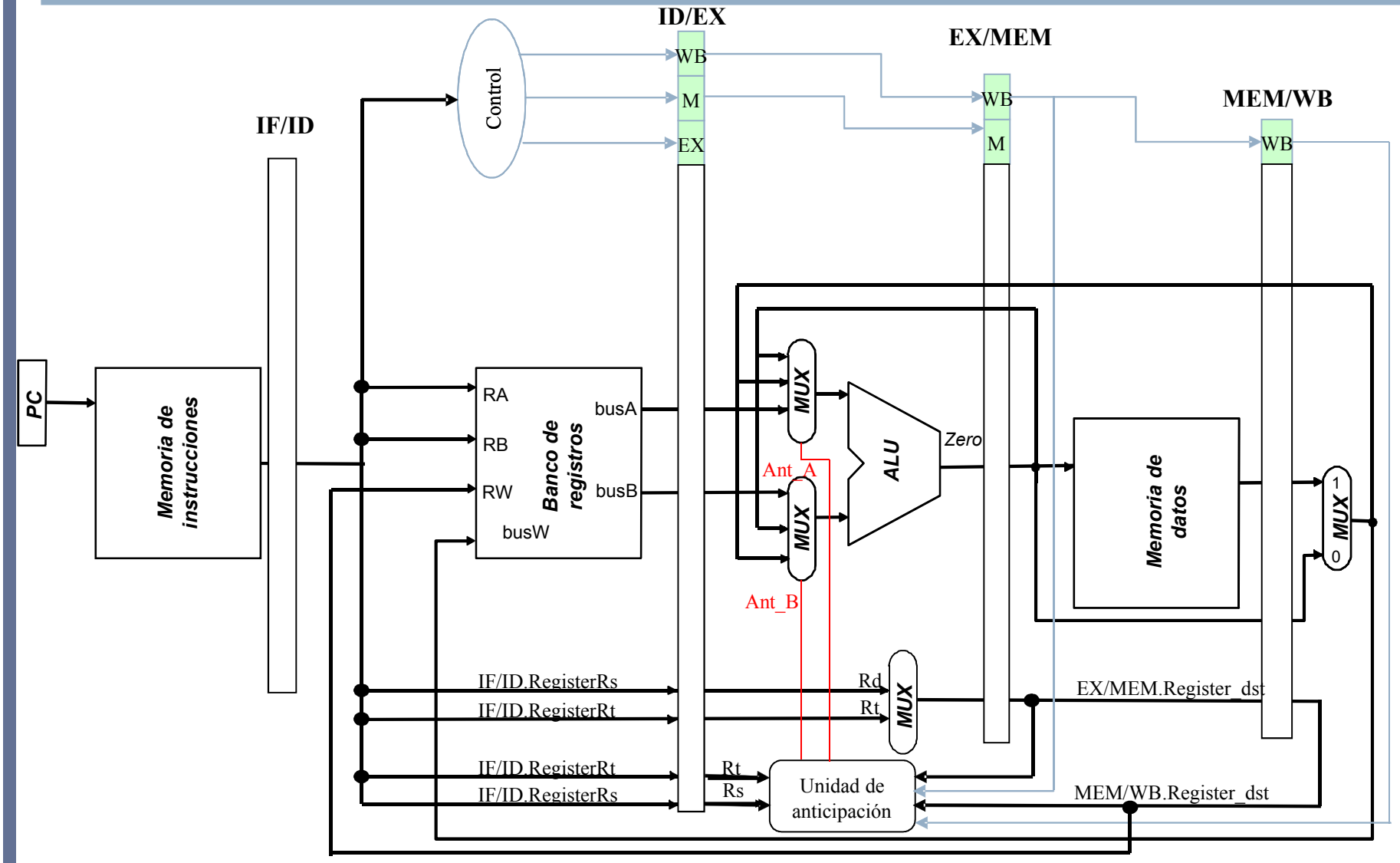
Si (**EX/MEM**.RegWrite
and **EX/MEM**.Register_**dst** = **ID/EX**.RegisterRs)) **Ant_A** = 10

Si (**EX/MEM**.RegWrite
and **EX/MEM**.Register_**dst** = **ID/EX**.RegisterRt)) **Ant_B** = 10

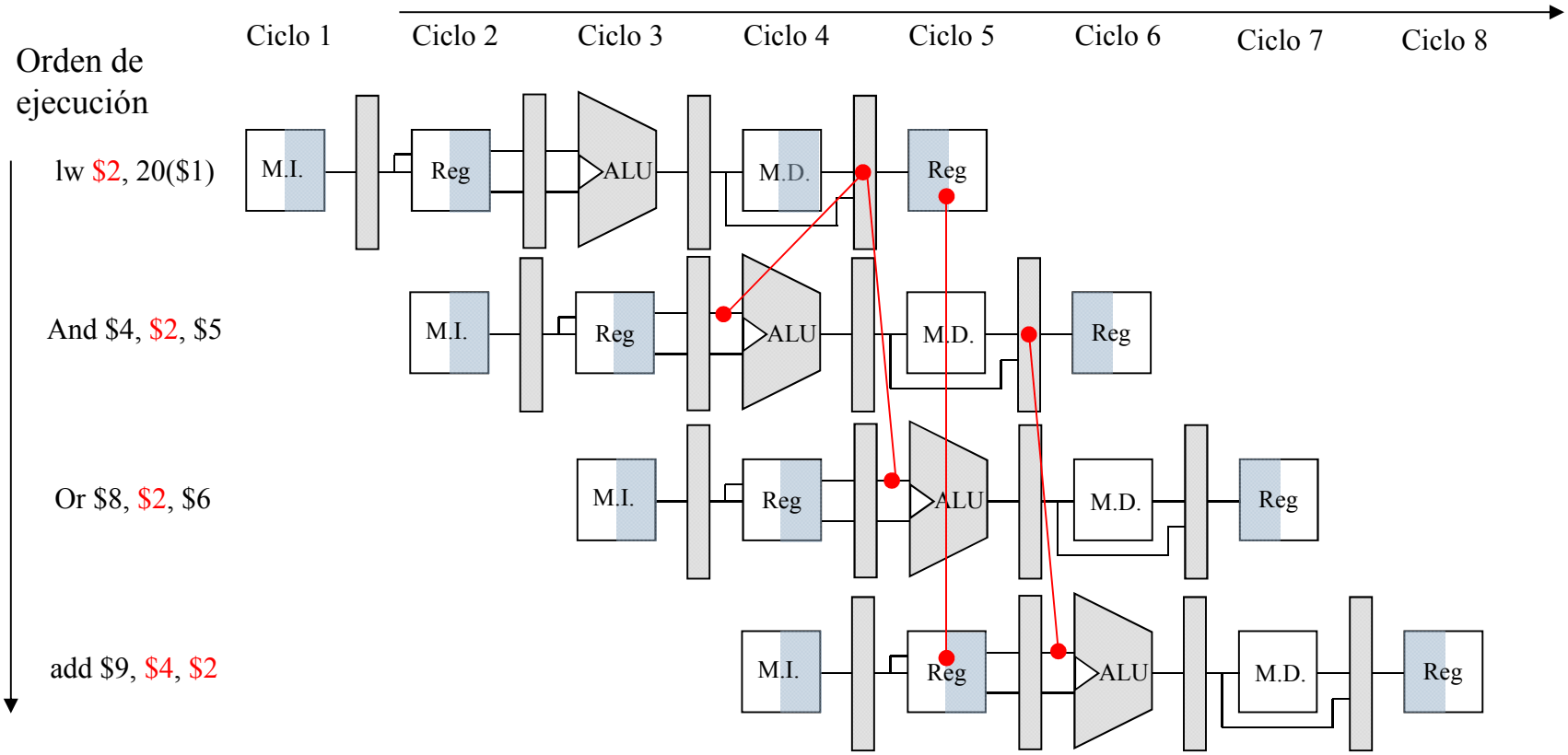
Si (**MEM/WB**.RegWrite
and **MEM/WB**.Register_**dst** = **ID/EX**.RegisterRs)) **Ant_A** = 01

Si (**MEM/WB**.RegWrite
and **MEM/WB**.Register_**dst** = **ID/EX**.RegisterRt)) **Ant_B** = 01

4.1.-Unidad de anticipación de operandos



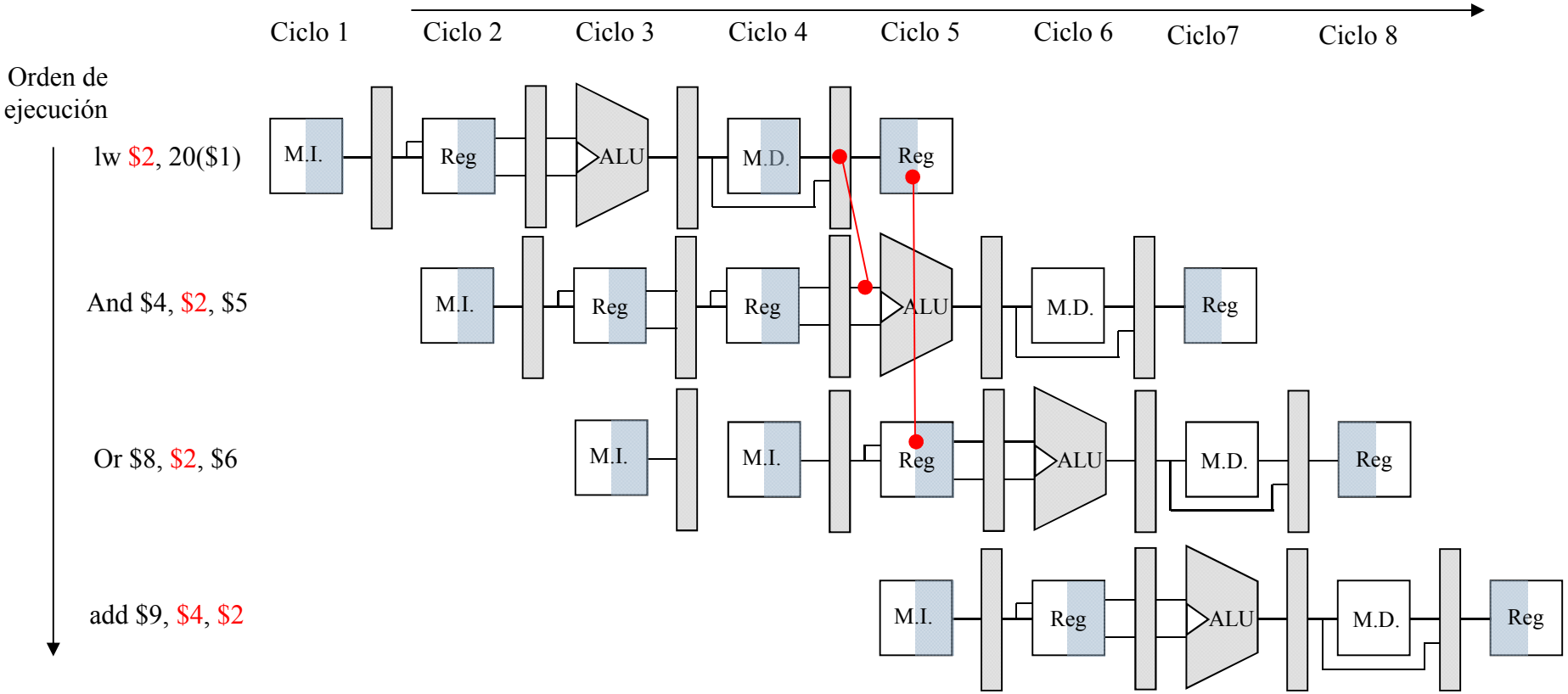
4.1.- La anticipación de operandos no siempre soluciona los riesgos



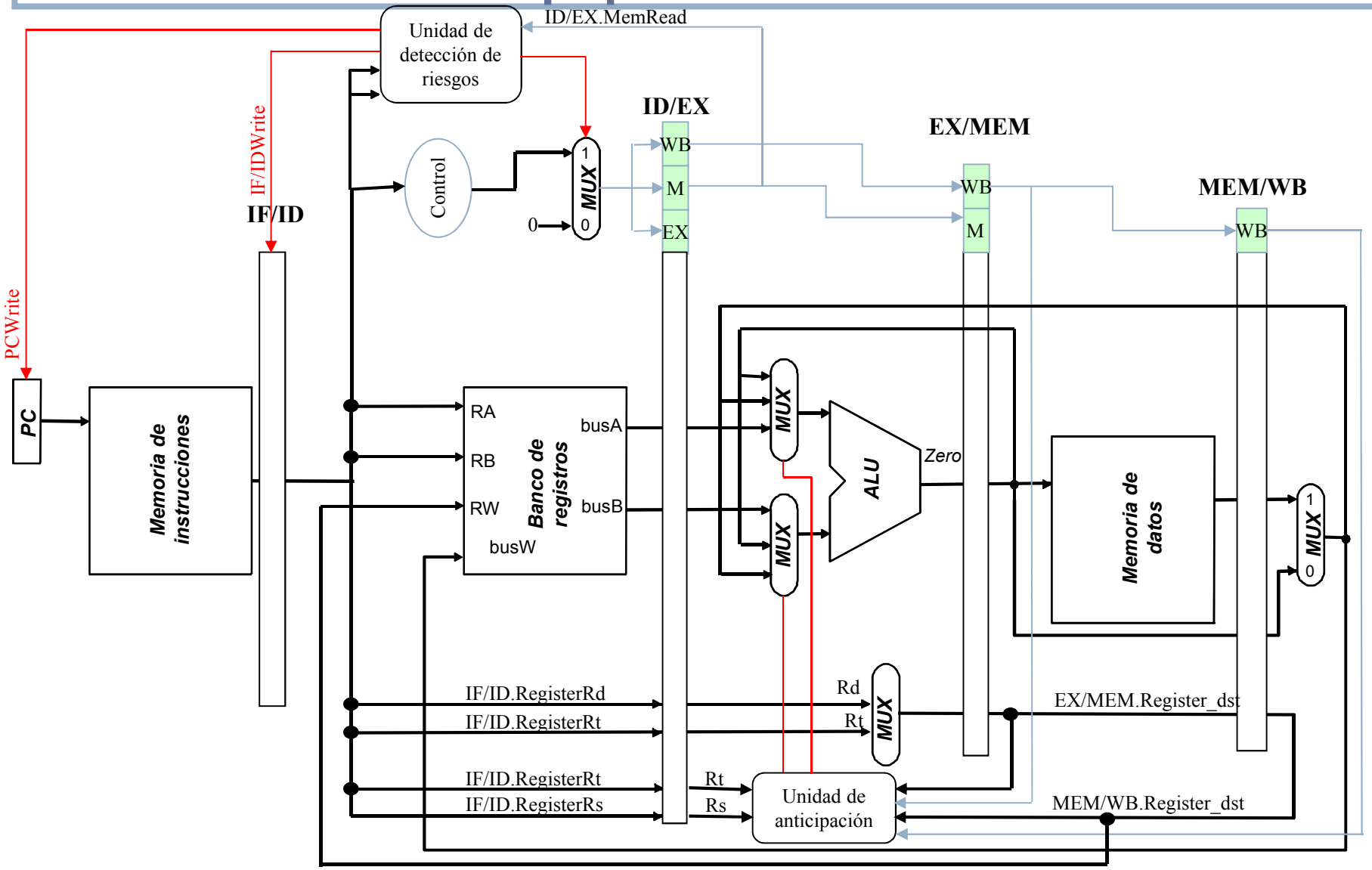
Necesitamos una unidad de detección de riesgos para ***detener*** a la instrucción load

4.1.- Detención del pipeline

- El conflicto LDE que se produce después de una instrucción **lw** no puede solucionarse.
- Hay que detener el pipeline manteniendo las instrucciones implicadas en la misma etapa, es decir evitando que el PC y el registro IF/ID cambien.
- Para detener el resto del pipeline podemos poner a 0 todos los puntos de control



4.1.- Unidad de detención del pipeline



4.1.- Reordenamiento el código

- Estrategia en tiempo de compilación:
  - ➢ Reordenar el código para minimizar el número de detenciones

```
lw    $2, 20($1)
and   $12, $2, $5
or    $13, $6, $2
add   $14, $5, $13
sub   $4, $5, $6
```

SOLUCIÓN

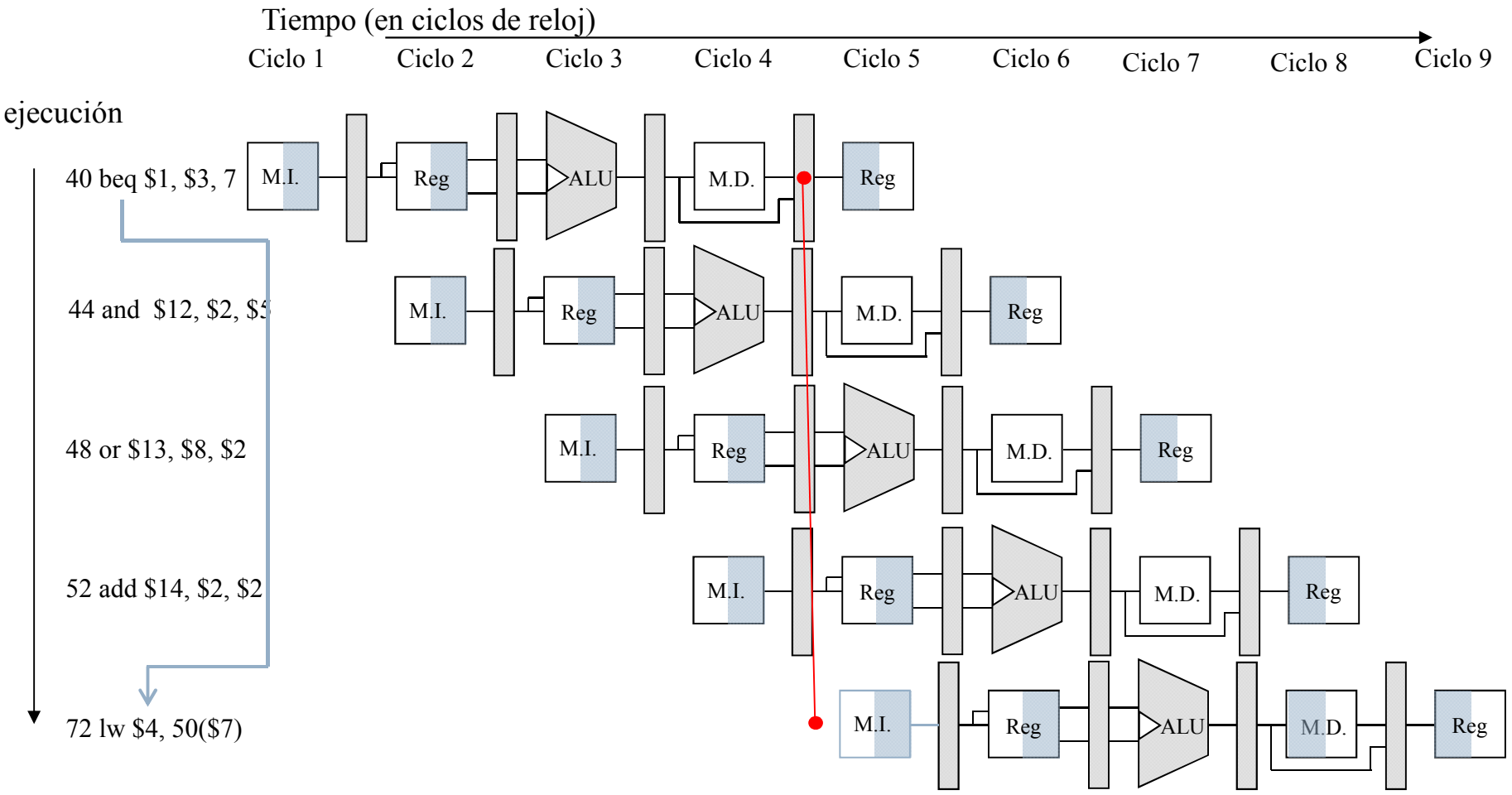
```
lw    $2, 20($1)
sub   $4, $5, $6
and   $12, $2, $5
or    $13, $6, $2
add   $14, $5, $13
```

Estrategias comunes en buenos compiladores:

- Loop unrolling *>> lo veremos en problemas*
- Software pipelining

4.2.- Riesgos de control

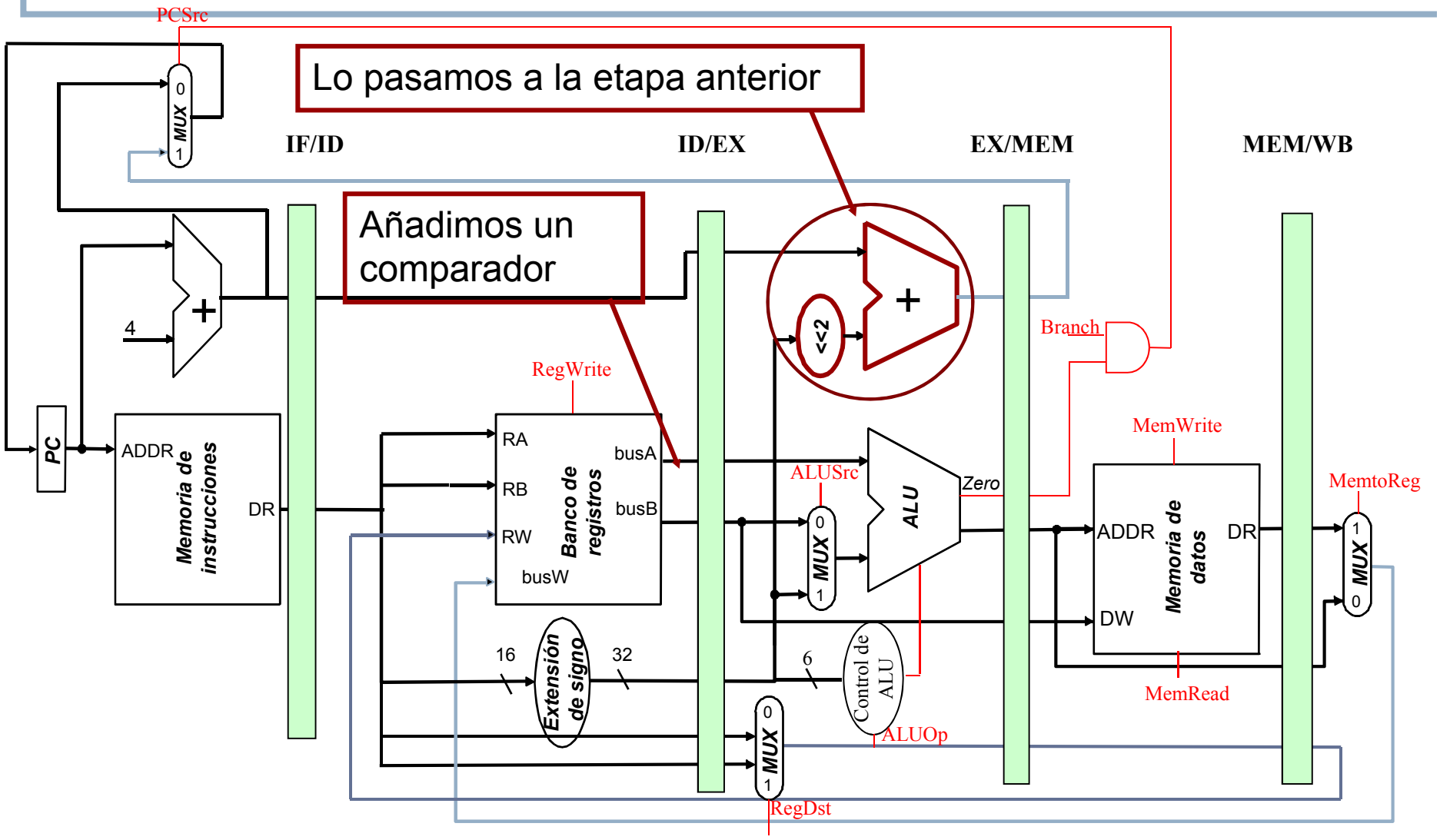
Salto condicional: no se conoce cuál es la siguiente instrucción hasta la etapa MEM



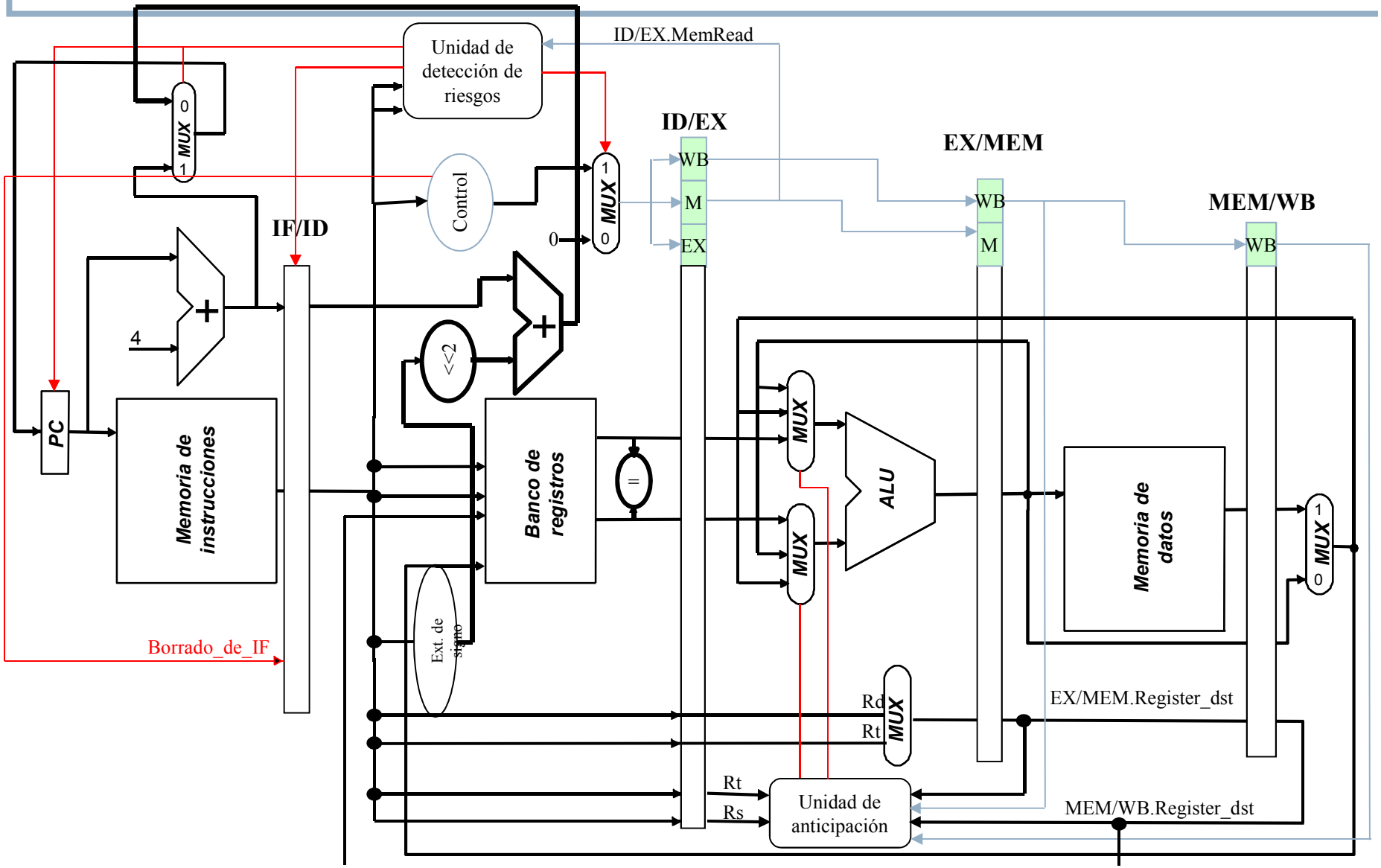
4.2.-Soluciones a los riesgos de control

- Reducir la penalización en los saltos
- Objetivo:
  - Calcular evaluación y destino lo antes posible
- Destino de salto
  - Etapa EX, pero tanto el sumador como la unidad << se podrían pasar a ID
- Evaluación del salto en BEQ (T / NT):
  - Comprobar si todos los bits de los dos operandos son iguales
    - Podemos realizarlo en ID con una XOR y una AND
    - La penalización sólo sería de 1 ciclo si salto NT

Cambios en la ruta de datos



4.2.-Soluciones a los riesgos de control



4.2.-Soluciones a los riesgos de control

- Detener el pipeline hasta que se conozca si el salto se produce.
  - Demasiado lenta.

- Asumir salto NT:
  - Buscar la siguiente instrucción secuencial
  - Si fallo de predicción (salto T):
    - Desechar todas las instrucciones que estén en ejecución equivocadamente.
    - Para ello se introducen ceros en sus valores de control.
  - Funciona bien si hay mucho saltos NT

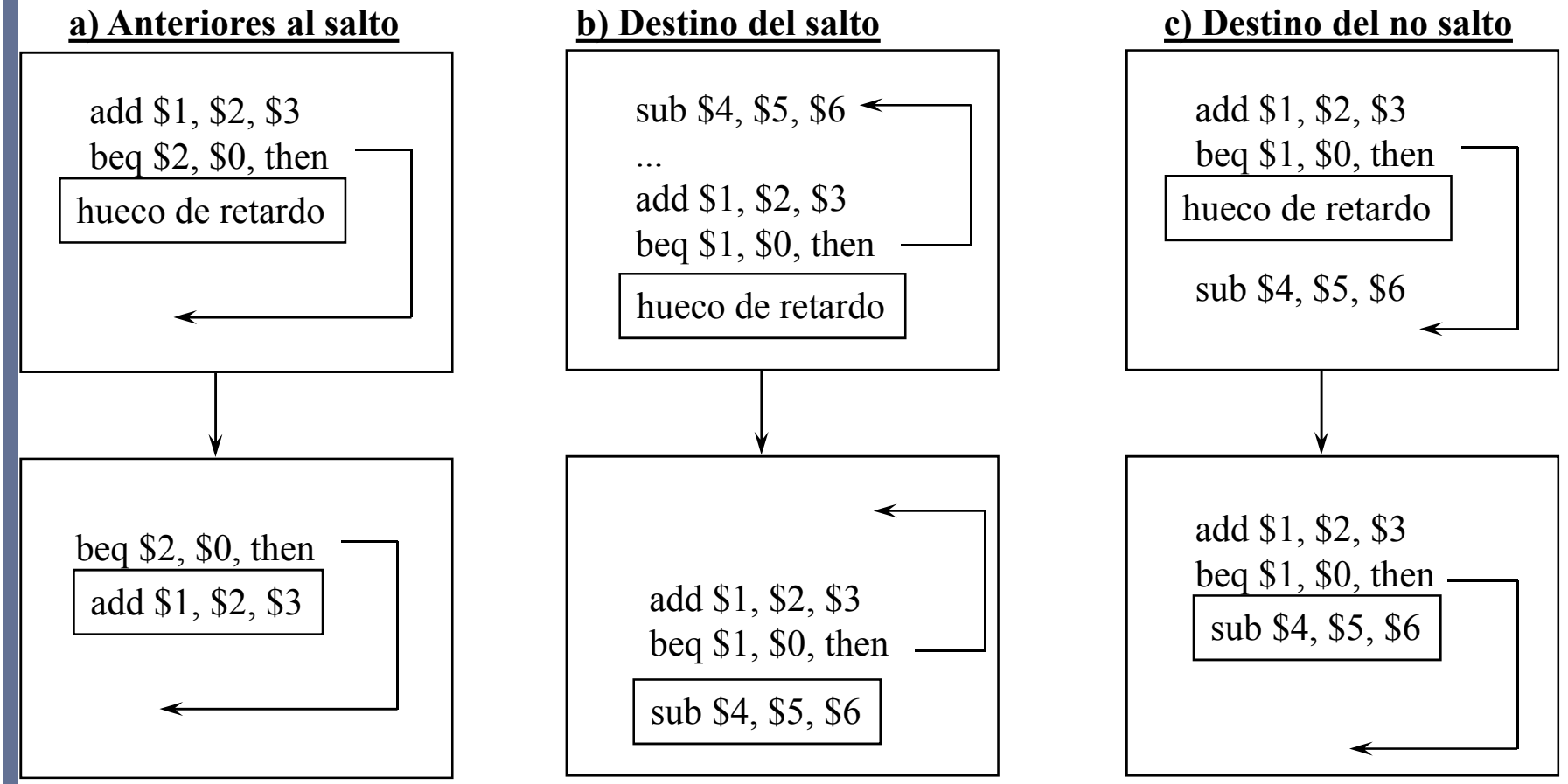
4.2.- Predicción dinámica de saltos

- Predicción dinámica de saltos:
  - Basada en los eventos pasados.
  - Predecimos que ocurrirá lo mismo que la vez anterior.
- El predictor almacena
  - la parte menos significativa de la dirección de la instrucción de salto
  - + un bit indicando si T o NT la última vez.
  - Por supuesto, esta información podría haberla guardado una instrucción diferente con los mismos bits finales (aliasing)
- Funciona bien en bucles

Saltos retardados

La siguiente instrucción a la de salto **siempre** se ejecuta.

El compilador debe seleccionar la instrucción adecuada:



Conclusiones

- La segmentación es una herramienta poderosa para mejorar el rendimiento sin aumentar demasiado el coste
- Para aplicarla se requiere:
  - Añadir registros extra
  - No reutilizar HW
  - Hacer que la información de control fluya con los datos
- Para conseguir resultados óptimos las etapas deben tener una duración parecidas
- Surgen dos nuevos problemas:
  - Los riesgos de datos:
    - Se pueden reducir añadiendo HW que busque los operandos allá donde se encuentren y separando las instrucciones con dependencias
  - Los riesgos de control:
    - Se pueden reducir anticipando la ejecución de las instrucciones beq, con estrategias de compilación y tratando de predecir si el salto se toma o no
  - Aun así en ocasiones habrá que detener el pipe