Tema 2. La CPU

Arquitectura de Computadores

Área de Arquitectura y Tecnología de Computadores
Departamento de Informática
Universidad de Oviedo

Curso 2021–2022

Objetivos

1.- Mejoras del rendimiento

Cambios organizativos para mejorar el rendimiento

2.- Soporte a los sistemas operativos multitarea

Funcionalidad necesaria para su correcto funcionamiento

3.- Soporte a la virtualización

Funcionalidad necesaria para ejecución eficiente

Arquitectura de ejemplo

MIPS64

- Primera de 64 bits
- Evolución de MIPS 32 bits
  - Compatible
- Muy parecida a ARMv8 (dispositivos móviles)

RISC

- Instrucciones sencillas de tamaño fijo
- Pocos modos de direccionamiento
- Gran número de registros

MIPS64

ISA (carga/almacenamiento)

- Ancho de la CPU
  - ancho de los operandos: 64 bits
- Espacio de memoria (m)
  - líneas en el bus de direcciones (64) $\Rightarrow m = 2^{64}$
- Memoria direccionable al byte
  - byte, media palabra, palabra, doble palabra
  - solo instrucciones de carga/almacenamiento
- Conjunto de registros (64 bits)
  - 32 enteros: `r1`-`r31` (`r0`= 0)
  - 32 reales en coma flotante de precisión doble: `f0`-`f31`

Tipos de datos

Enteros (complemento a dos)

- Byte: 8 bits
- Media palabra: 16 bits
- Palabra: 32 bits
- Doble palabra: 64 bits

Reales (IEEE-754)

- Precisión simple: 32 bits
- Precisión doble: 64 bits

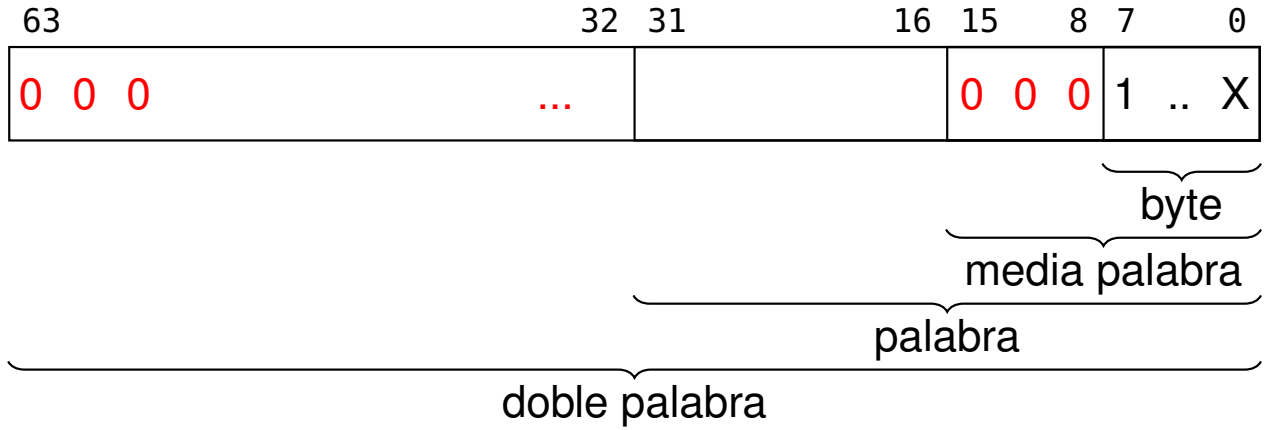
Acceso a varios bytes

- Endianity
- Accesos alineados

Tipos de datos

Carga de byte sin signo

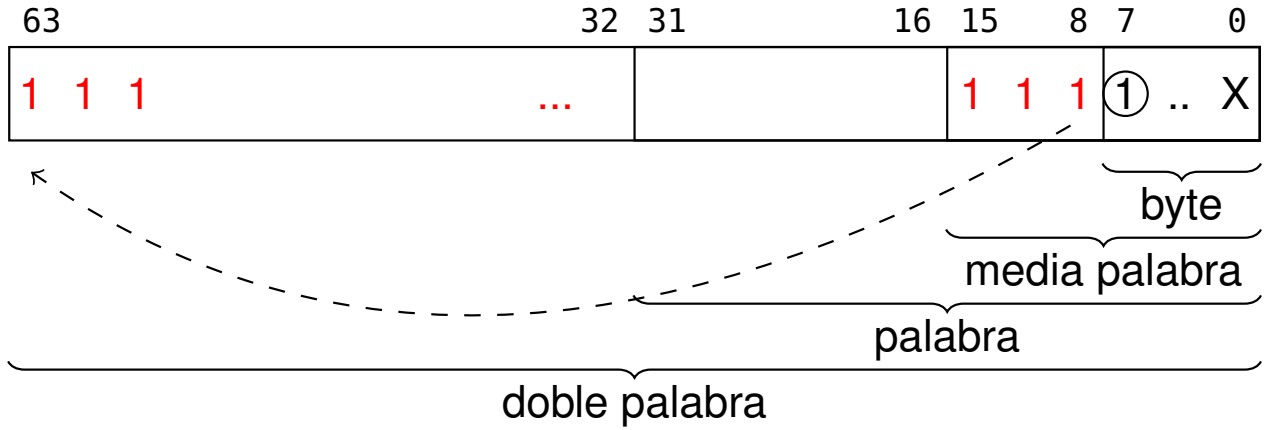
```
lbu r4, 100(r0)
```



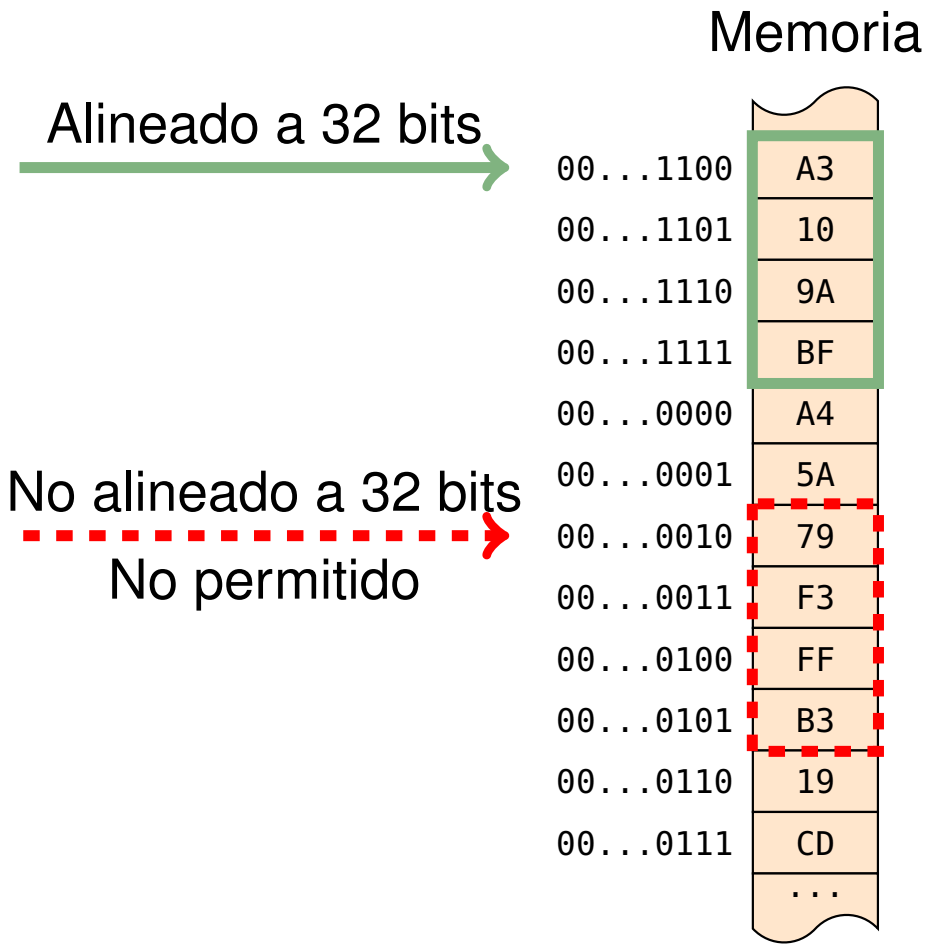
Tipos de datos

Carga de byte con signo

```
lb r4, 100(r0)
```



Accesos alineados



Juego de instrucciones

- Aritméticas
- Lógicas
- Carga/Almacenamiento
- Saltos incondicionales
- Saltos condicionales
- Desplazamientos
- Copia/Conversione

Modos de direccionamiento

Ubicación de los operandos

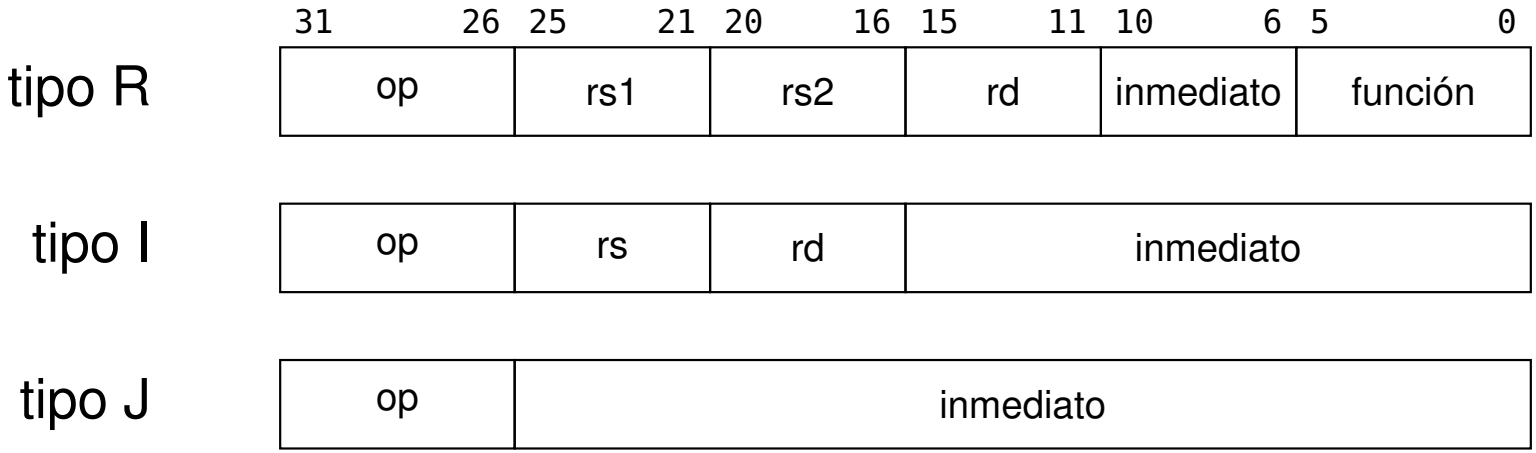
- Código de la instrucción
- Registro
- Memoria

Modos

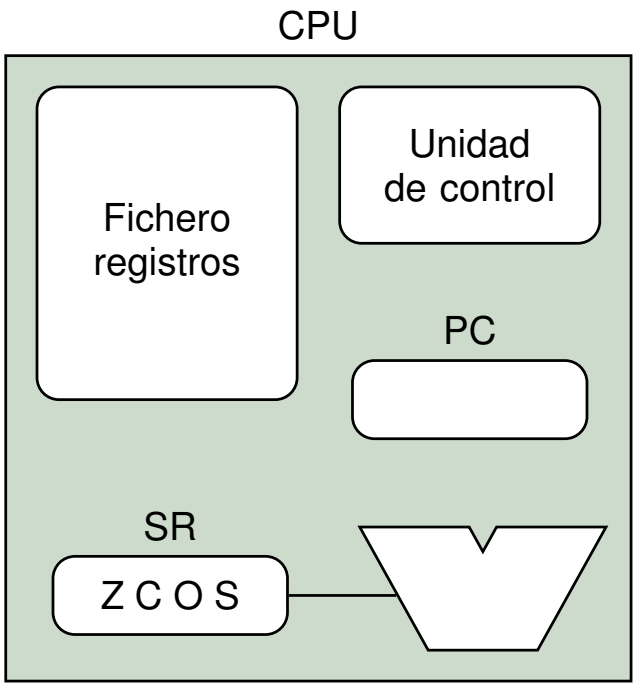
- Inmediato: `daddi r4, r8, -2`
- Registro: `dadd r2, r18, r3`
- Base más desplazamiento: `ld r6, 100(r3)`
- Relativo al PC: `beq r2, r9, -5`
- Pseudodirecto: `j 1000`

Codificación

Todas el mismo tamaño: 32 bits



Ejecución de instrucciones



Condiciones

- Instrucciones en memoria
- PC apunta a la siguiente instruccion

Repetir

- Búsqueda de instrucción
- Incrementar PC
- Decodificar la instrucción
- Búsqueda de operandos
- Ejecución de la instrucción
- (carga/almacenamiento) Acceso a memoria
- Almacenar los resultados

Ejemplo

- `int32 a; // (en registro **r10**)`
- `int32 b; // (en direccion 1534h)`

C

```
if (a == b)
  a = 0;
```

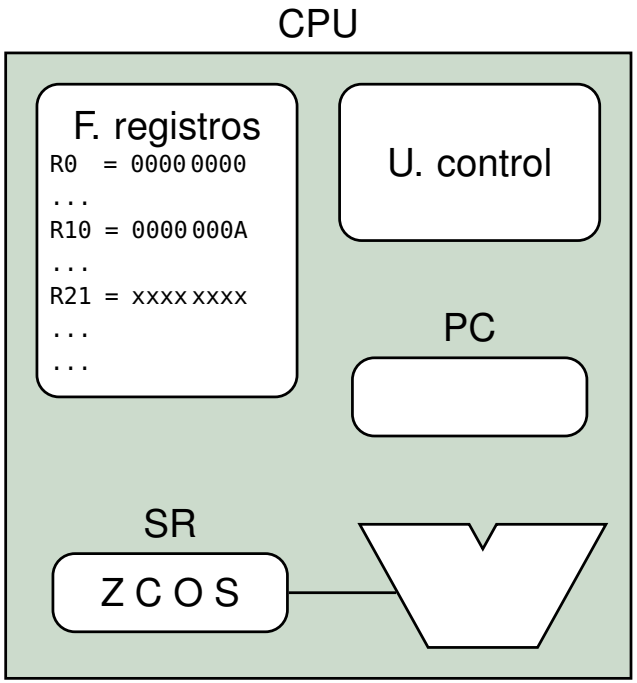
MIPS64

```
     lw    r21, 0x1534(r0)
     bne   r21, r10, skip
     xor   r10, r10, r10
skip:
```

C. máquina

```
8C151534h
15550001h
014A5026h
```

Ejemplo

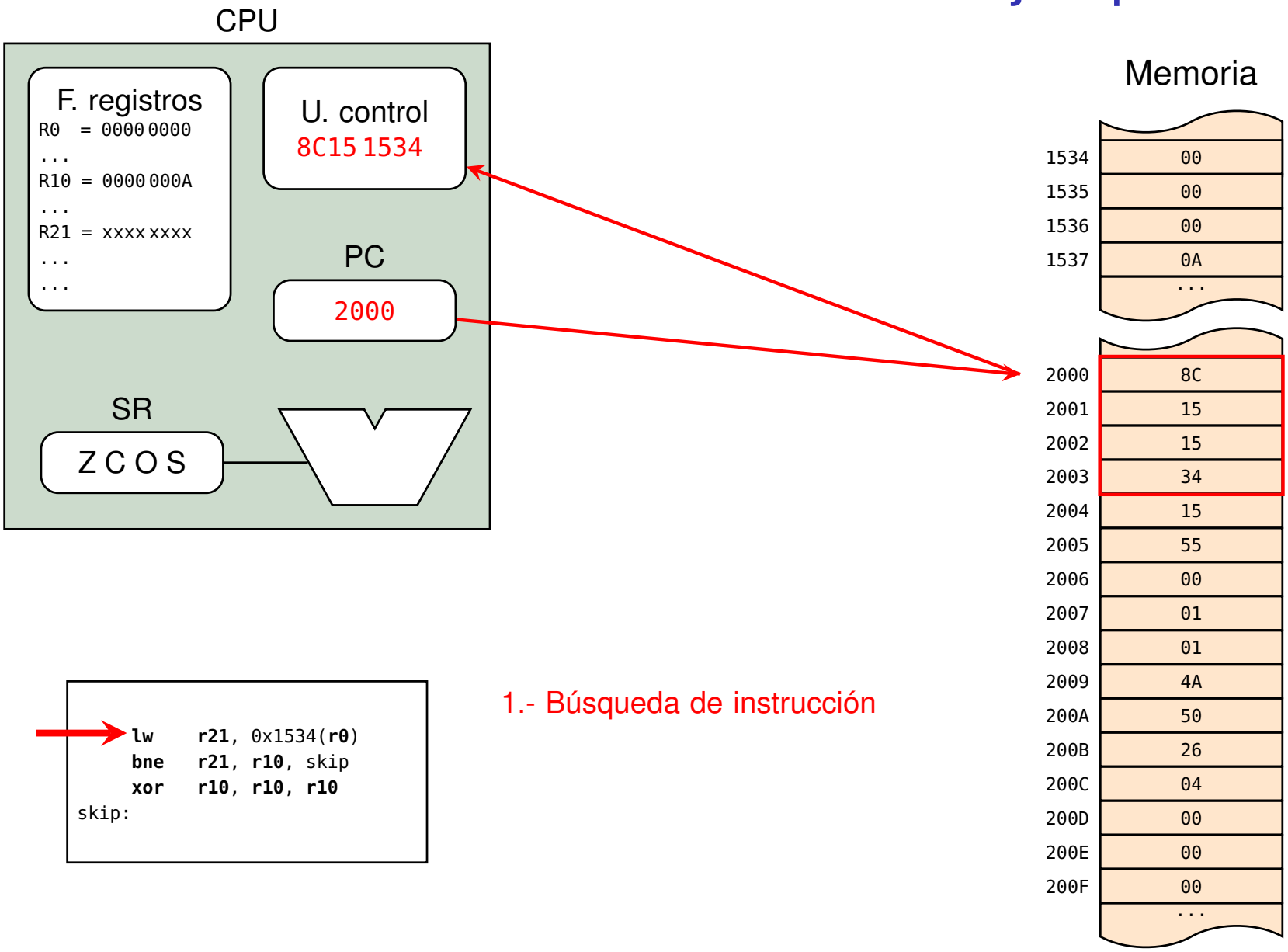



```
    lw    r21, 0x1534(r0)
    bne   r21, r10, skip
    xor   r10, r10, r10
skip:
```

Memoria

1534	00
1535	00
1536	00
1537	0A
...	
2000	8C
2001	15
2002	15
2003	34
2004	15
2005	55
2006	00
2007	01
2008	01
2009	4A
200A	50
200B	26
200C	04
200D	00
200E	00
200F	00
...	

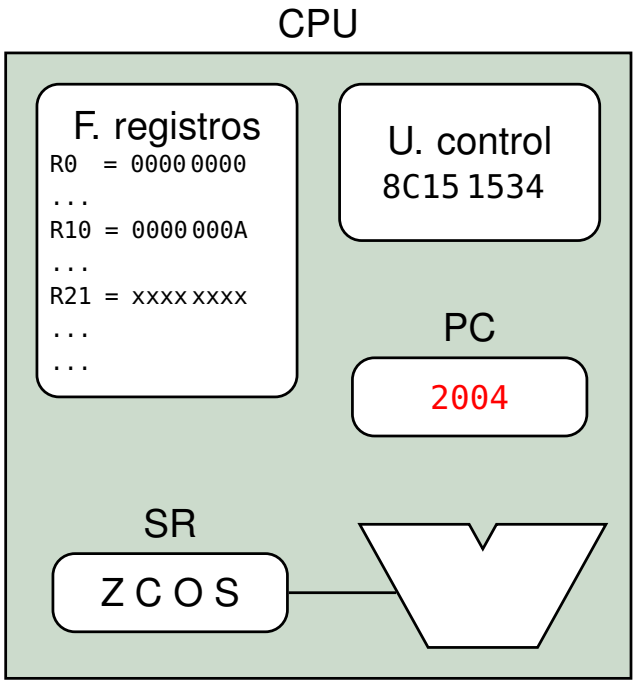
Ejemplo

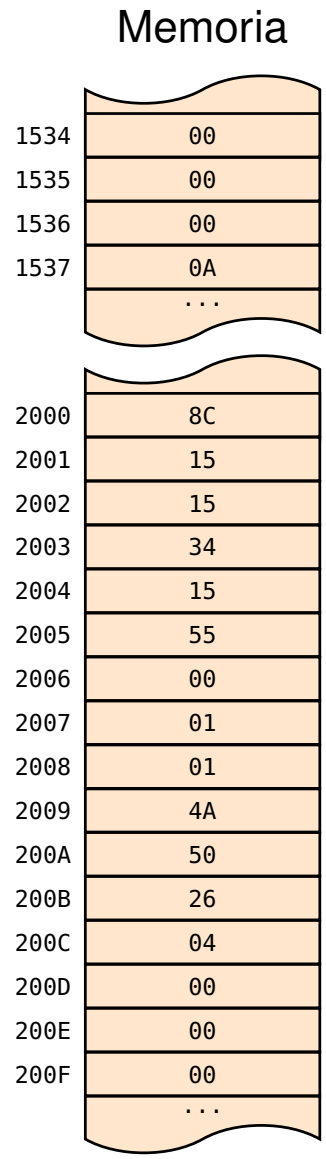



```
    lw    r21, 0x1534(r0)
    bne   r21, r10, skip
    xor   r10, r10, r10
skip:
```

1.- Búsqueda de instrucción

Ejemplo



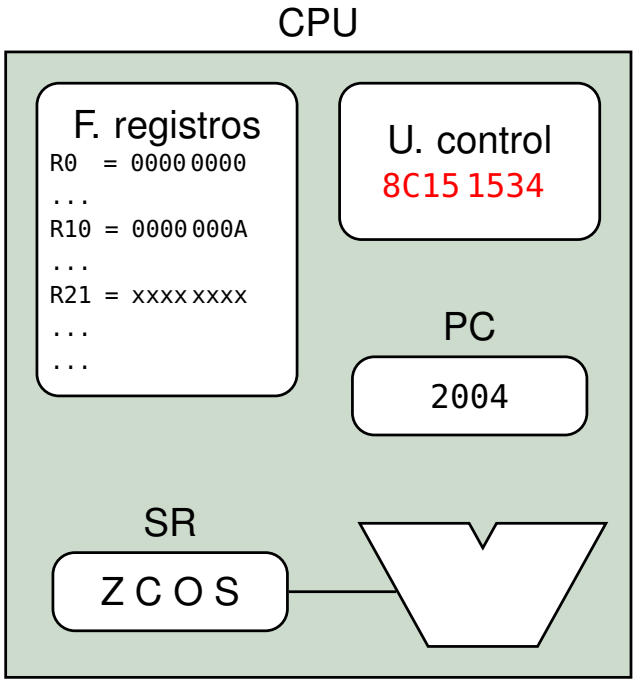



```
    lw    r21, 0x1534(r0)
    bne   r21, r10, skip
    xor   r10, r10, r10
skip:
```

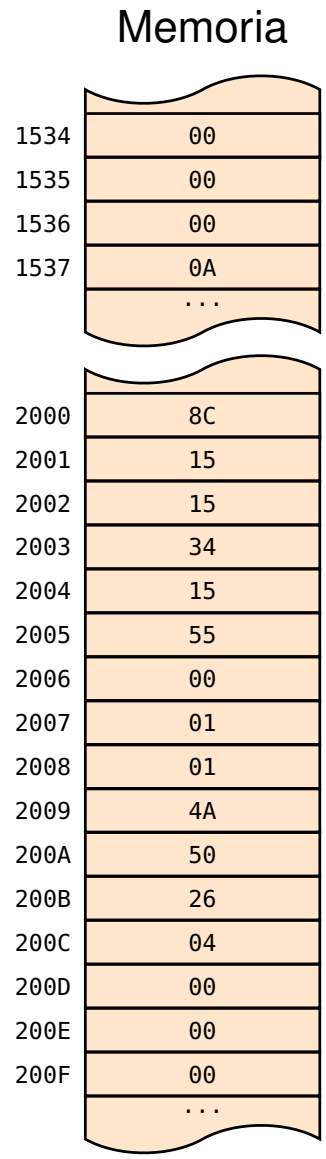
2.- Incrementar PC (+4)

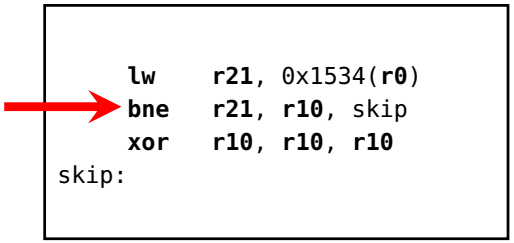
2000 + 4

Ejemplo



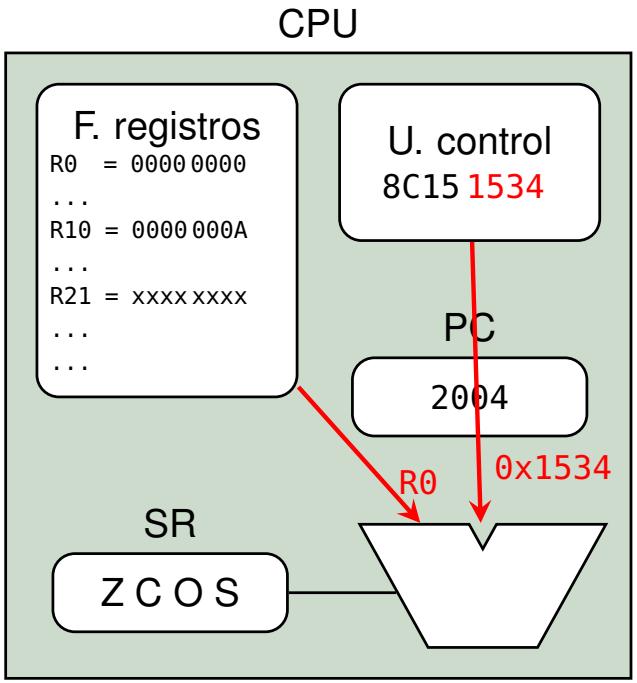
lw r21, 0x1534(**r0**)



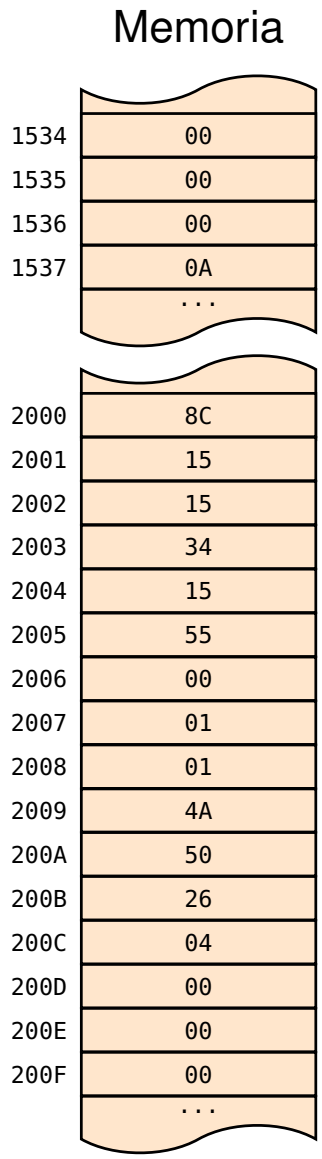


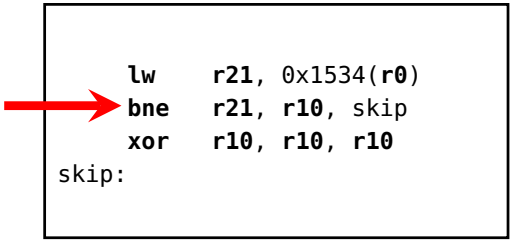
3.- Decodificar la instrucción

Ejemplo



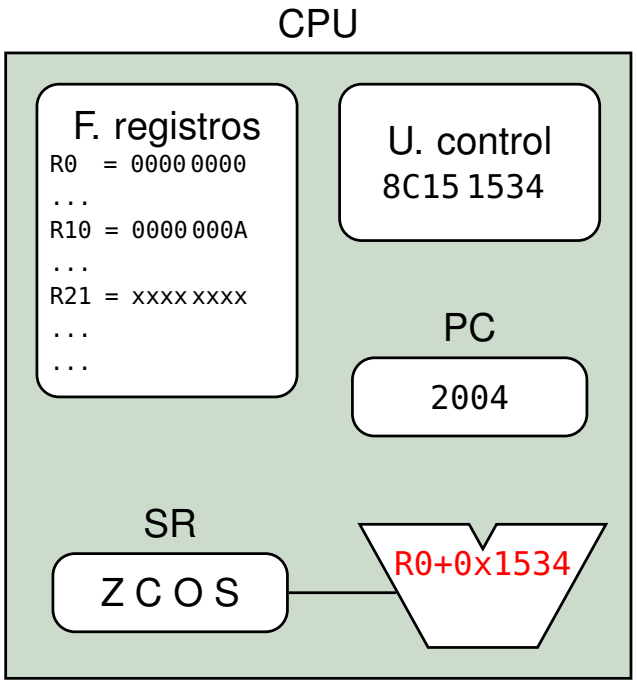
lw r21, 0x1534(**r0**)



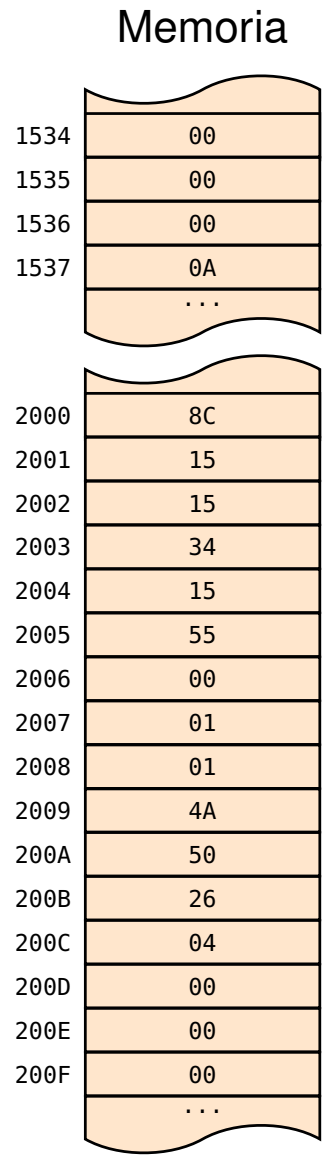


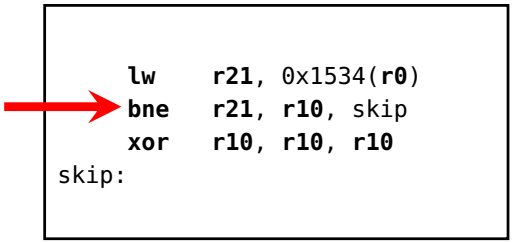
4.- Búsqueda de operandos

Ejemplo



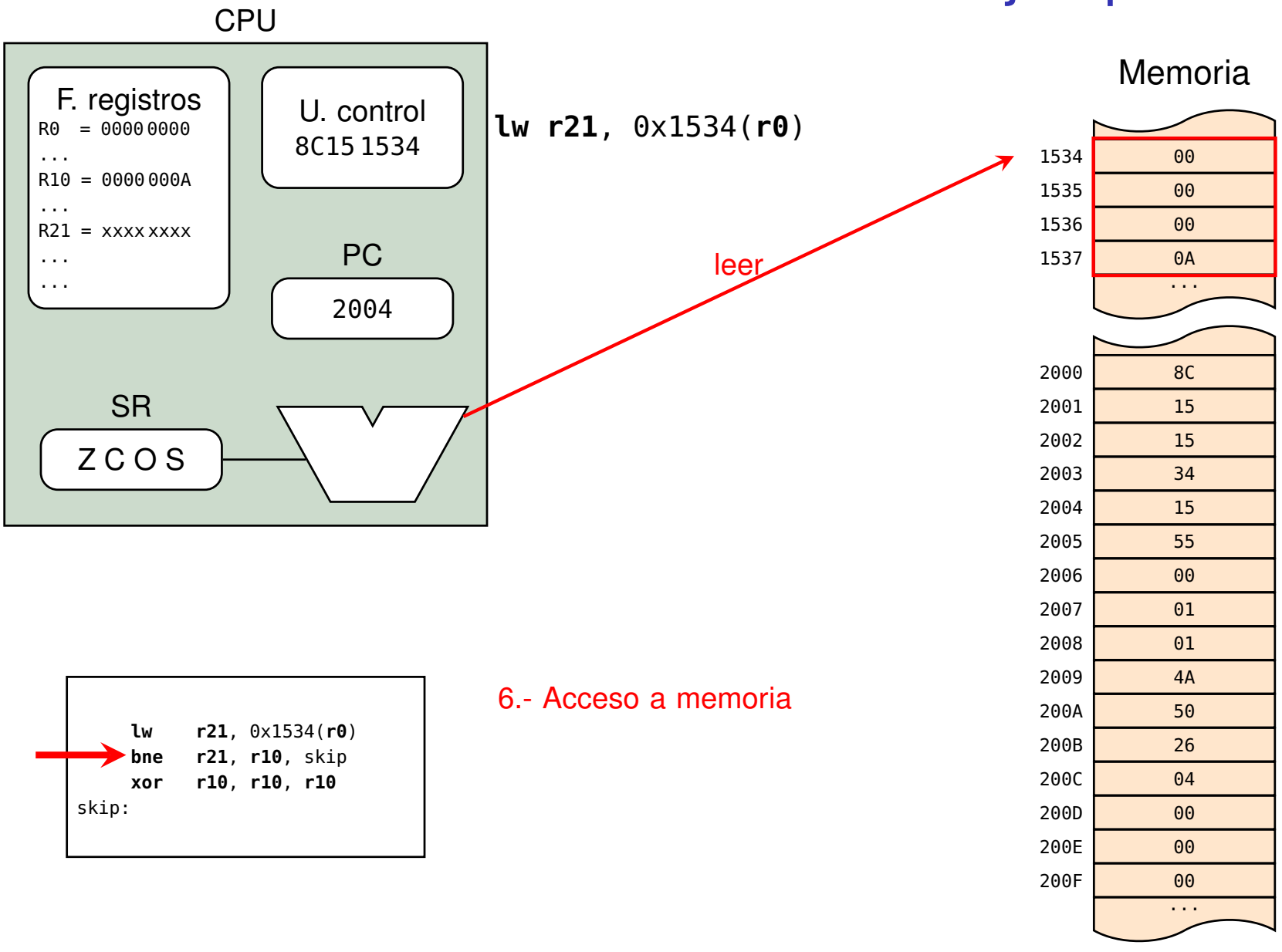
lw r21, 0x1534(**r0**)





5.- Ejecución de la instrucción

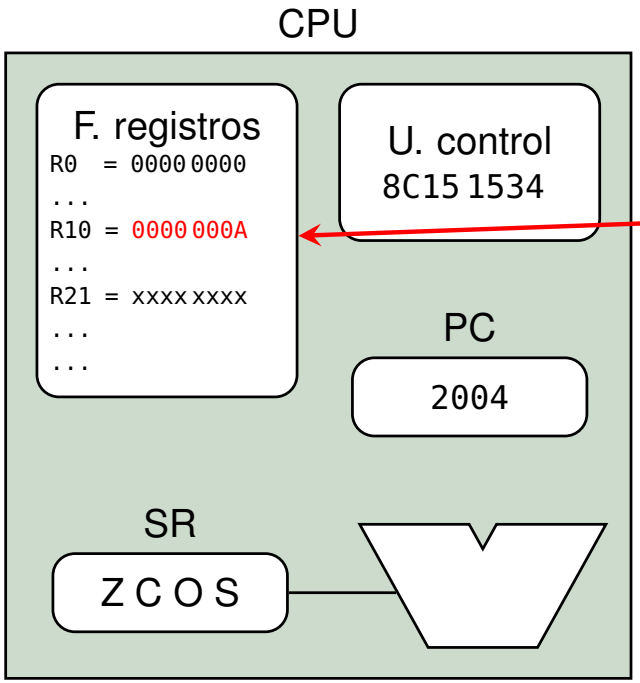
Ejemplo



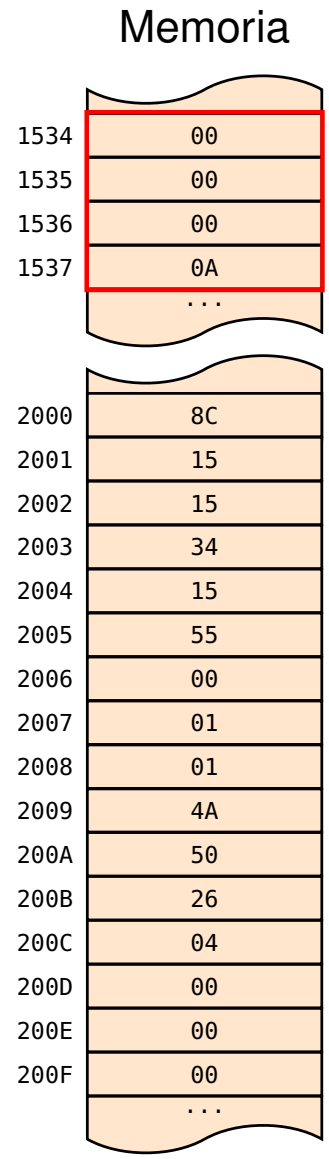

```
    lw    r21, 0x1534(r0)
    bne   r21, r10, skip
    xor   r10, r10, r10
skip:
```

6.- Acceso a memoria

Ejemplo



lw r21, 0x1534(r0)




```
    lw    r21, 0x1534(r0)
    bne   r21, r10, skip
    xor   r10, r10, r10
skip:
```

7.- Almacenar los resultados

Microarquitectura

Simplificación

Instrucción	Tipo	Descripción
`ld Rd,inm_16(Ri)`	I	Carga de doble palabra
`sd Rd,inm_16(Ri)`	I	Almacenamiento de doble palabra
`beq Rs1,Rs2,inm_16`	I	Salto condicional si los registros son iguales
`daddi Rd,Rs,inm_16`	I	Suma de registro y valor inmediato
`j inm_26`	J	Salto incondicional
`dadd Rd,Rs1,Rs2`	R	Suma de registros
`dsub Rd,Rs1,Rs2`	R	Resta de registros
`and Rd,Rs1,Rs2`	R	Operación lógica AND
`or Rd,Rs1,Rs2`	R	Operación lógica OR
`slt Rd,Rs1,Rs2`	R	`(Rd = Rs1 < Rs2 ? 1 : 0)`

Requisitos

Idea básica

Replicar hardware

- ejecutar todas las instrucciones con CPI = 1

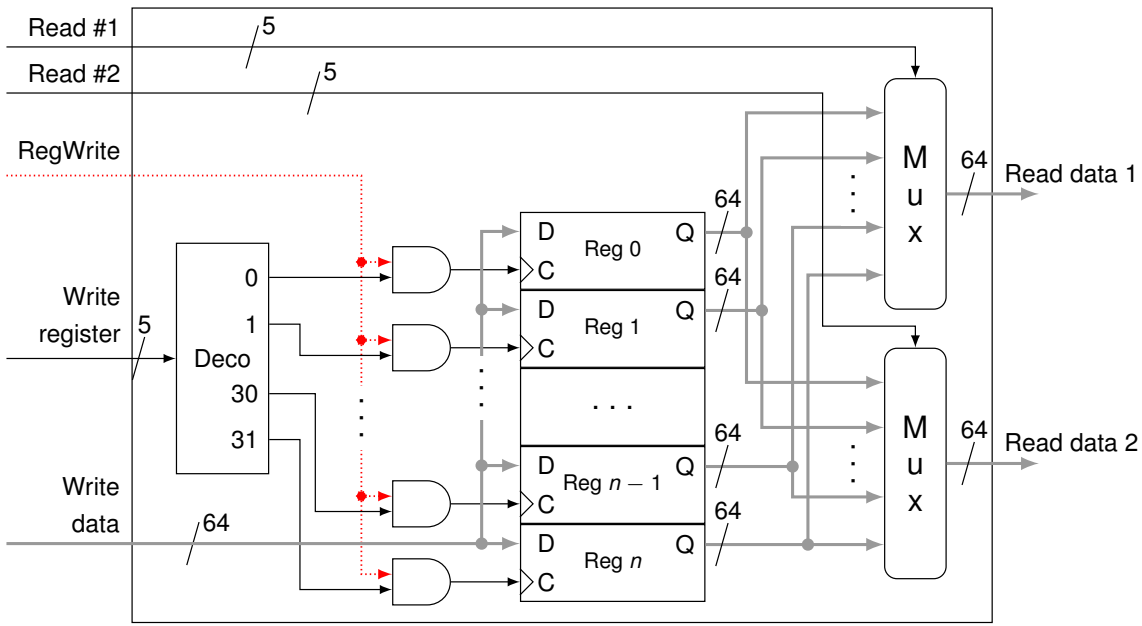
Fases comunes

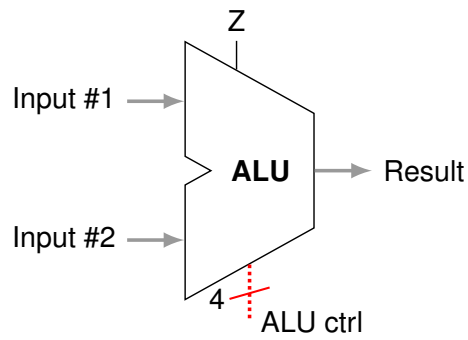
- Buscar el código de instrucción
- Decodificar la instrucción

Unidades funcionales

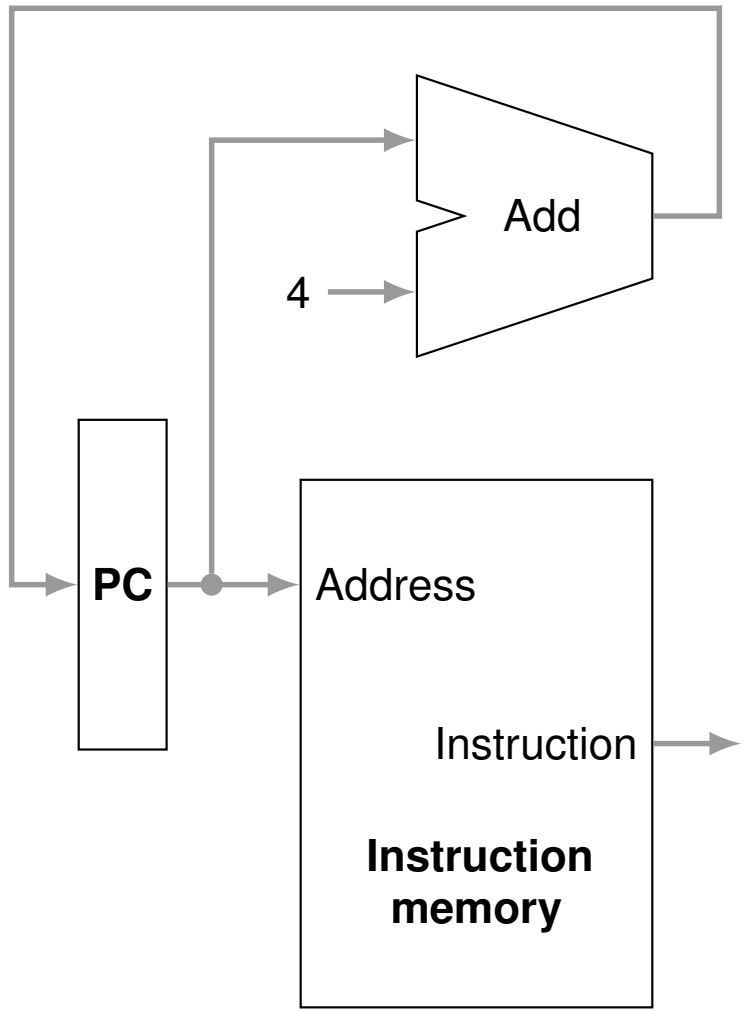
Elementos constructivos del camino de datos

- Registro PC
- Memoria de instrucciones/datos
- Fichero de registros
- ALU
- Más: sumadores, desplazadores, multiplexores





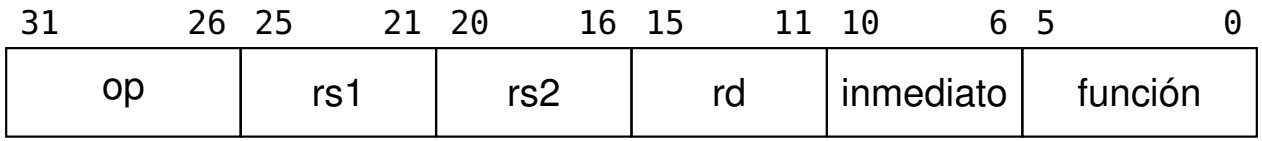
Búsqueda del código de instrucción



Instrucciones tipo R

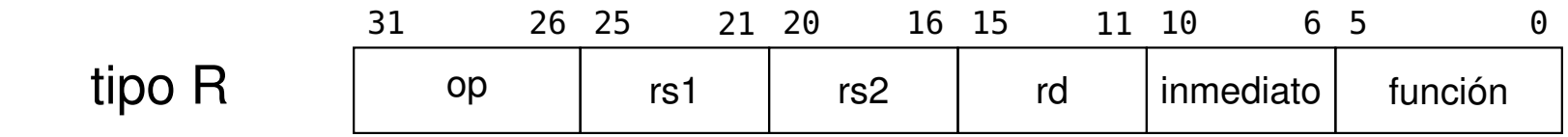
Instrucción	Tipo	Descripción
dadd Rd,Rs1,Rs2	R	Suma de registros (doble palabra)
dsub Rd,Rs1,Rs2	R	Resta de registros (doble palabra)
and Rd,Rs1,Rs2	R	Operación lógica AND
or Rd,Rs1,Rs2	R	Operación lógica OR
slt Rd,Rs1,Rs2	R	Comparación entera (if Rs1<Rs2 Rd=1; else Rd=0)

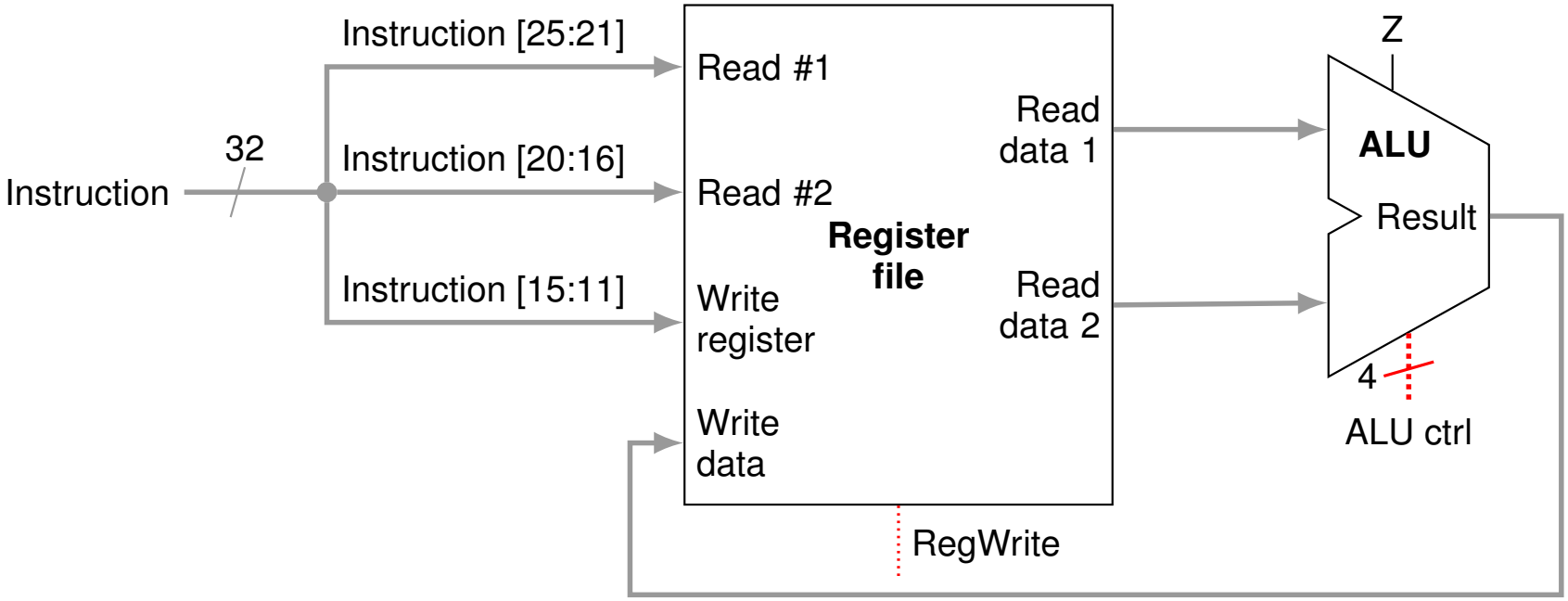
tipo R



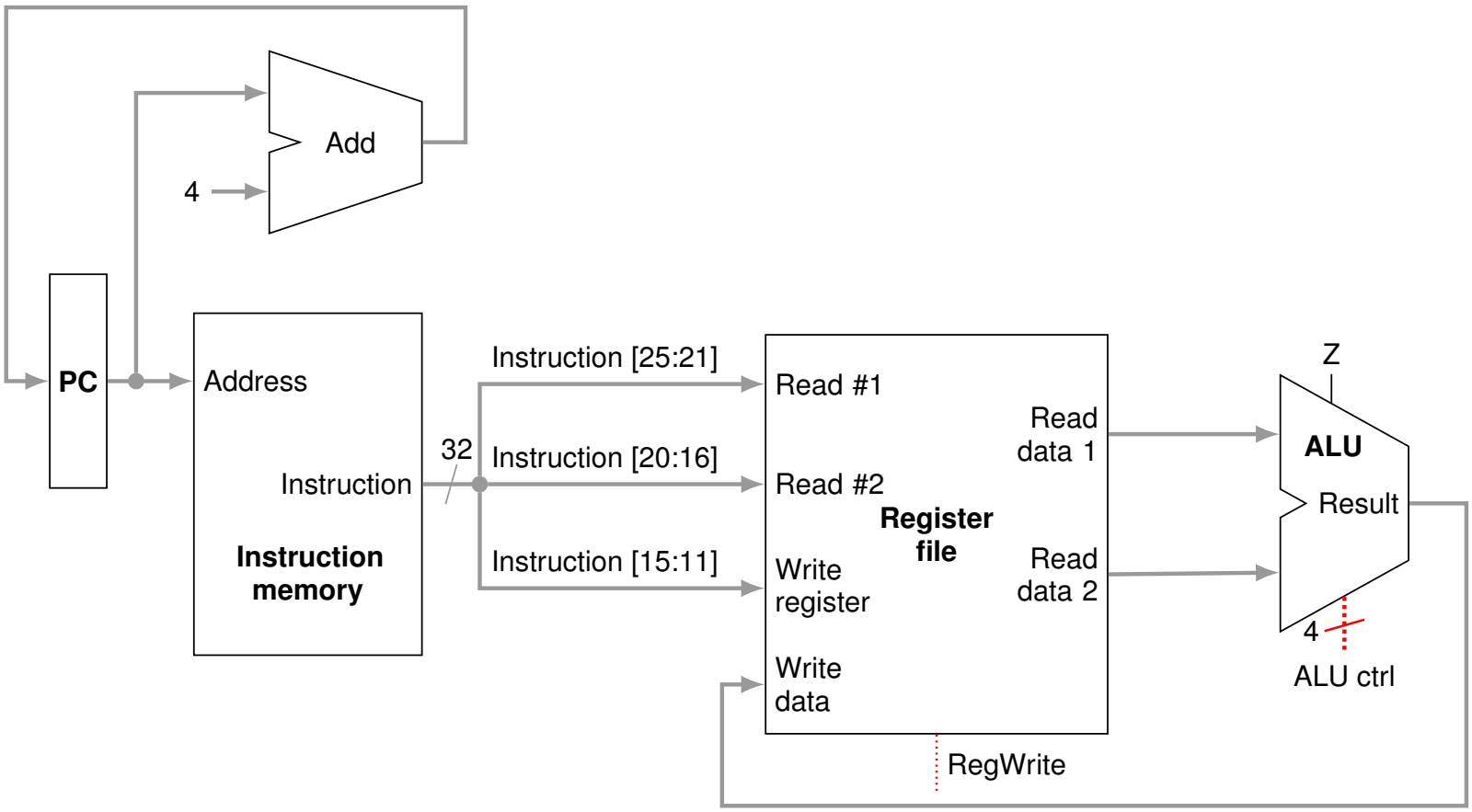
Ejecución de instrucciones de tipo R

```
dadd r2, r5, r8
```





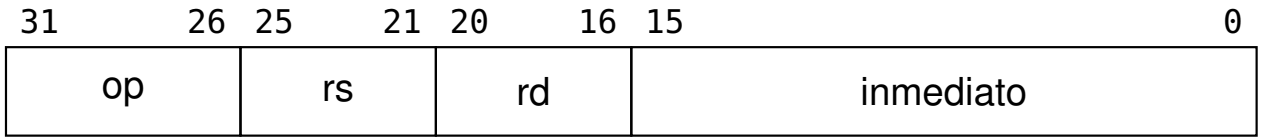
Datapath para instrucciones tipo R



Instrucciones carga/almacenamiento

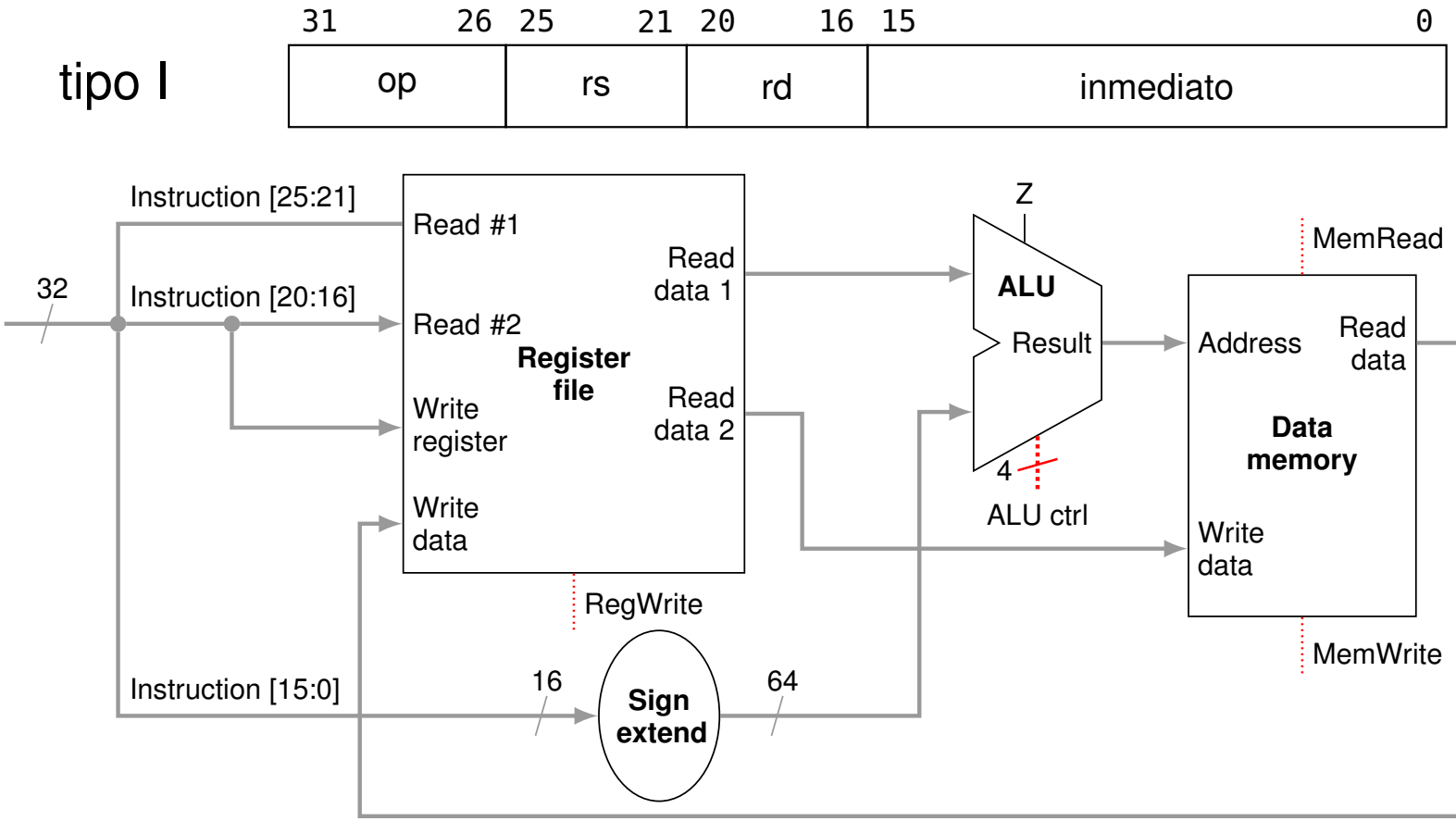
Instrucción	Tipo	Descripción
ld Rd,inm_16(Ri)	I	Carga de doble palabra
sd Rd,inm_16(Ri)	I	Almacenamiento de doble palabra

tipo I

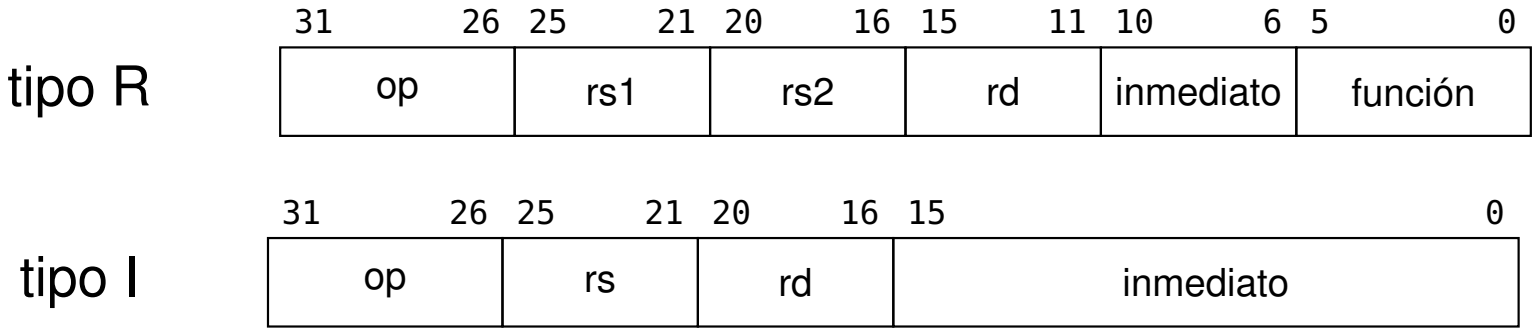


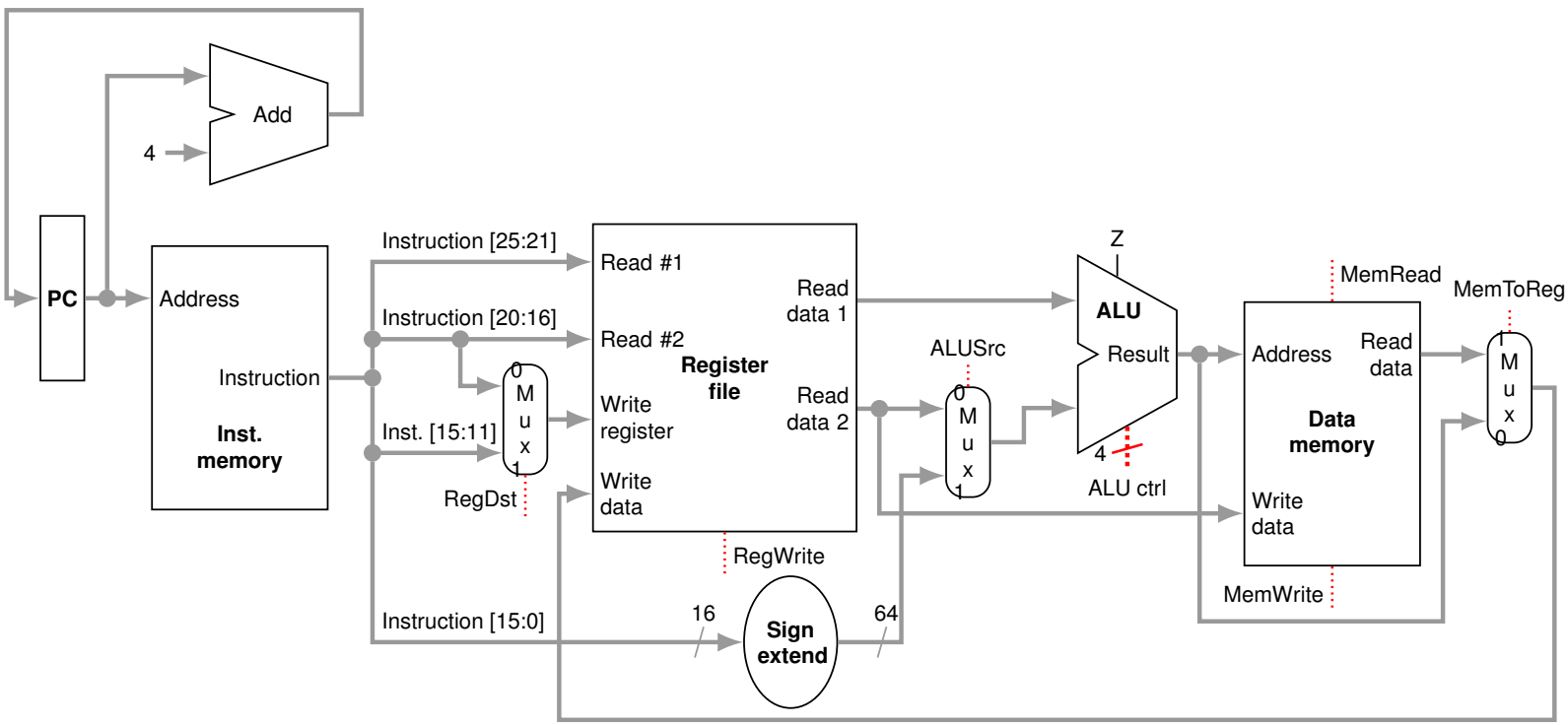
Ejec. instr. carga/almacenamiento

`ld r4, 200(r2)` / **`sd r2, 100(r4)`**



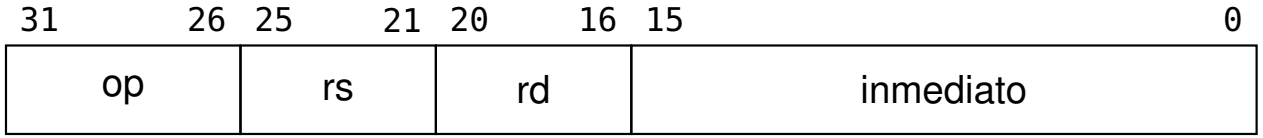
Datapath inst. R y carga/almac.

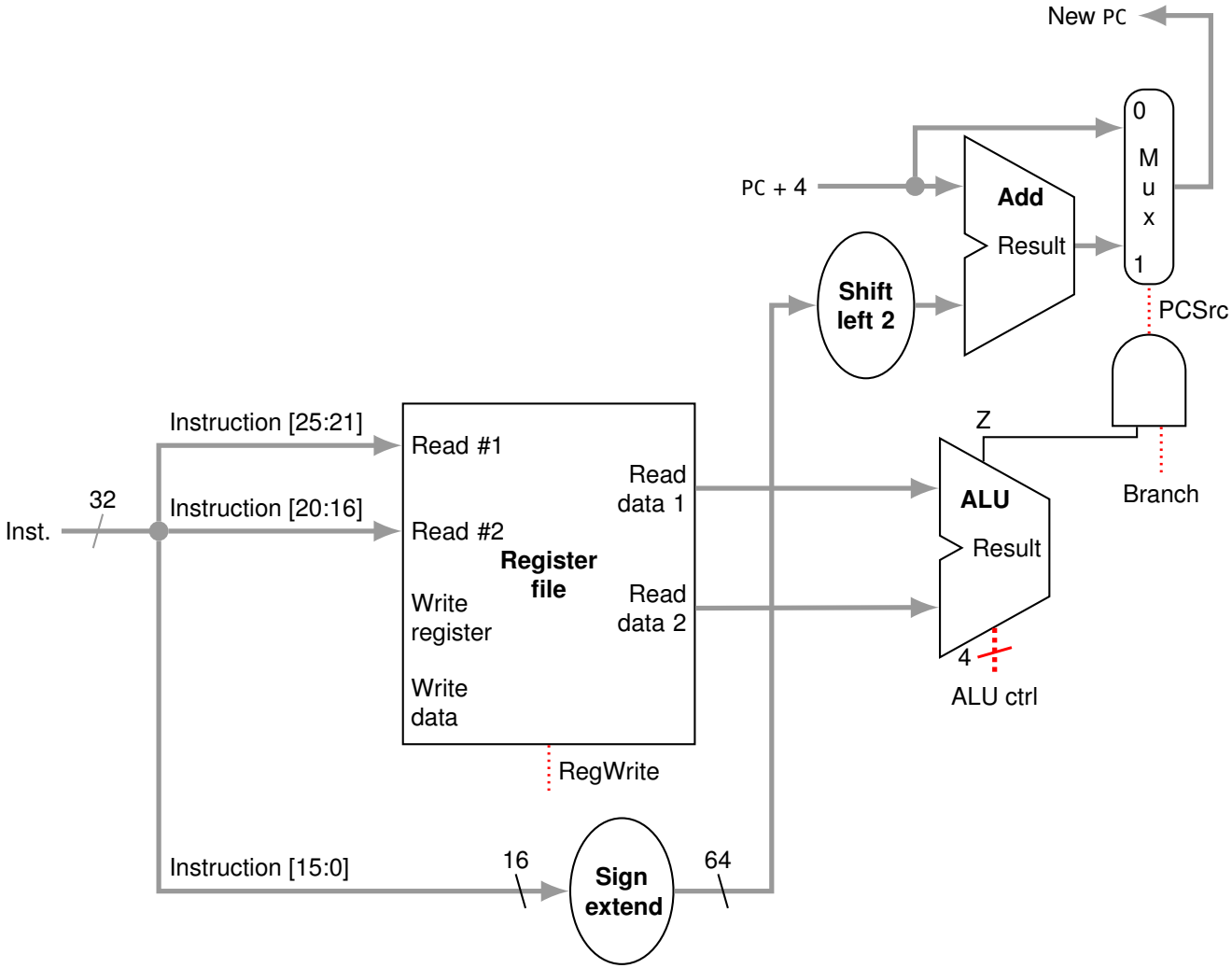




Ejec. de la instrucción `beq Rs1,Rs2,inm_16`

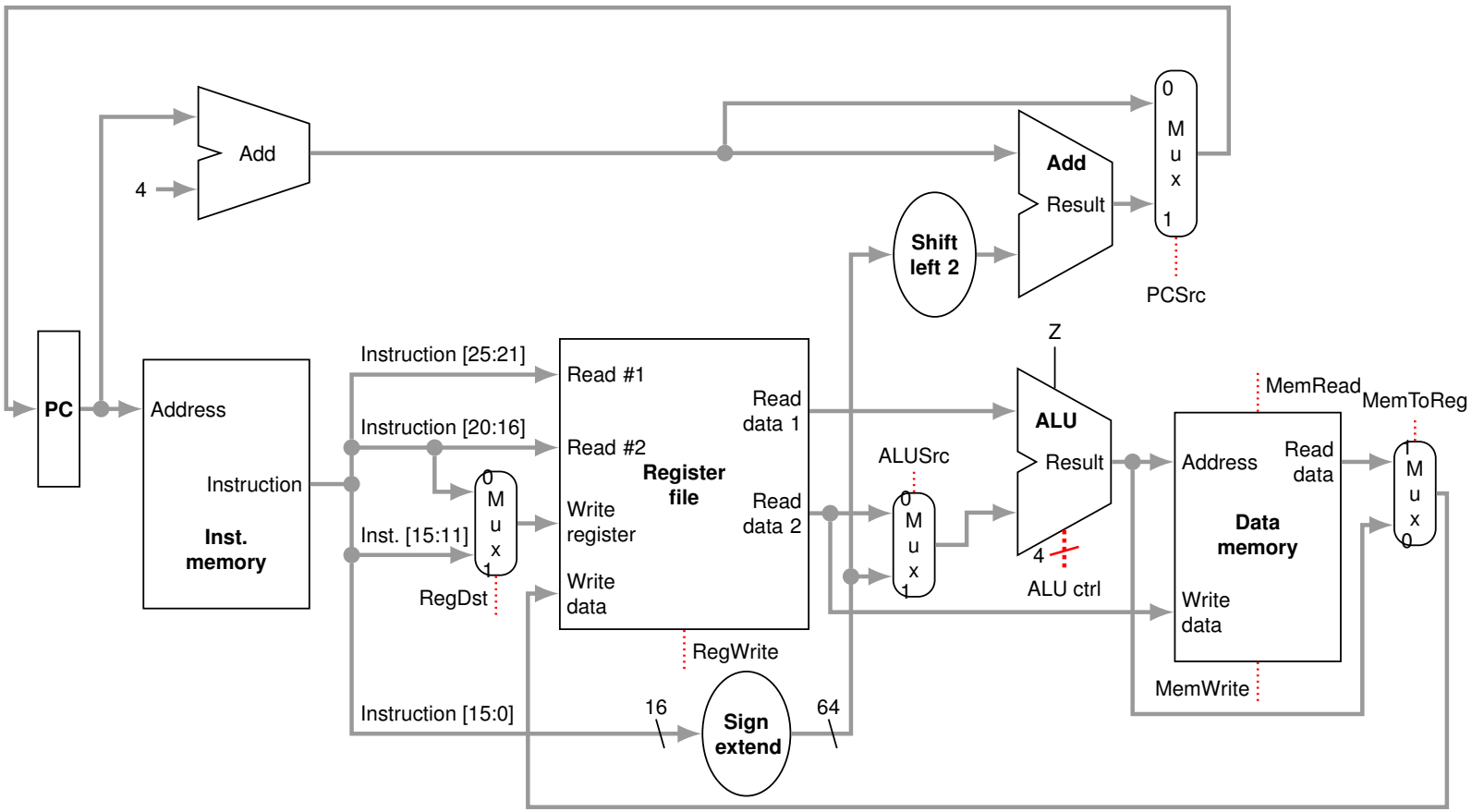
tipo I





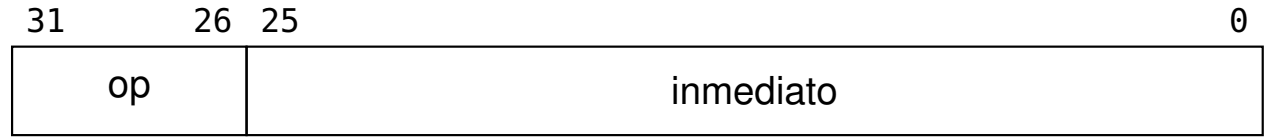
Camino de datos: Tipo R, carga/alm.,

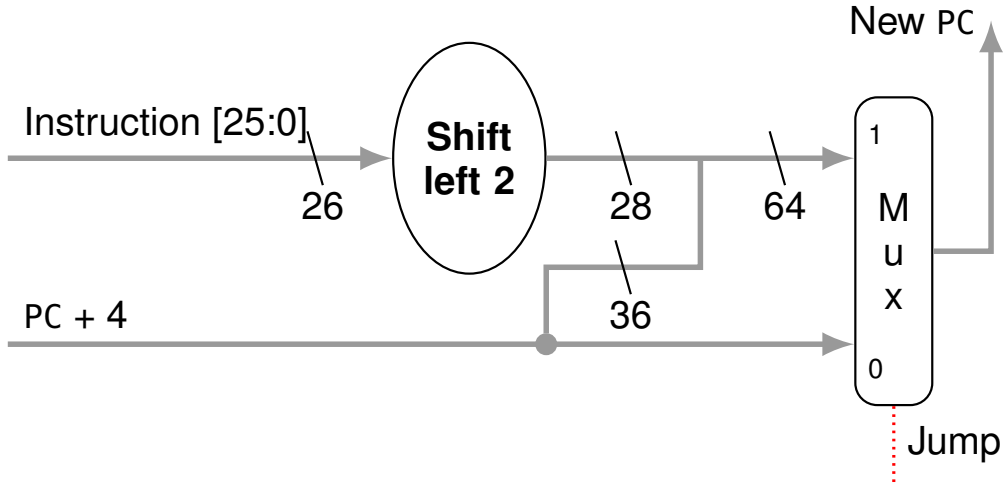
beq



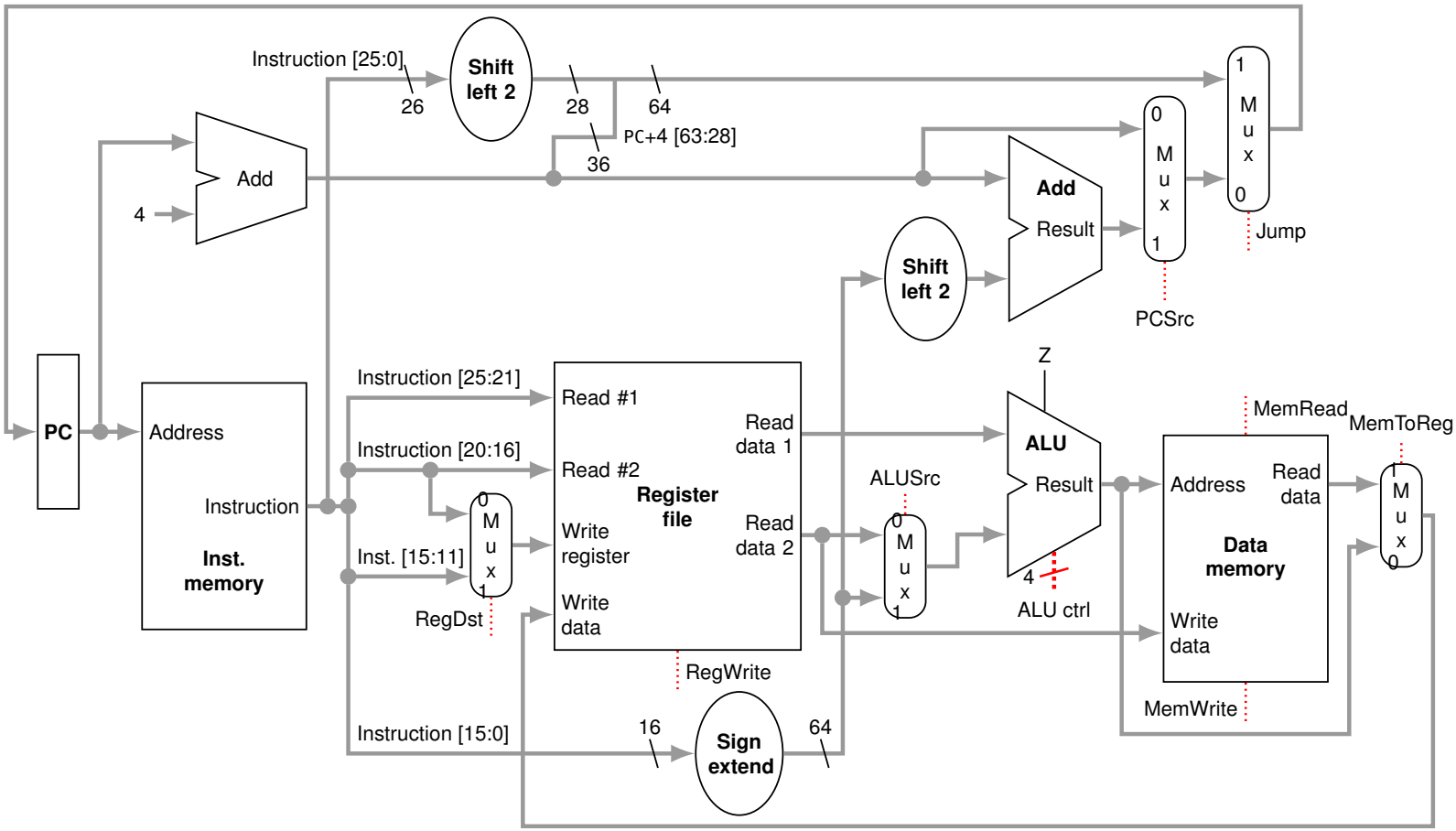
Ejec. de la instrucción `j inm_26`

tipo J



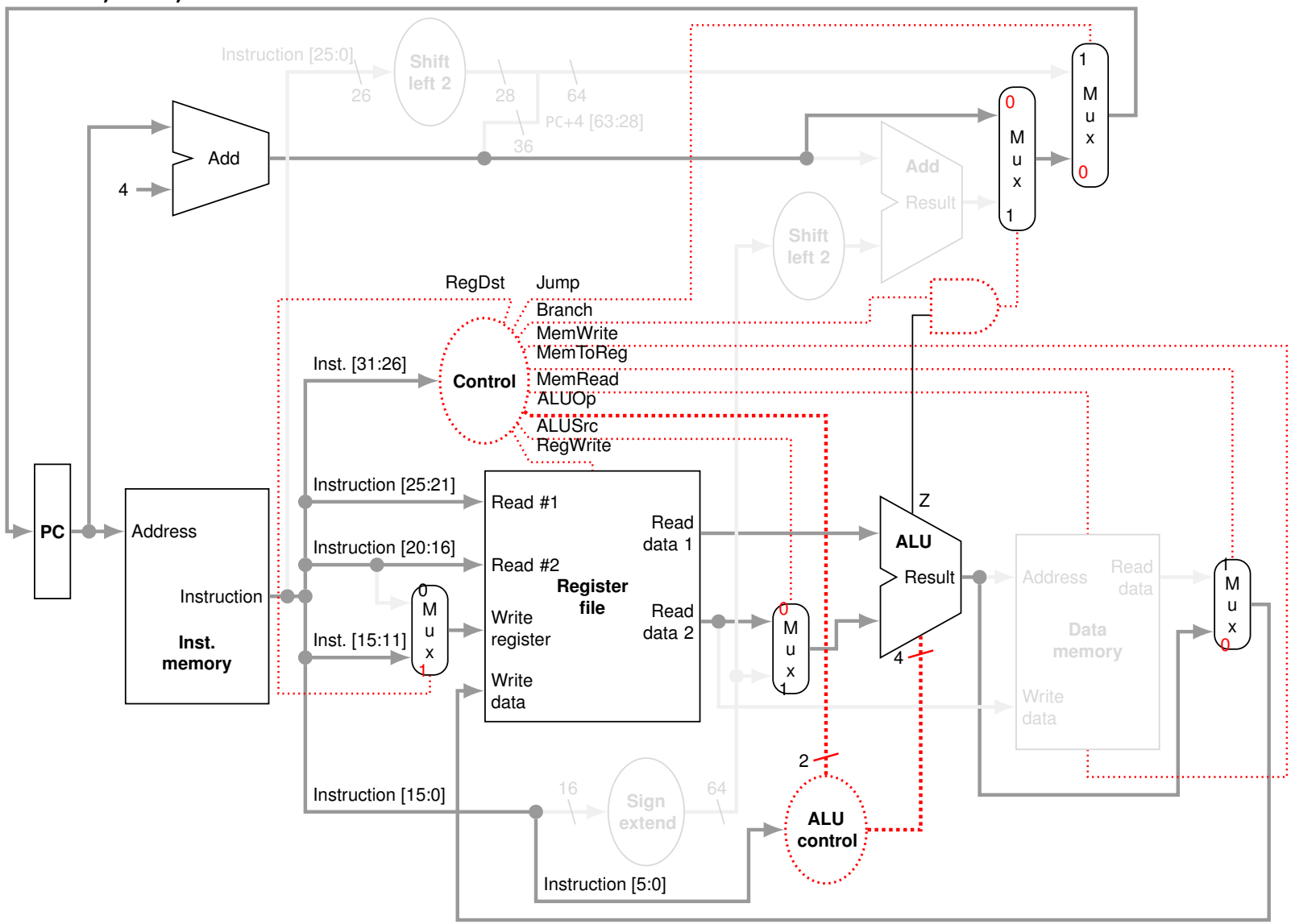


Camino de datos completo



Ejemplo

dadd r2, r3, r6



Monociclo

Inconvenientes

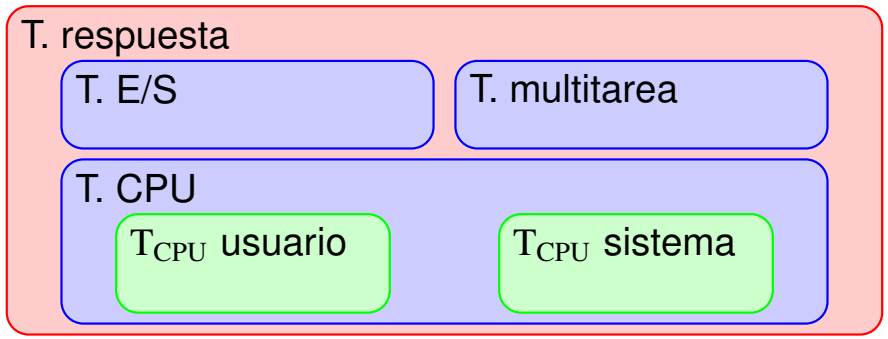
$$\mathrm{T_{CPU}} = \text{Instrucciones del programa} \times \mathrm{CPI} \times T$$

- Diseño sencillo
- CPI = 1
- Ciclo largo
- Imposible optimizar las instrucciones comunes

Mejoras de rendimiento

Reducción de tiempo de CPU

Reduce el tiempo de respuesta



Aceleración definida en la ley de Amdahl

- Tiempo de CPU

Mejoras de rendimiento

$$T_{CPU} = Nº \text{ instrucciones} \times CPI \times T$$

Reducción del tiempo de CPU

1. Número de instrucciones de los programas
   - Juego de instrucciones
   - Compilador
2. El periodo de reloj (incrementar frecuencia)
   - mejoras en las tecnologías de fabricación
   - mejoras organizativas (*pipeline*)
3. El CPI
   - número medio de ciclos por instrucción

Segmentación $\Rightarrow$ CPI=1 y $\downarrow T$

CPI

Microarquitectura	CPI	Duración del ciclo (T)
Monociclo	1	Ciclo largo
Multiciclo	>1	Ciclo corto
Segmentada	1 (ideal)	Ciclo corto

Segmentación de instrucciones (*pipeline*)

Símil

Tarea = Hacer la colada

1. Lavar
2. Secar
3. Planchar
4. Colocar

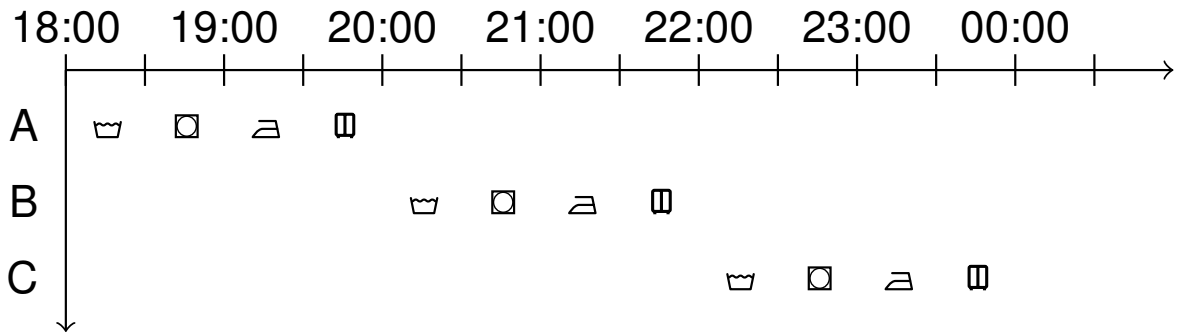
Clave

Dividir la ejecución de las instrucciones en etapas que funcionan en paralelo

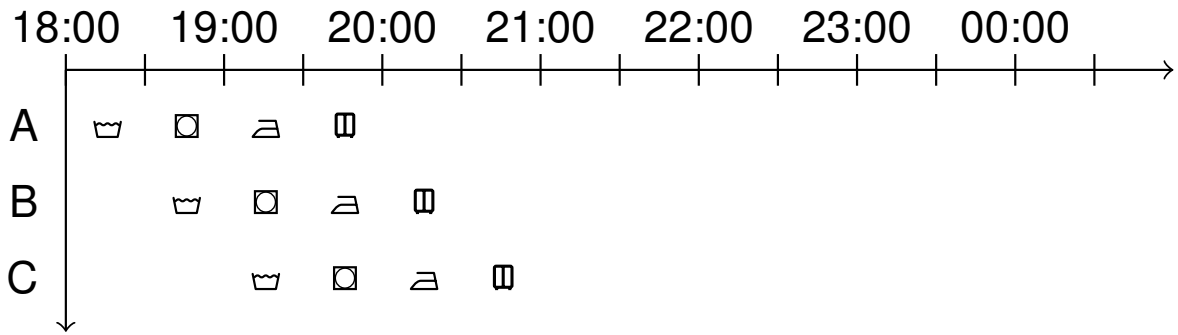
- transparente al software
- muy usada hoy en día

Símil

- No segmentada



- Segmentada



Pipeline

Crucial: Etapas en paralelo

- tiempos distintos
- etapas rápidas esperan por lentas
- balanceo de las etapas
- se mejora productividad, no el tiempo de respuesta

Caso ideal: colada 120 minutos

Etapas balanceadas: 30, 30, 30, 30 (minutos)

- tiempo de colada (t. respuesta): 120 minutos
- productividad: 1 colada cada 30 minutos
- aceleración = número de etapas

Caso real: colada 155 minutos

Tiempos: 40, 50, 45, 20 (minutos)

- ciclo del *pipeline*: 50 minutos
- tiempo de colada: 200 minutos (frente a 155)
- aceleración = 155/50 = 3.1

Pipeline MIPS

Etapas

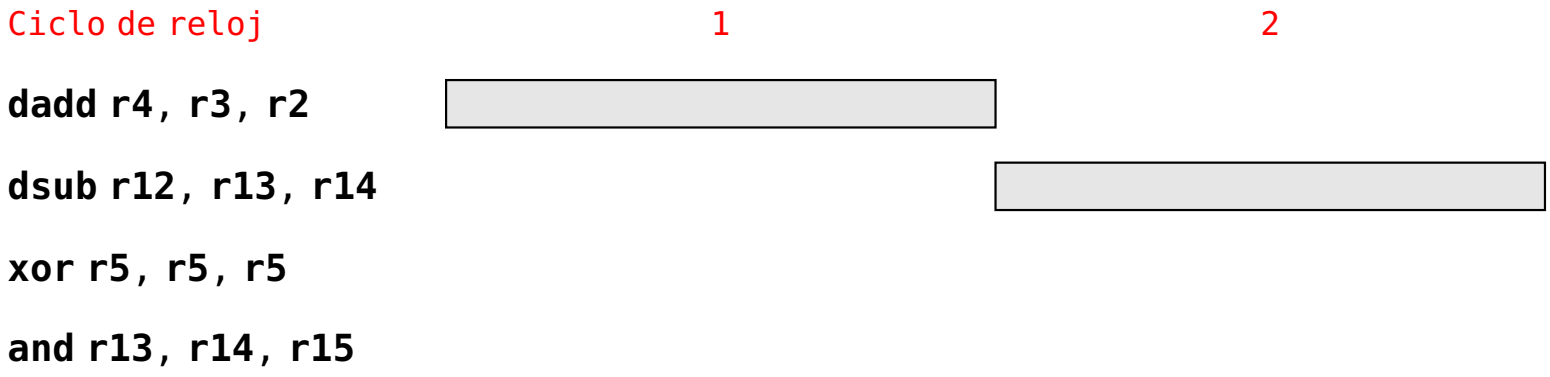
1. IF: carga código instrucción + incremento de PC
2. ID: decodificación de instrucción + lectura registros
3. EX: ejecución + cálculo de direcciones efectivas
4. MEM: acceso a memoria
5. WB: escritura diferida de registros

Número de etapas

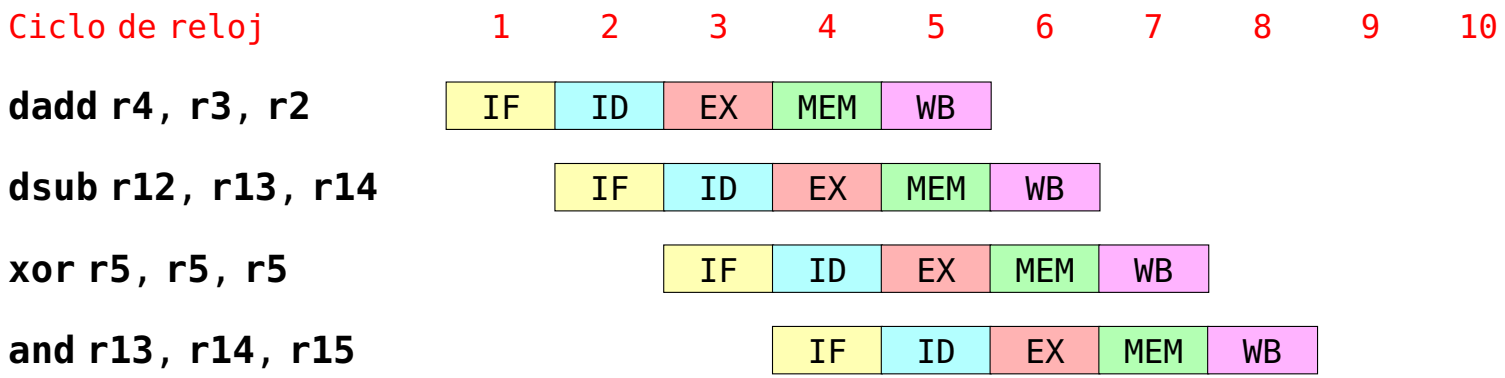
- duración de etapas = 1 ciclo
- $\uparrow$ número etapas $\Rightarrow \downarrow$ duración ciclo
- imposible incrementar indefinidamente (muro energía, balanceo, control)

Ejecución de instrucciones

Monociclo



Segmentada



La segmentación de instrucciones

¿Cuánto tarda en ejecutarse una instrucción?

Medida de tiempo de respuesta (latencia)

¿Cada cuánto la CPU completa una instrucción?

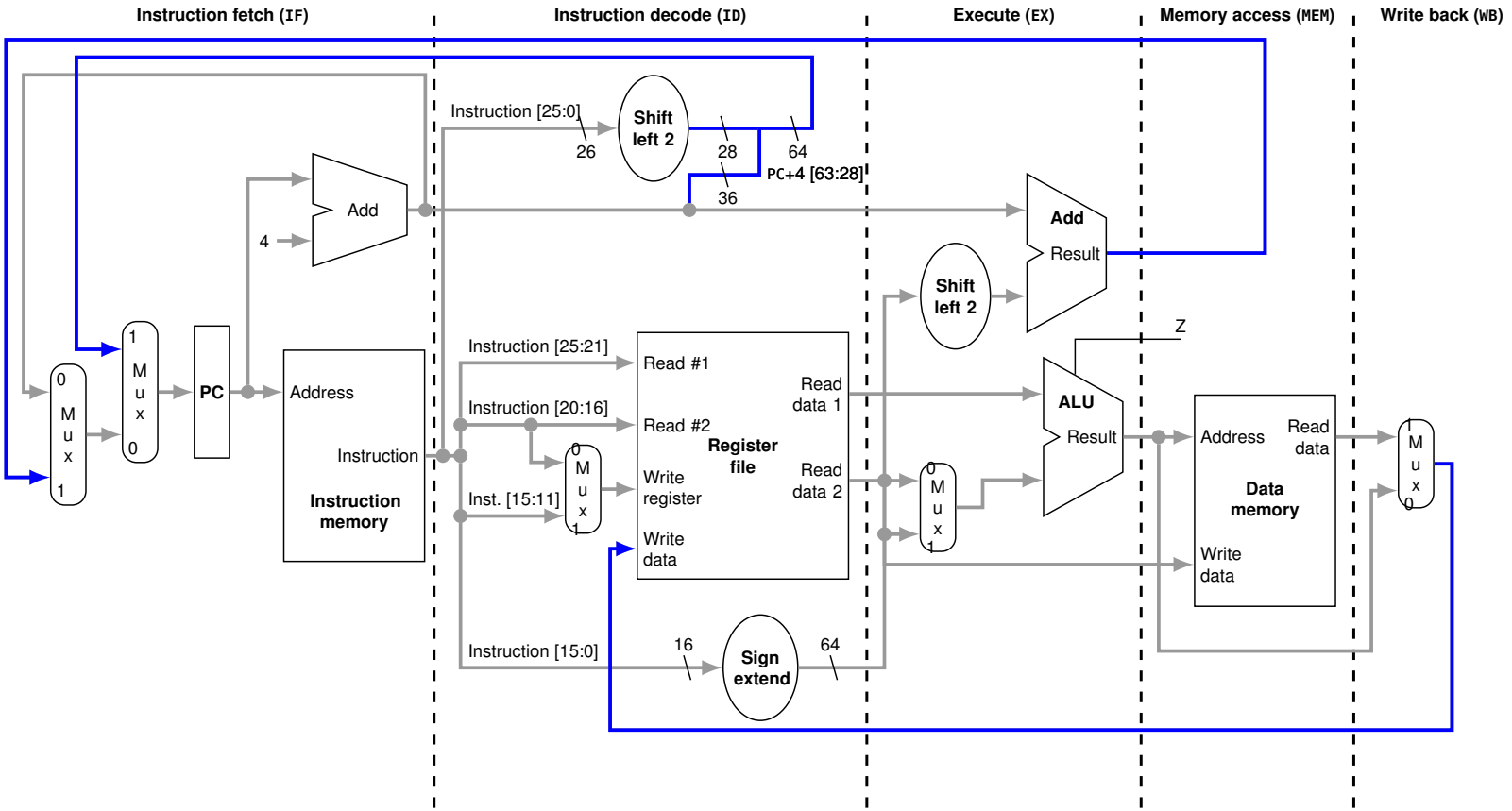
Instrucciones ejecutadas por unidad de tiempo (retiradas)
Medida de productividad

Caso ideal

Aceleración = número de etapas

- aumenta la complejidad de la unidad de control de la CPU
- riesgos de la segmentación $\Rightarrow \uparrow$ CPI medio
- muro de energía
- típicamente entre 10 y 20 etapas

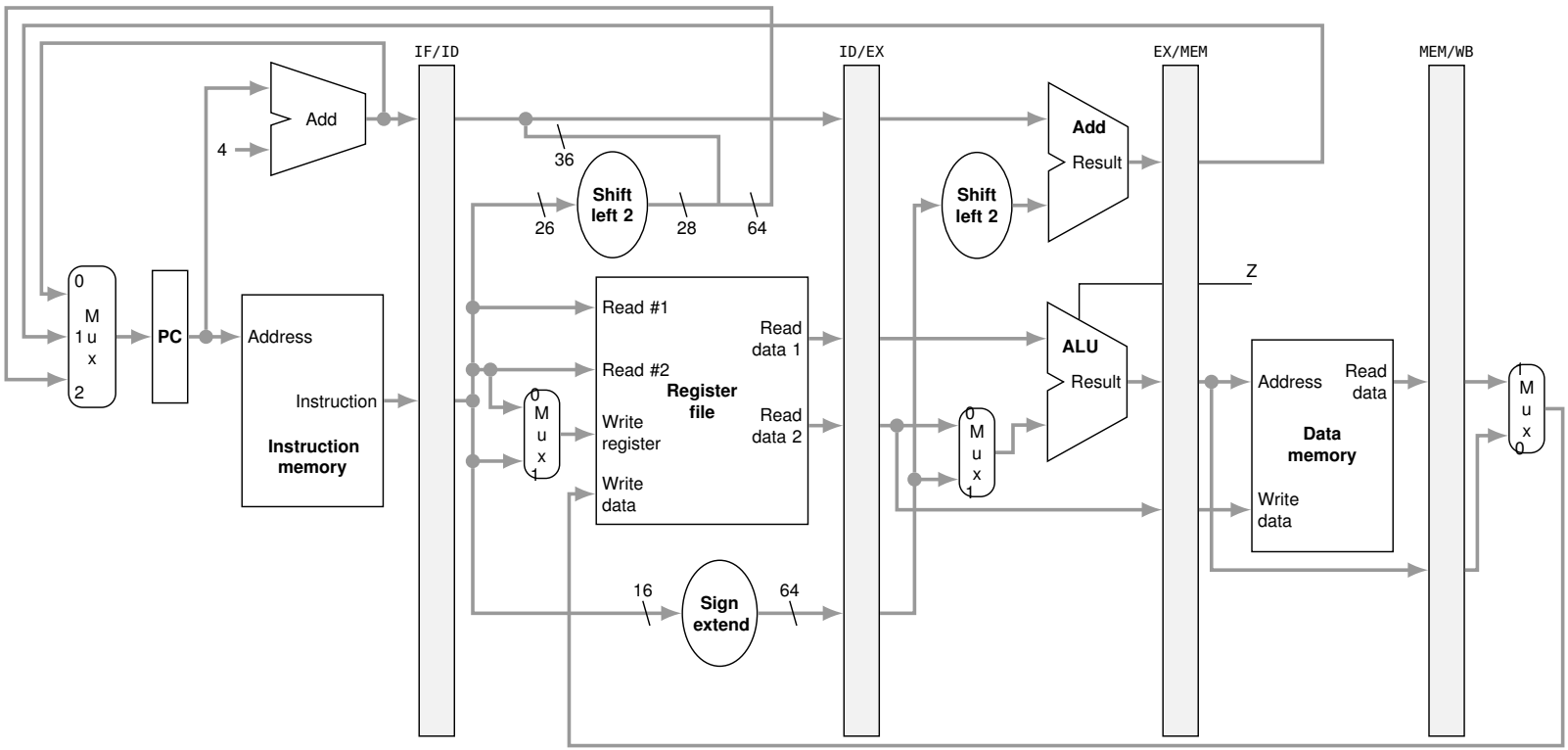
Camino de datos identificando etapas



Camino de datos segmentado

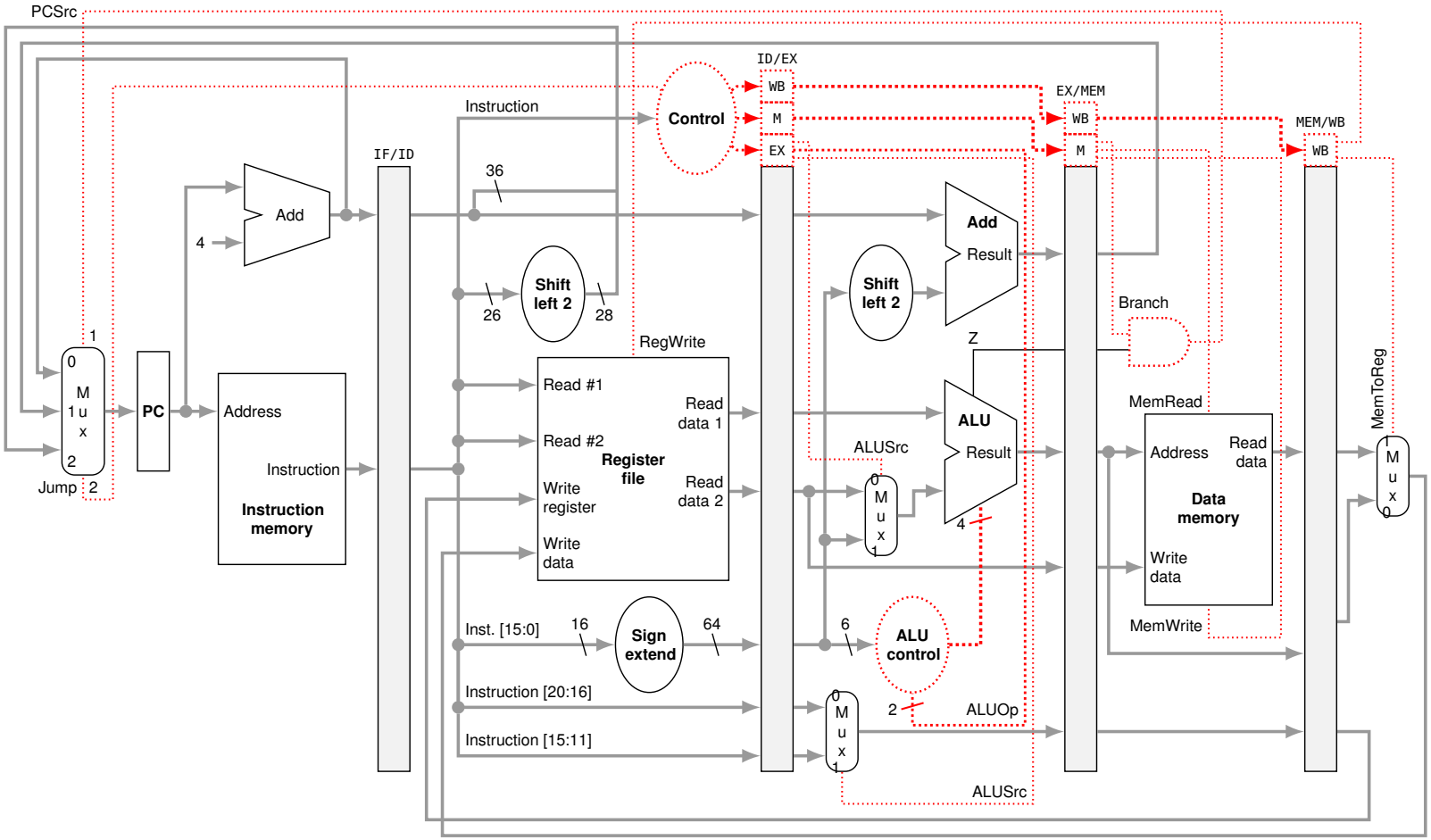
Registros de segmentación

- independizar etapas
- cauce avanza de registro de segmentación al siguiente



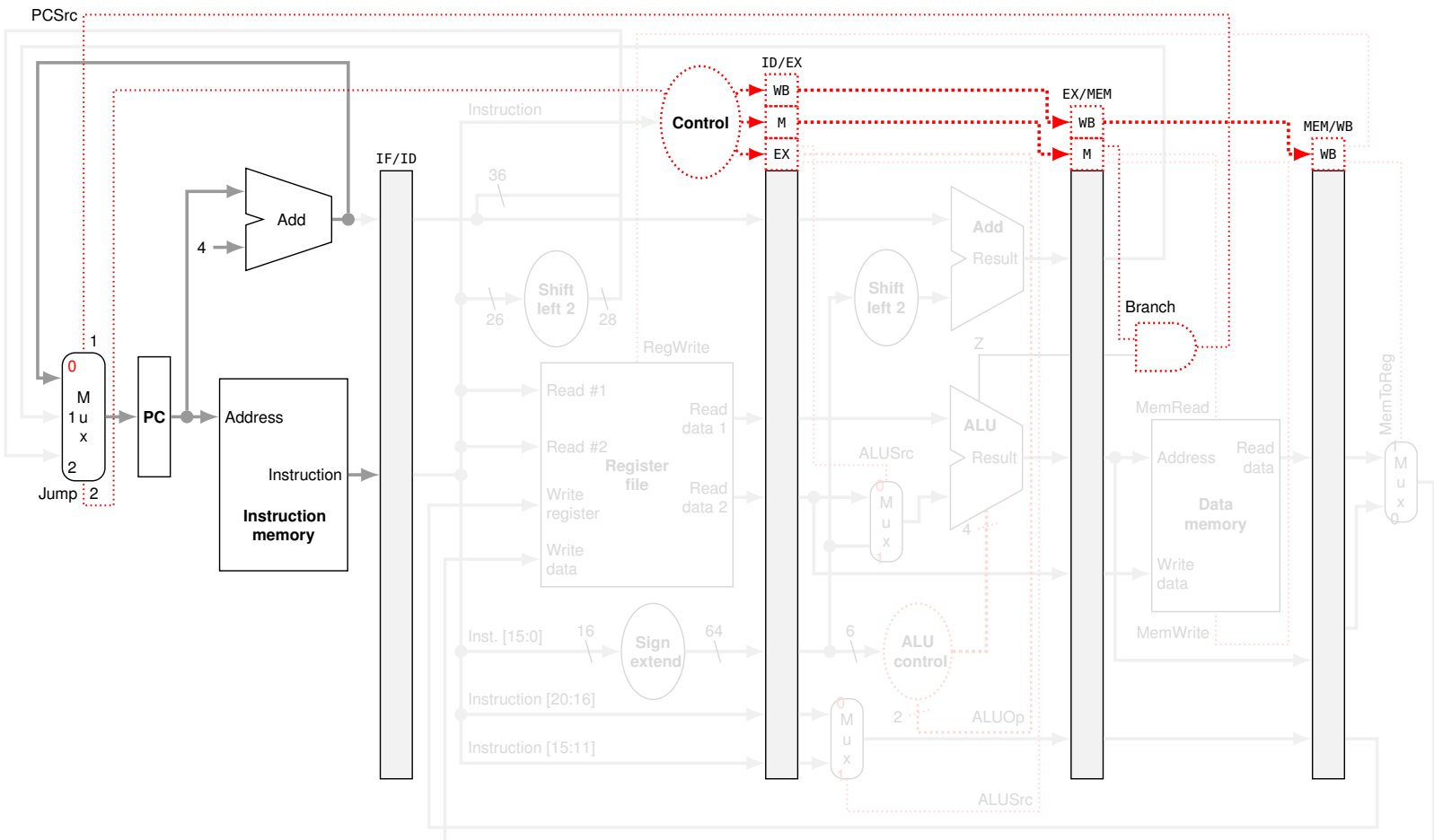
Camino segmentado con señales de control

Sin riesgos



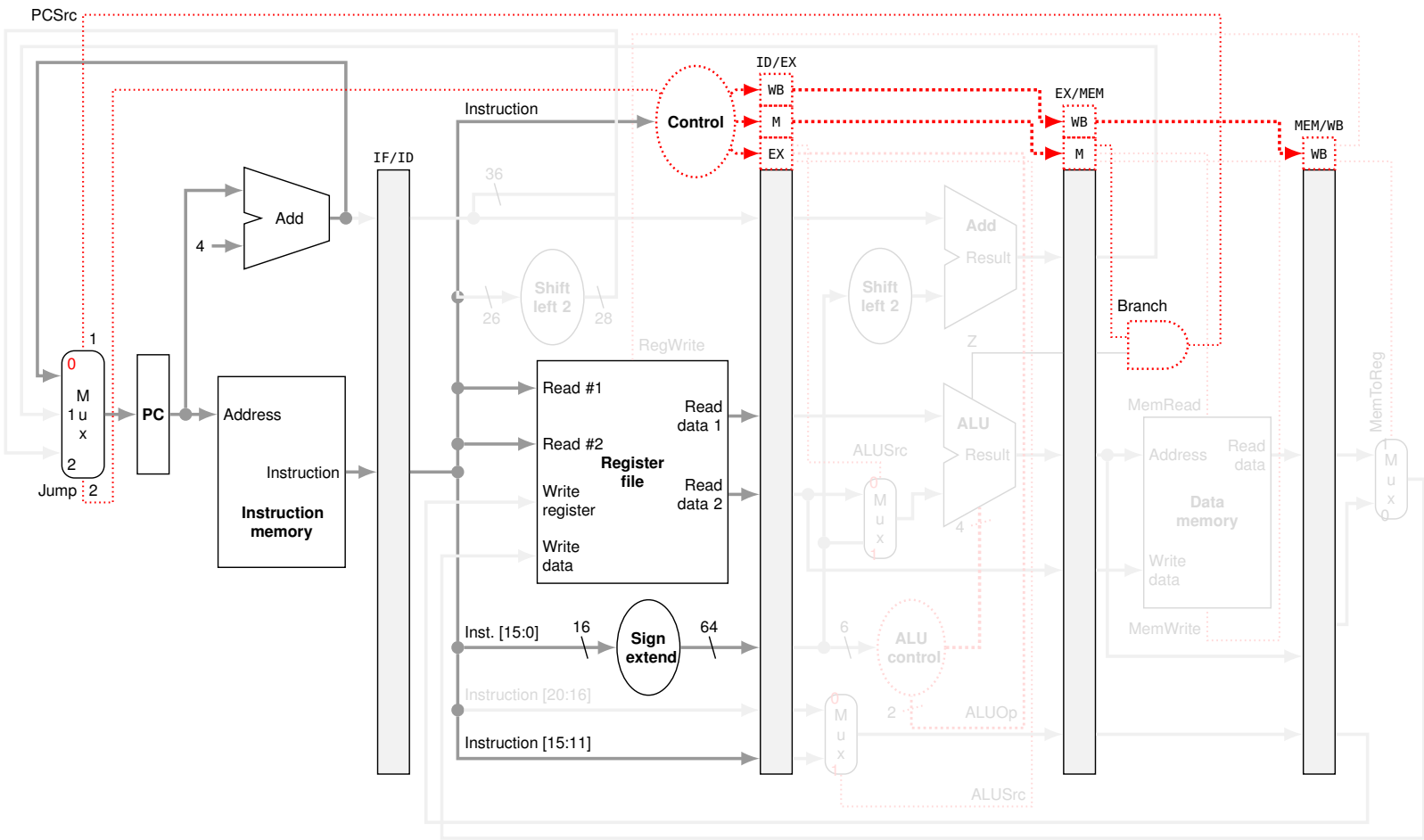
Ejemplo

dadd r3, r5, r8



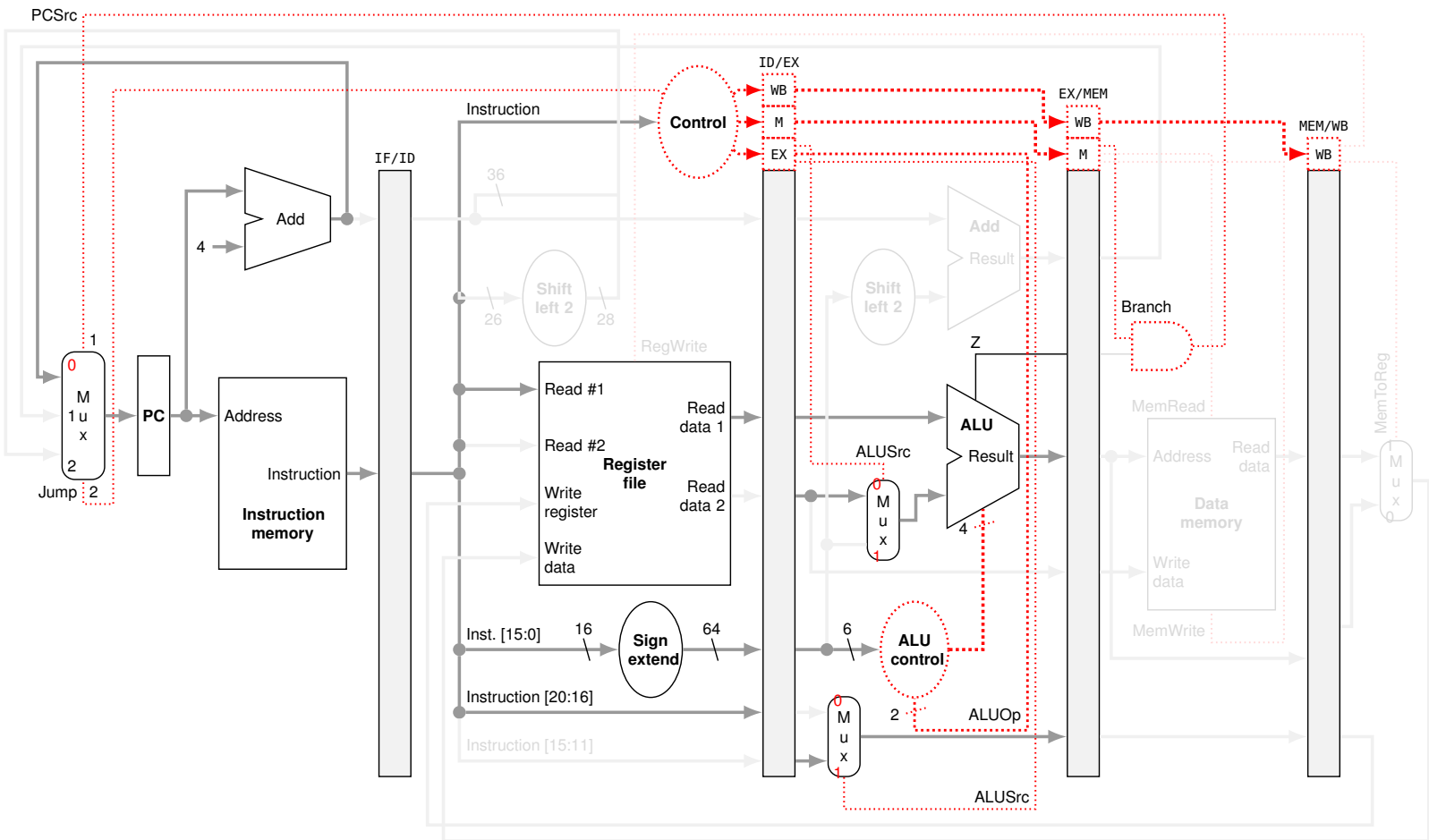
Ejemplo

`andi r1, r7,` 8 **`dadd r3, r5, r8`**



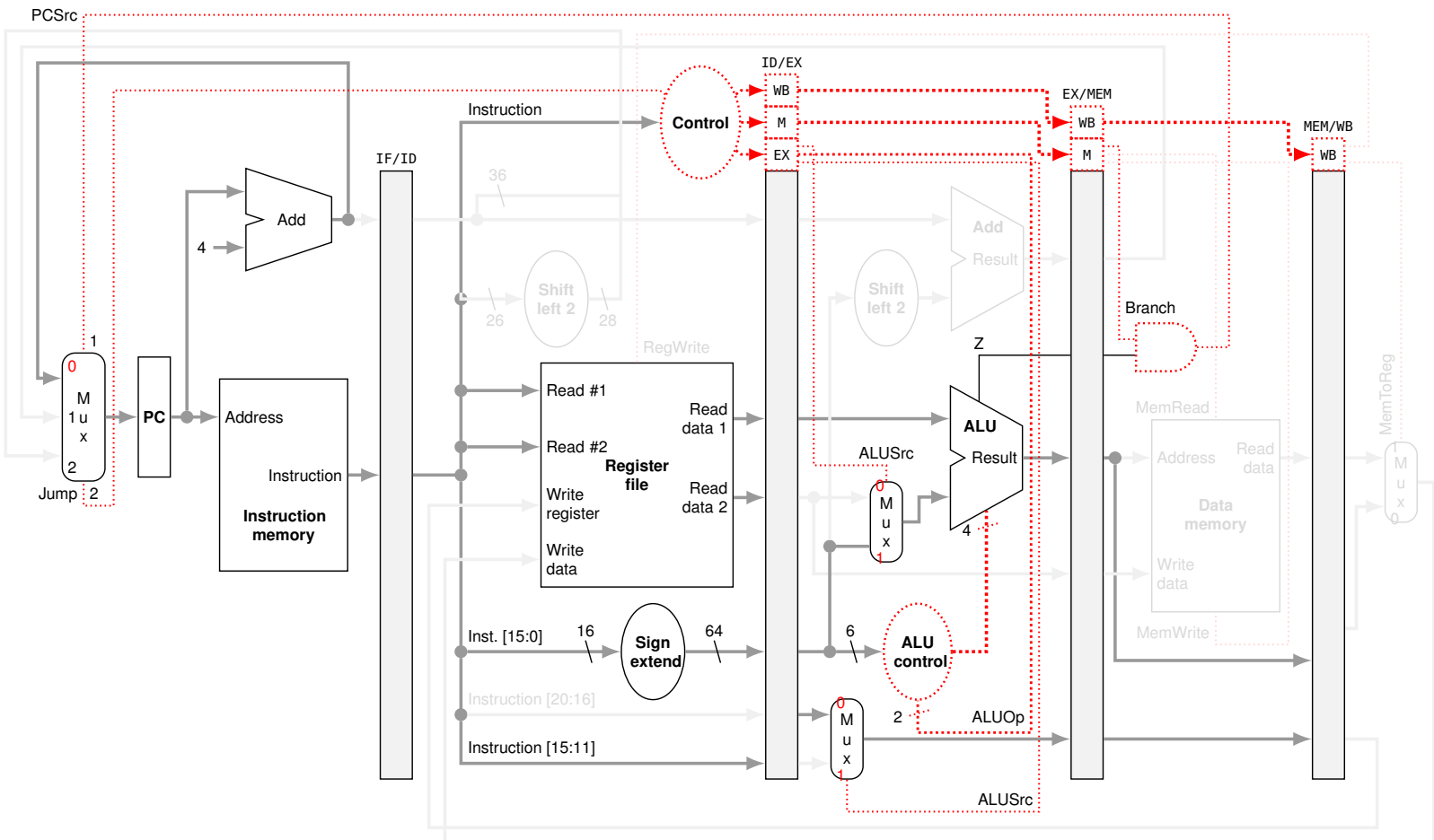
Ejemplo

```
or r2, r9, r17        andi r1, r7, 8        dadd r3, r5, r8
```



Ejemplo

xor r3, r5, r8 **or r2, r9, r17** **andi r1, r7,** 8 **dadd r3, r5, r8**



Riesgos de la segmentación

Riesgos

Fuerza la detención del cauce

- En la práctica: introducir **nop**

Tipos

- Estructurales
- Dependencia de datos
- Control (dependencia sobre PC)

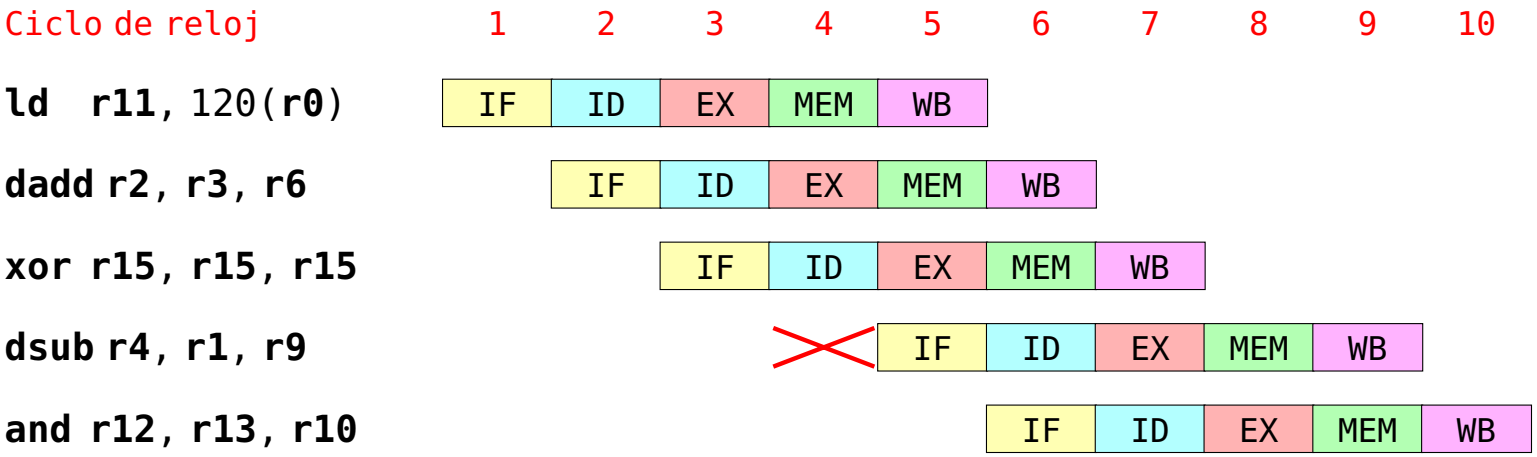
Riesgos estructurales

Definición

Dos etapas requieren el mismo hardware

- Imposibles en la microarquitectura MIPS64

Ejemplo asumiendo memoria única



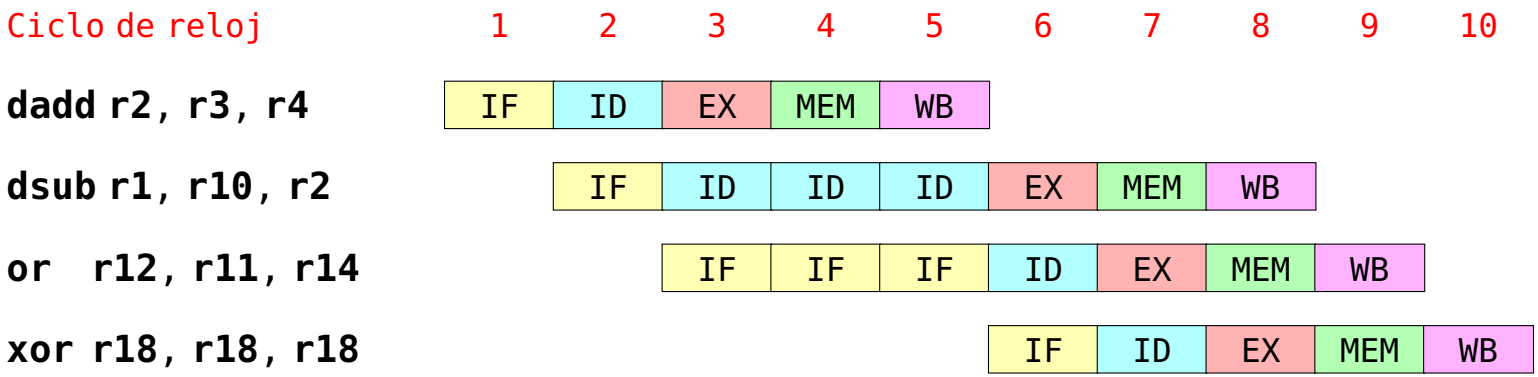
Riesgos por dependencia de datos

Dependencia de datos

- RAW: Read After Write
- WAR: Write After Read
- WAW: Write After Write

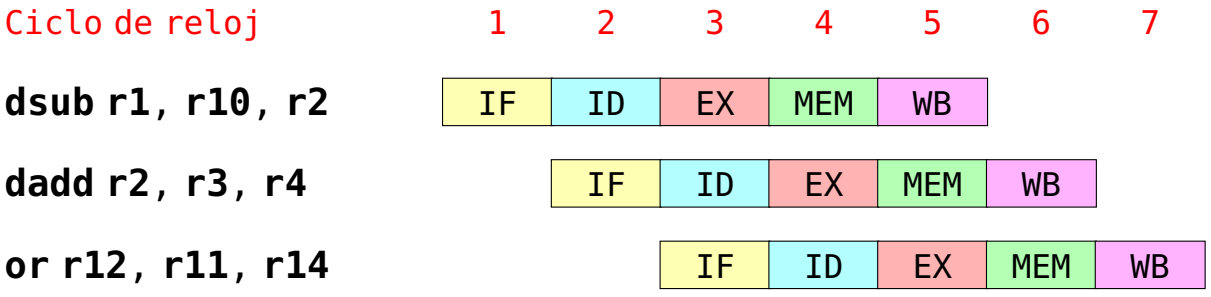
Entre instrucciones del cauce segmentado $\Rightarrow$ riesgos por dependencias de datos

Ejemplo RAW

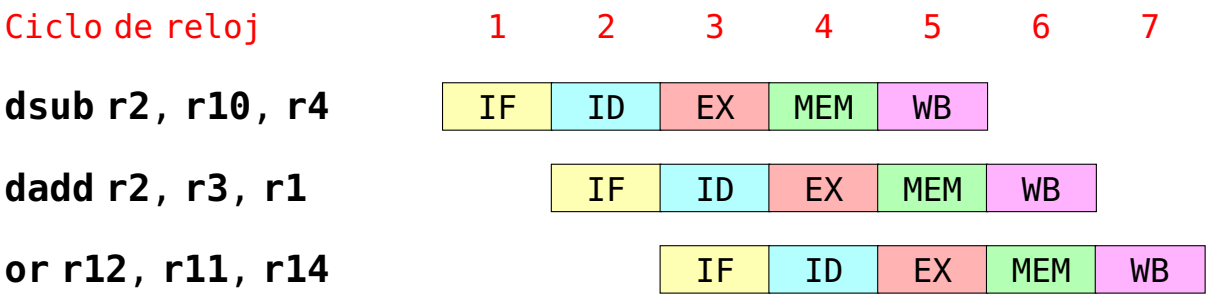


Riesgos por dependencia de datos

Ejemplo WAR



Ejemplo WAW



Riesgos de control

Definición

Cambio en el flujo del programa

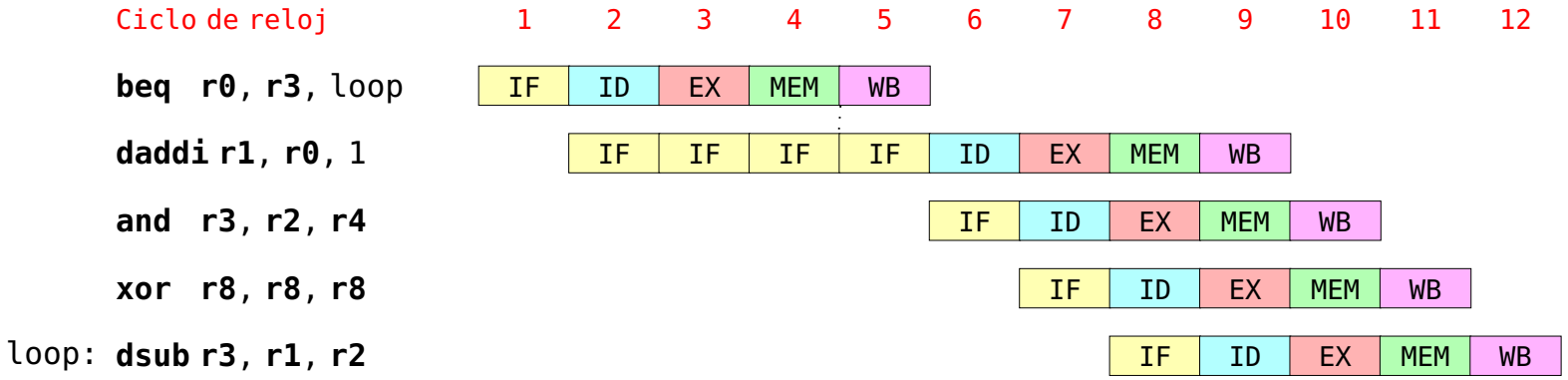
- Vaciar el *pipeline*
- Costoso
- Dependencia de datos sobre `PC`

Ejemplo

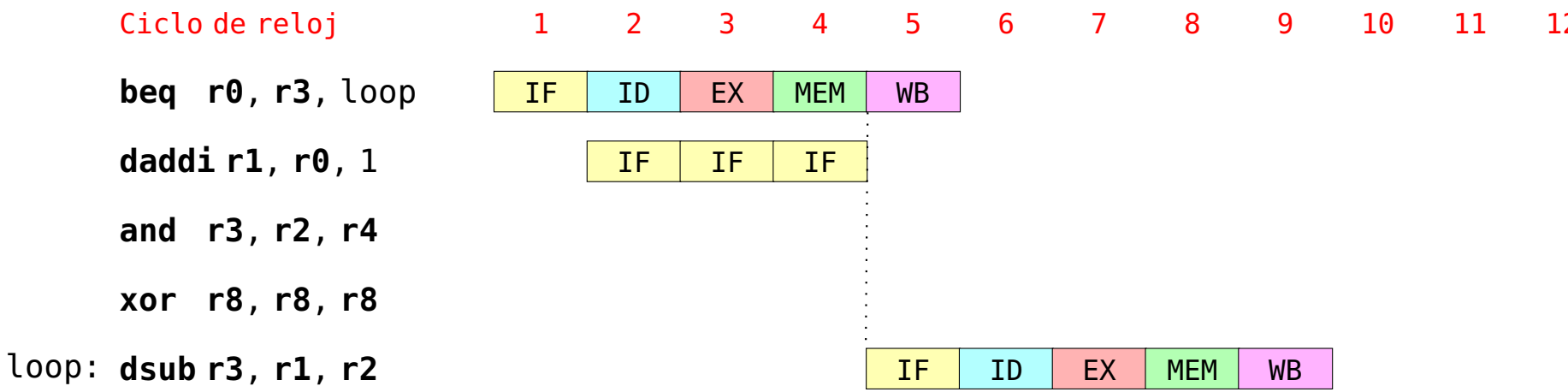
```
      beq r0, r3, loop
      daddi r1, r0, 1
      and r3, r2, r4
      xor r8, r8, r8
loop: dsub r3, r1, r2
```

Riesgos de control

No tomado

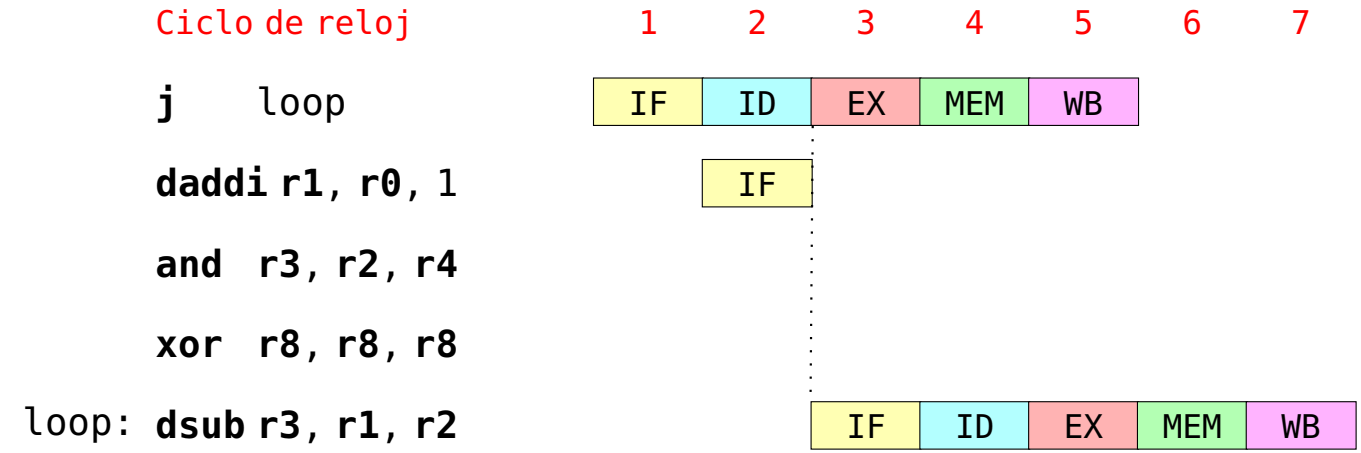


Tomado



Riesgos de control

Salto incondicional

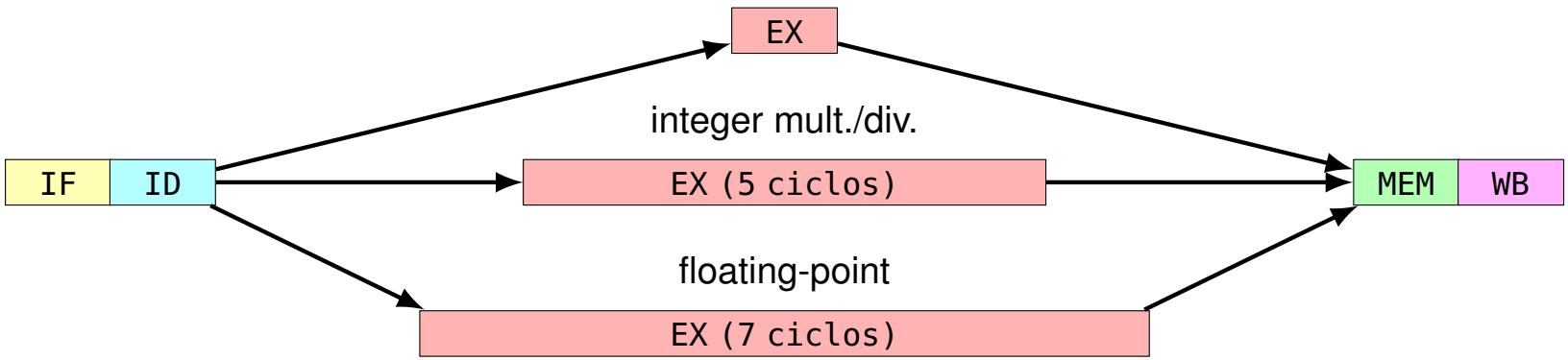


Operaciones multiciclo

Operaciones complejas

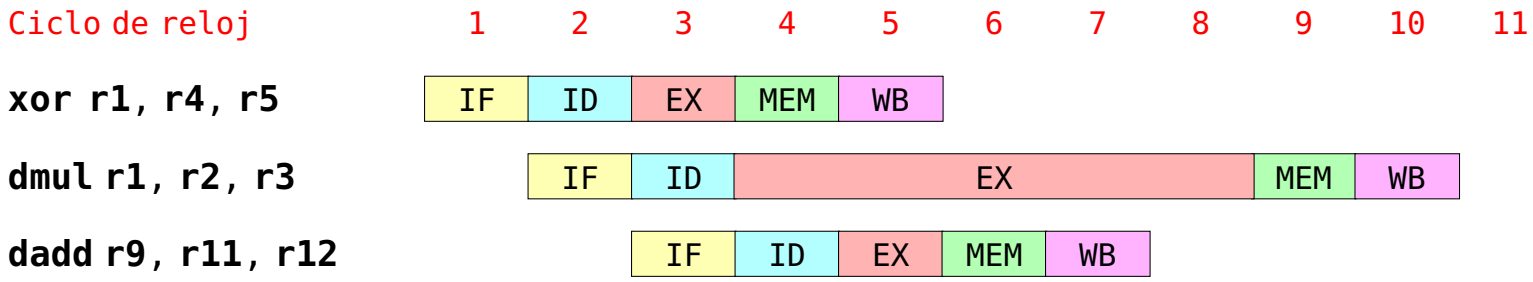
Etapa EX compleja

- Multiplicaciones, divisiones, punto flotante ...
- ✘ Alargar ciclo de reloj
- ✔ Solución: varios ciclos en EX



Operaciones multiciclo

Optimización: emisión múltiple

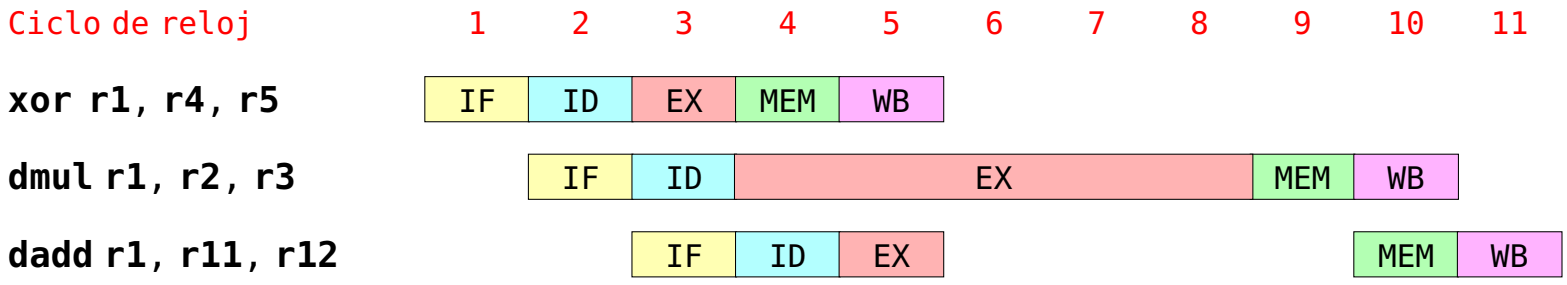


Estados de instrucciones (en orden / fuera de orden)

- Emitida (Preparada para `EX`)
- Ejecutada (Ha terminado `EX`)
- Retirada (Ha terminado `MEM` + `WB`)

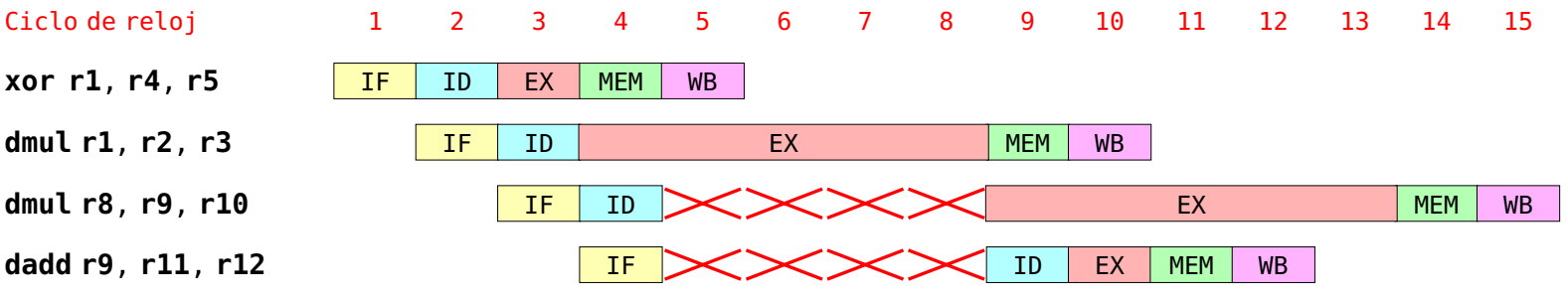
Operaciones multiciclo

Problema: dependencias WAW y WAR



Problema

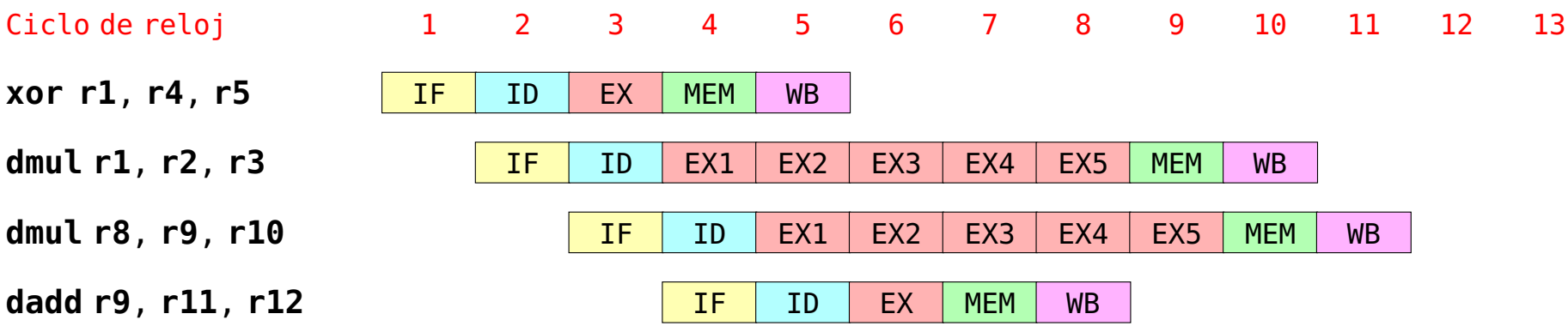
Aparecen riesgos estructurales



Operaciones multiciclo

Mejora

Segmentar unidades EX complejas



Excepciones

Cambios en el flujo del programa

Errores / interrupciones

- División por cero, código de instrucción inválido, etc.
- El S.O. toma el control del sistema
- Guarda el contexto
- Restaura y continua transparentemente
- Asíncrono (Se producen en cualquier momento)

Excepciones precisas

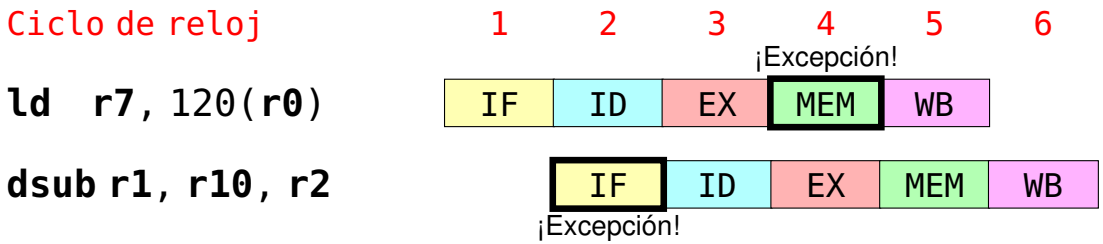
Hardware se encarga

- Lidiar con instrucciones a medio ejecutar
- Comportamiento esperado como no segmentado

Excepciones

Ocurrencia

En cualquier momento y varias al mismo tiempo

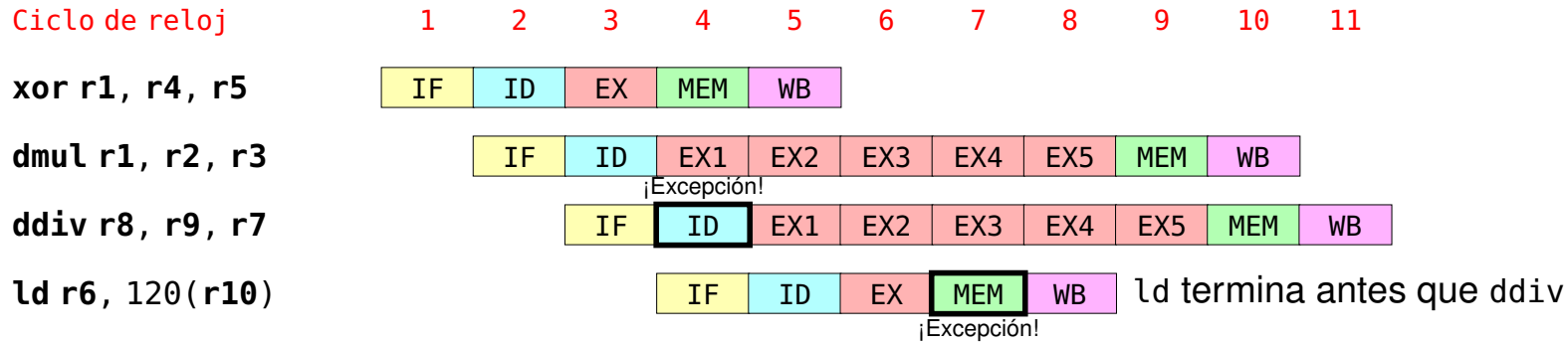


Procesar en orden

- Todas en la etapa `WB`

Excepciones

Problema: ejecución fuera de orden

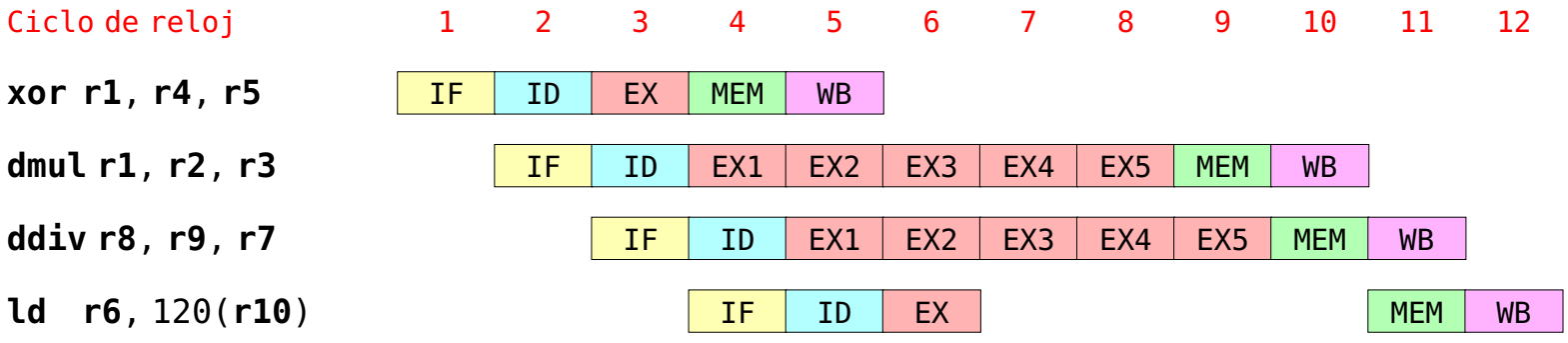


Solución

- Ejecución fuera de orden
- Deben terminar en orden (`MEM` y `WB`)

Excepciones

Ejemplo



Mejoras en la segmentación

Rendimiento CPU segmentada

$$\mathrm{T_{CPU}} = \text{Instrucciones del programa} \times \text{CPI} \times T$$

Idealmente CPI = 1

- Detenciones por riesgos aumentan CPI
- MIPS64: solo RAW y control

Posibles mejoras

- $\downarrow T \Rightarrow \uparrow$ Profundidad de segmentación
- $\downarrow$ CPI $\Rightarrow \downarrow$ Detenciones
  - Reducir riesgos de datos
  - Reducir riesgos de control

Reducción detenciones datos

Dependencias

- RAW: dependencia verdadera
- WAR: antidependencia
- WAW: dependencia de salida

Estrategias

- Planificación de instrucciones
- Rutas de reenvío

} Reduce detenciones

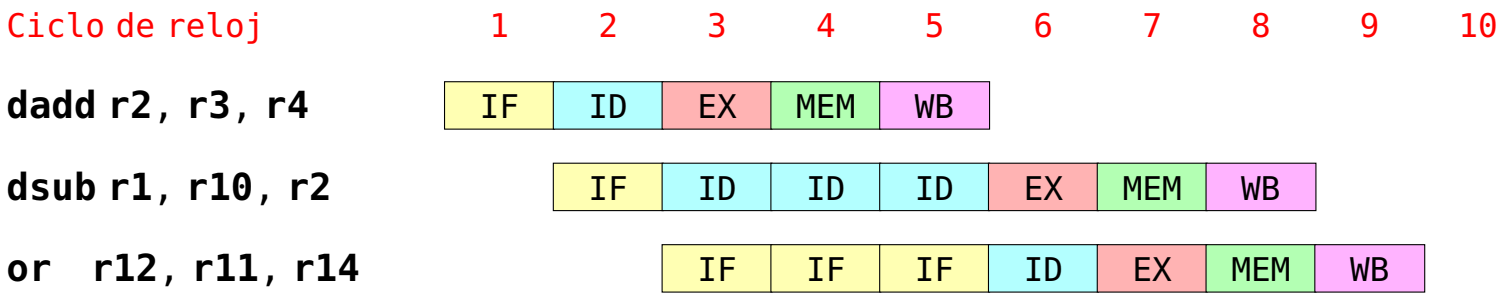
- Renombrado de registros $\rightarrow$ Elimina dependencias

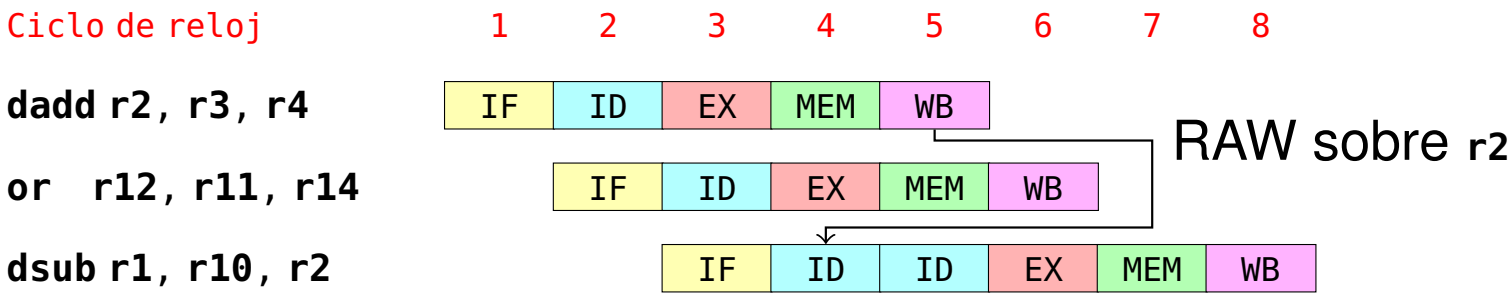
Planificación de instrucciones

HW y SW

Reordenar instrucciones

- Reducir detenciones
- Sin cambiar semántica
- Software: recompilar para cada microarquitectura





Rutas de reenvío

Forwarding

- Reducir/eliminar detenciones por dependencias de datos
- Adelantar uso de registros calculados en el cauce
- No esperar a `WB`

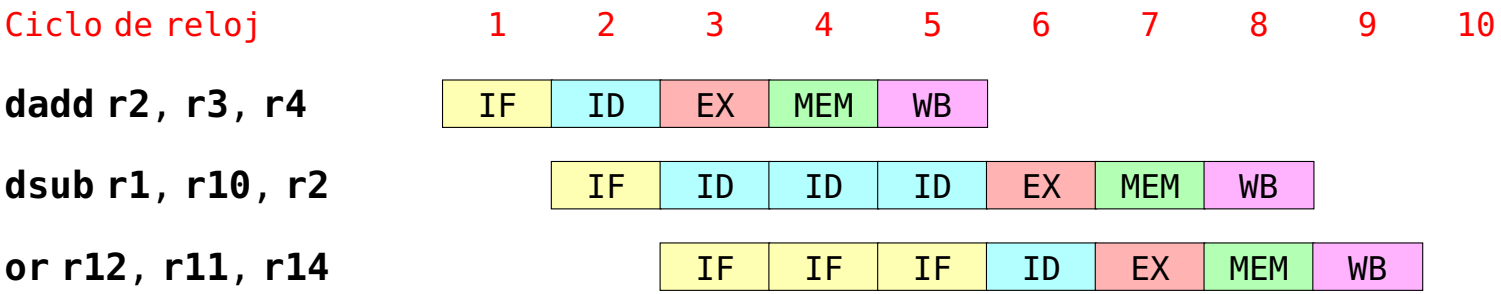
Situaciones

- escritura sin carga y lectura consecutivas
- escritura sin carga y lectura separadas 1 instrucción
- escritura de carga y lectura consecutivas
- escritura de carga y lectura separadas 1 instrucción

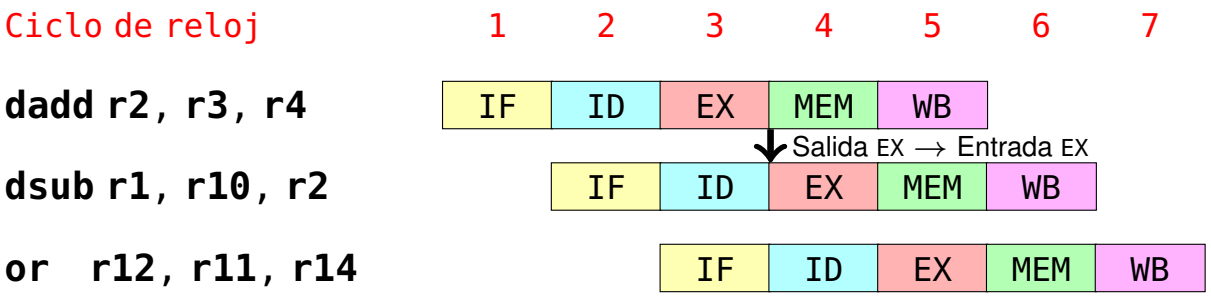
Rutas de reenvío

Escritura y lectura consecutivas

Sin *forwarding*



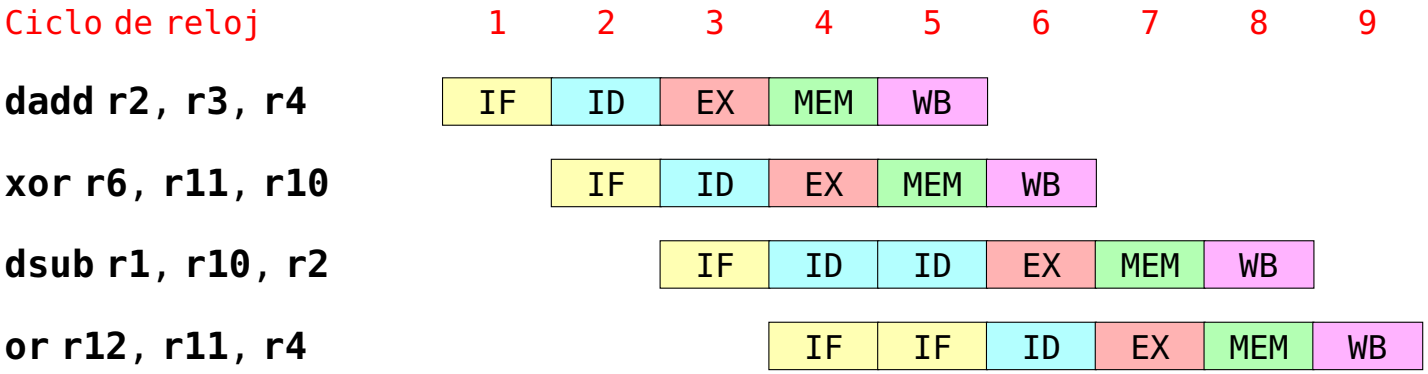
Con *forwarding*



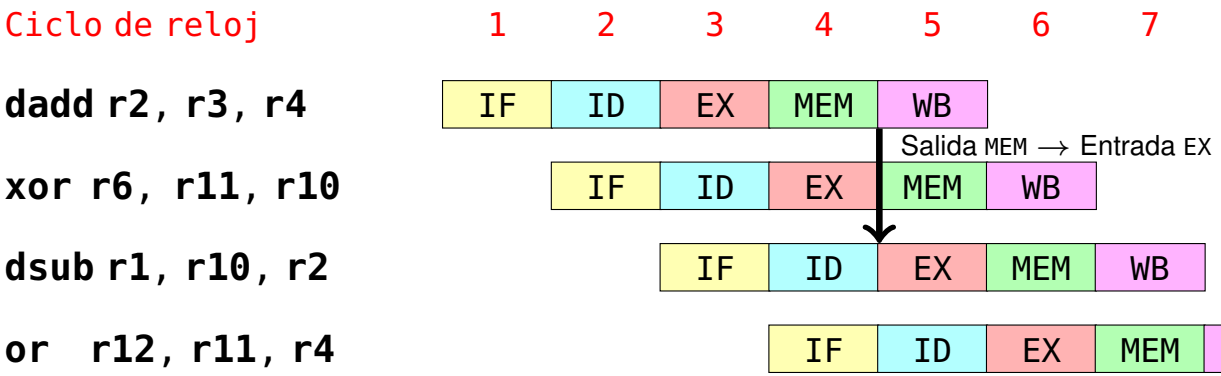
Rutas de reenvío

Escritura y lectura separadas 1 instrucción

Sin *forwarding*



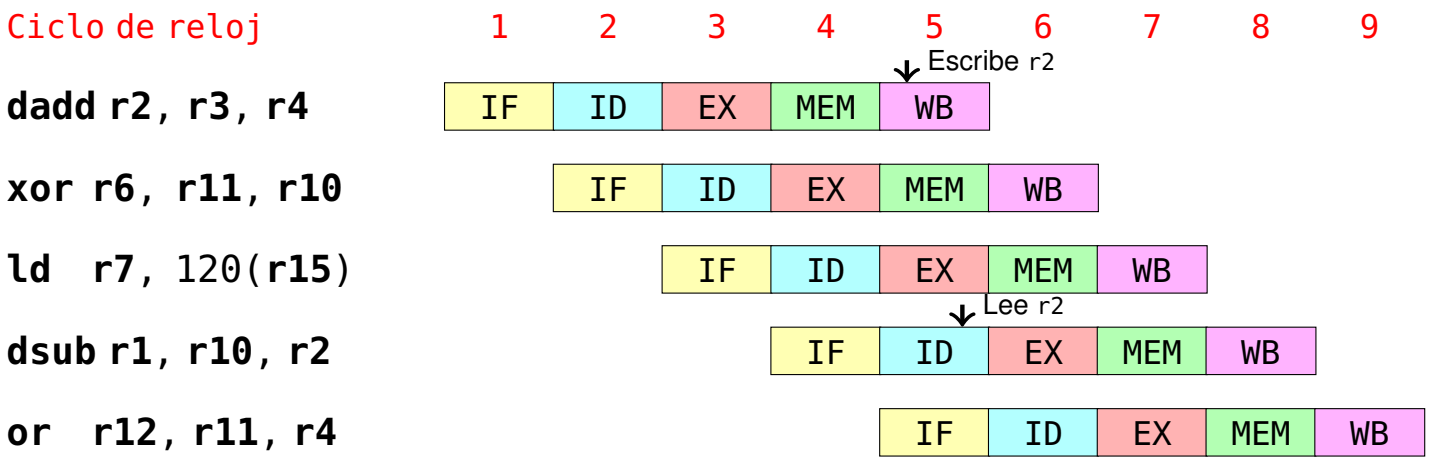
Con *forwarding*



Rutas de reenvío

Escritura y lectura separadas 2 instrucciones

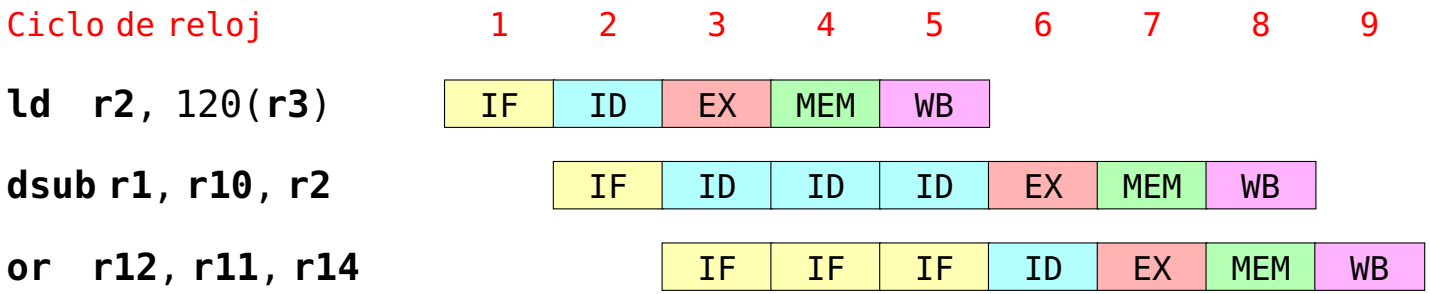
No necesario *forwarding*



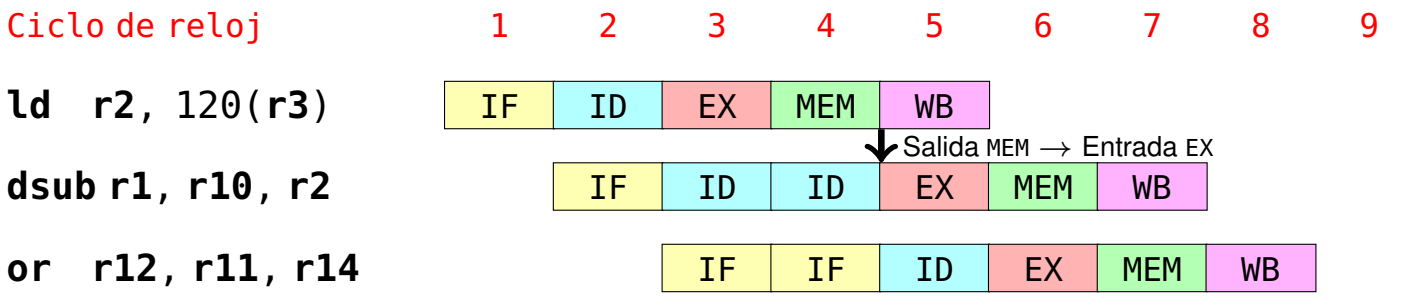
Rutas de reenvío

Carga y lectura consecutivas

Sin *forwarding*



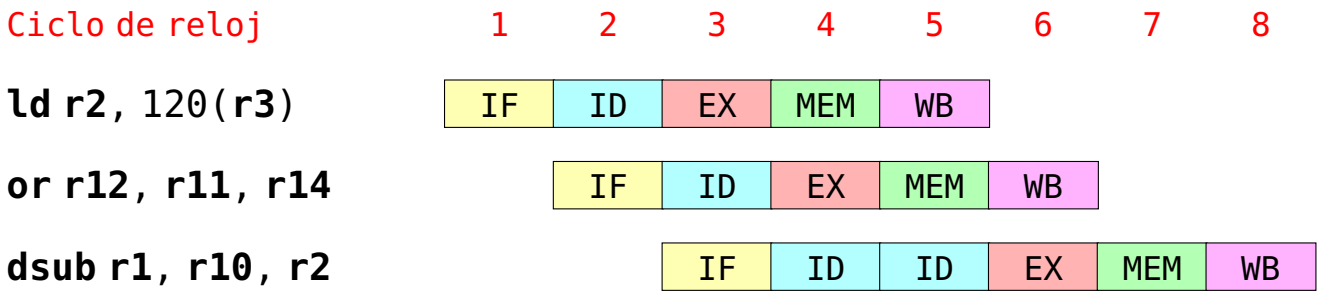
Con *forwarding*



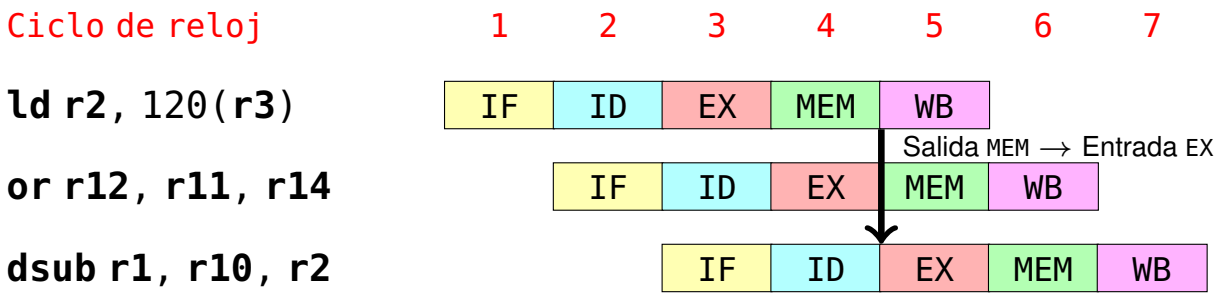
Rutas de reenvío

Carga y lectura separadas 1 instrucción

Sin *forwarding*



Con *forwarding*



Reciclado de registros

Dependencias RAW

- Semántica del programa
- Dependencias verdaderas

Dependencias WAR y WAW

- Debidas a reciclado de registros
- Número limitado de registros
- Impacto en ejecución fuera de orden
- Evitables

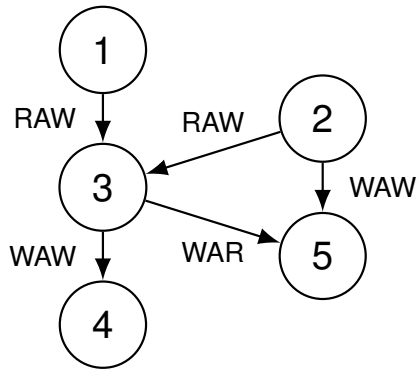
Reciclado de registros

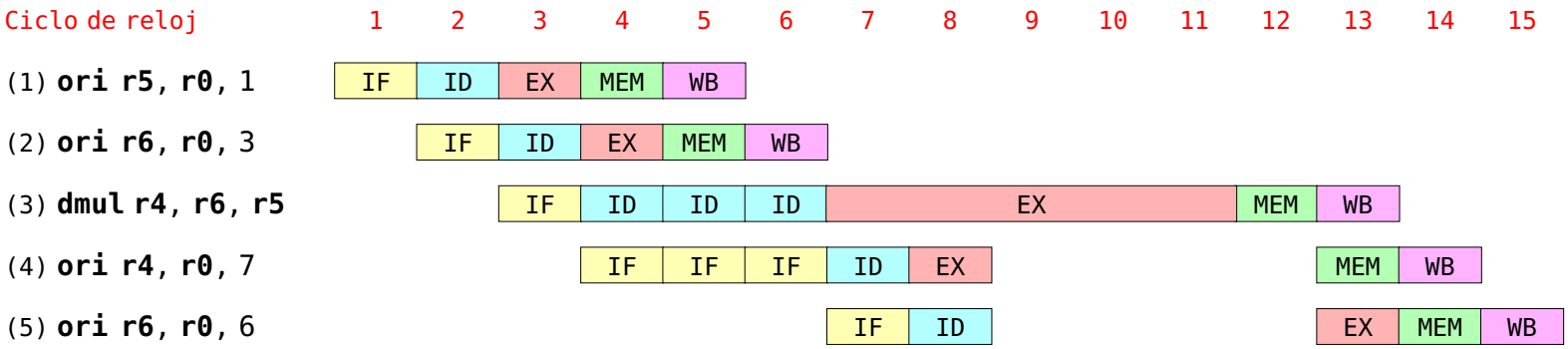
Ejemplo

Con reciclado

```
a = 1;
b = 3 * a;
c = 7;
d = 6;
```

```
(1) ori  r5, r0, 1  ; r5 <- a
(2) ori  r6, r0, 3  ; r6 <- 3
(3) dmul r4, r6, r5 ; r4 <- b
(4) ori  r4, r0, 7  ; r4 <- c
(5) ori  r6, r0, 6  ; r6 <- d
```





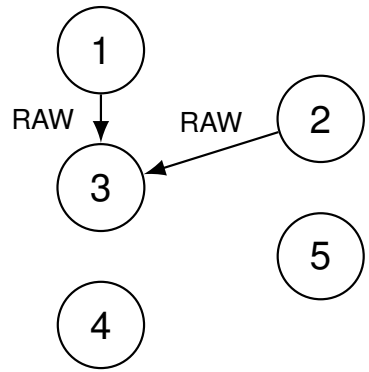
Reciclado de registros

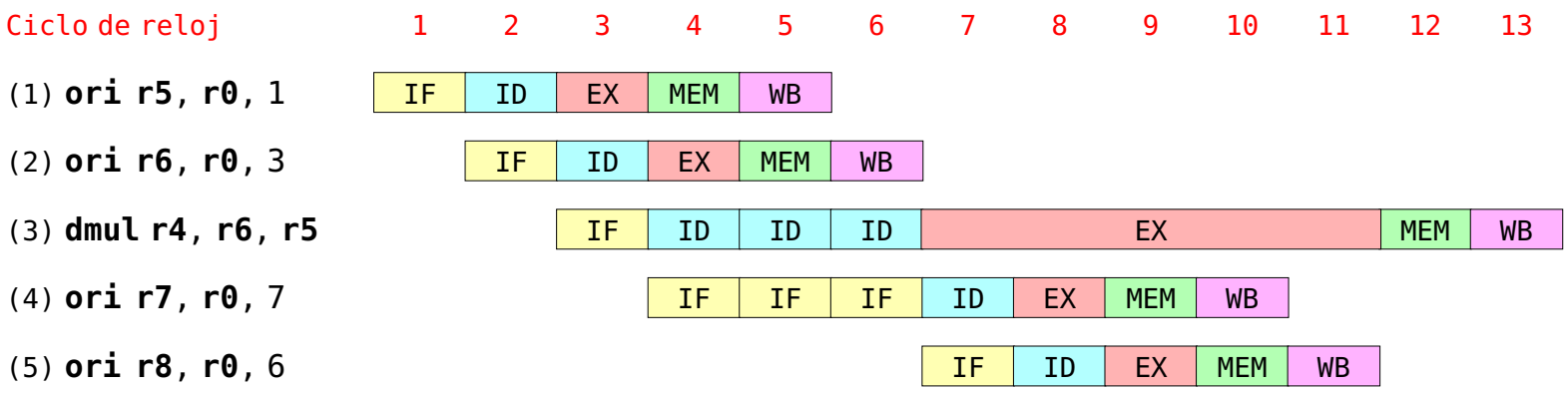
Ejemplo

Sin reciclado

```
a = 1;
b = 3 * a;
c = 7;
d = 6;
```

```
(1) ori  r5, r0, 1  ; r5 <- a
(2) ori  r6, r0, 3  ; r6 <- 3
(3) dmul r4, r6, r5 ; r4 <- b
(4) ori  r7, r0, 7  ; r7 <- c
(5) ori  r8, r0, 6  ; r8 <- d
```





Renombrado de registros

Ideal

- Disponer de muchos registros
- ✘ Problema: muchos bits en el código de instrucción

Solución

- Separar registros arquitectónicos de físicos
  - Arquitectónicos $\Rightarrow$ simbólicos
  - Gran número de registros físicos
  - Tabla de renombrado: mapeo inicial por defecto

Renombrado de registros

Tabla renombrado

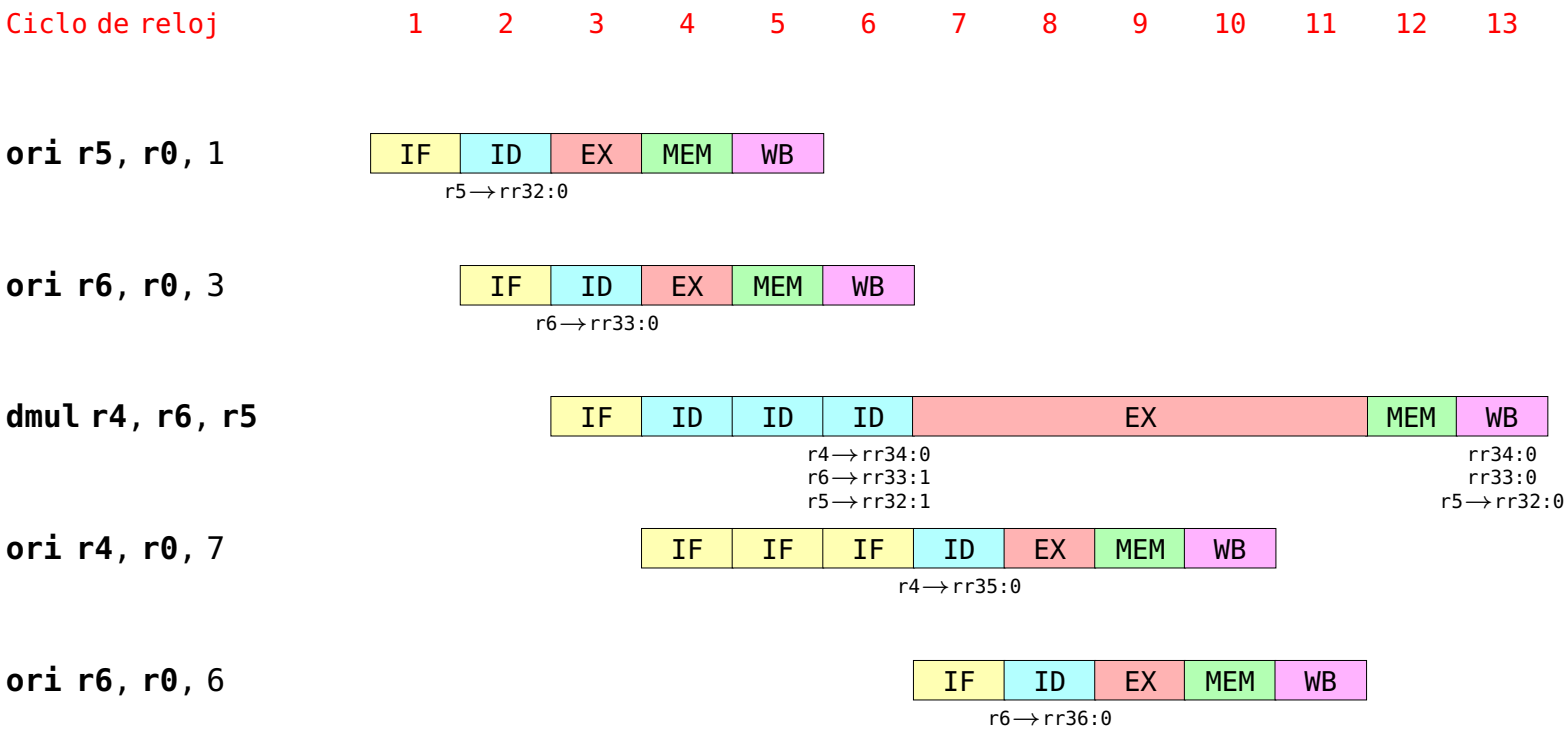
- Asociación registro físico - arquitectónico
- Estado registro físico
  - Asociado y en uso
  - Asociado y en desuso
  - Disponible

Funcionamiento

1. Arranque: mapeo por defecto y en desuso (**r6**→rr6)
2. Etapa ID: identificar registros arquitectónicos
   - Renombrar registro destino
   - Incrementar contador de registros origen
   - **r0** no se renombra
3. Etapa WB: decrementar contador uso
   - Liberar registros físicos

Renombrado de registros

Ejemplo



Reducción detenciones control

Se producen

- Saltos condicionales (`MEM`) e incondicionales (`ID`)
- Excepciones (`WB`)

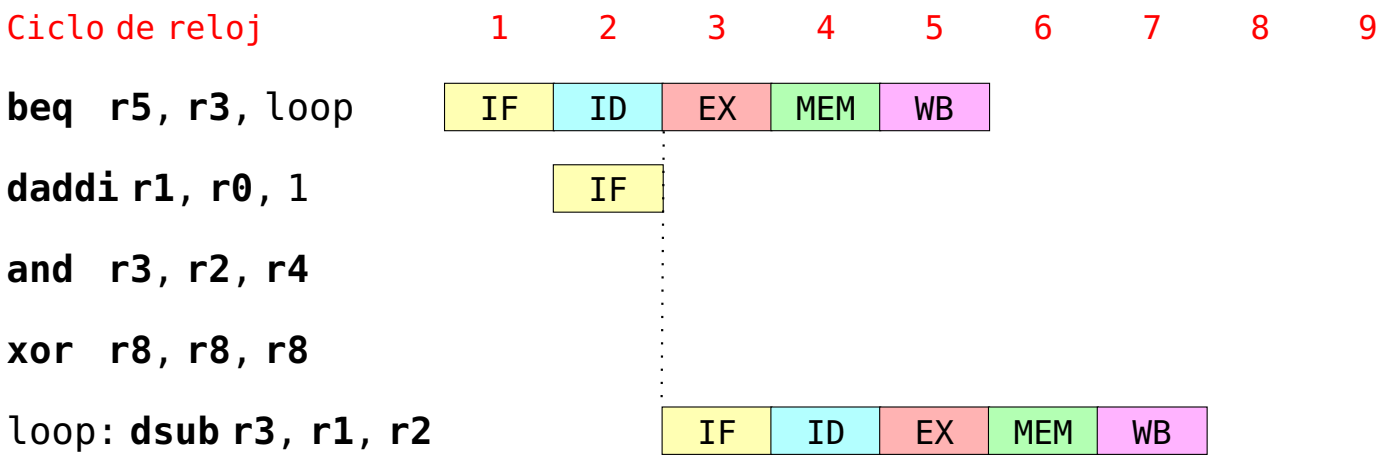
Estrategias

- Evaluación agresiva de saltos
- Predicción de saltos

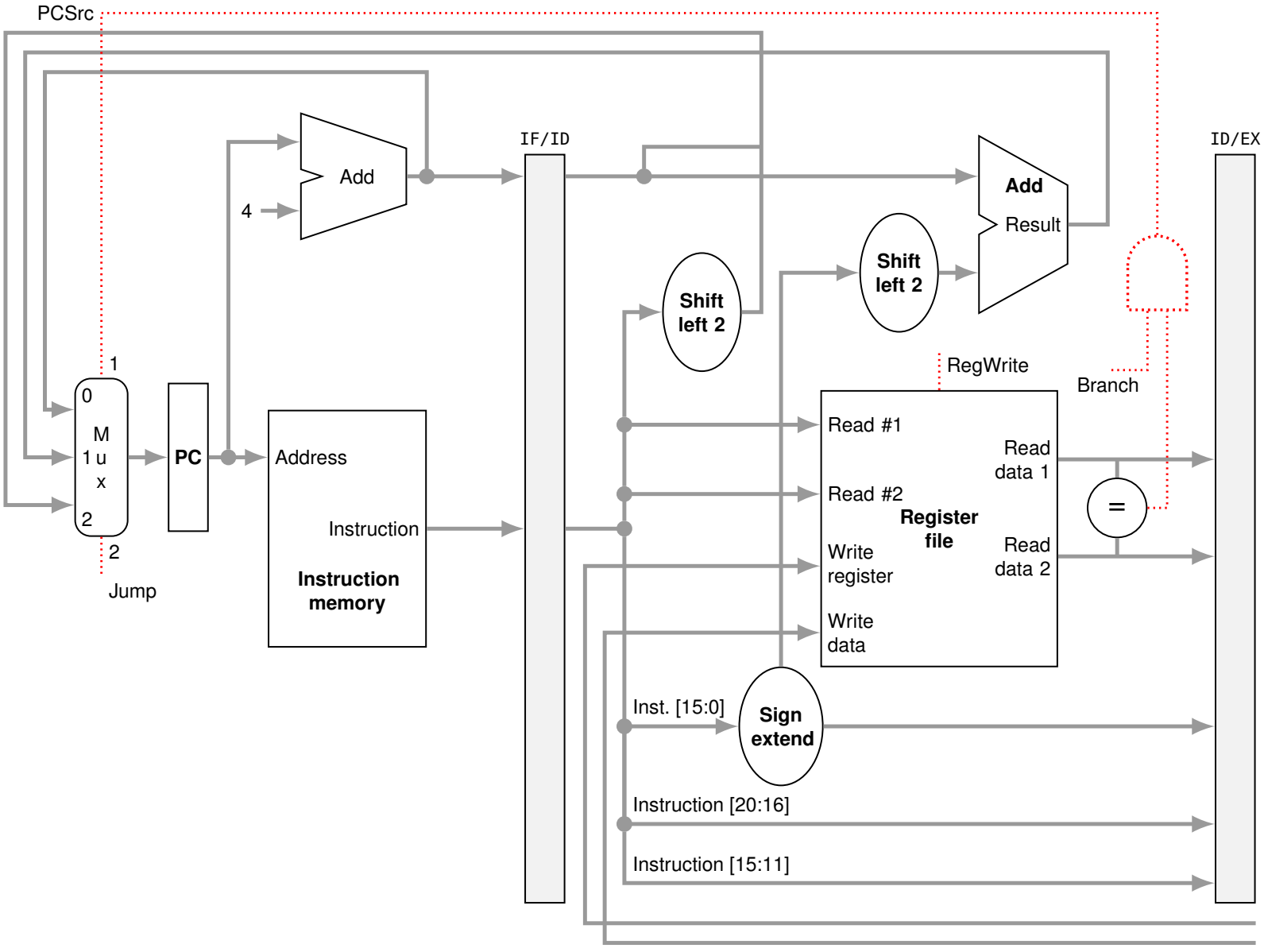
Evaluación agresiva de saltos

Idea

- Adelantar evaluación salto
- Etapa `ID`
- ✘ Puede alargar duración del ciclo
- ✘ Nuevas rutas de reenvío

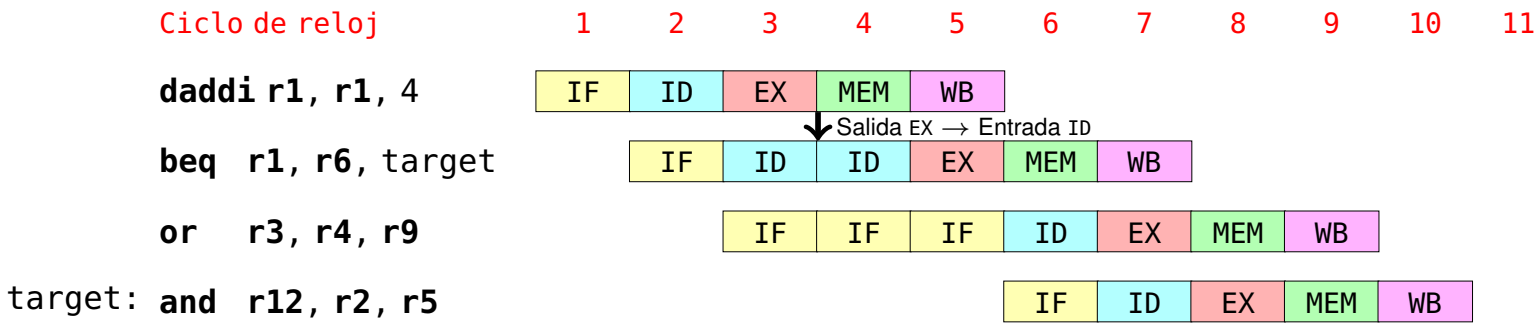


Evaluación agresiva de saltos



Evaluación agresiva de saltos

Ruta reenvío



Predicción de saltos

Objetivo

Reducir penalización por riesgos de control

- No necesaria participacion del compilador (pero aconsejable)

Ejecución especulativa

Por una de las dos ramas (salto/no salto)

- Acierto $\Rightarrow$ no hay detención
- Fallo $\Rightarrow$ deshacer trabajo (registros/memoria)

Predicción de saltos

Ejemplo

```
  beq  r4, r5, target
  dadd r9, r0, r7
  dsub r11, r10, r5
  or   r12, r5, r14
target:
  dadd r7, r1, r3
```

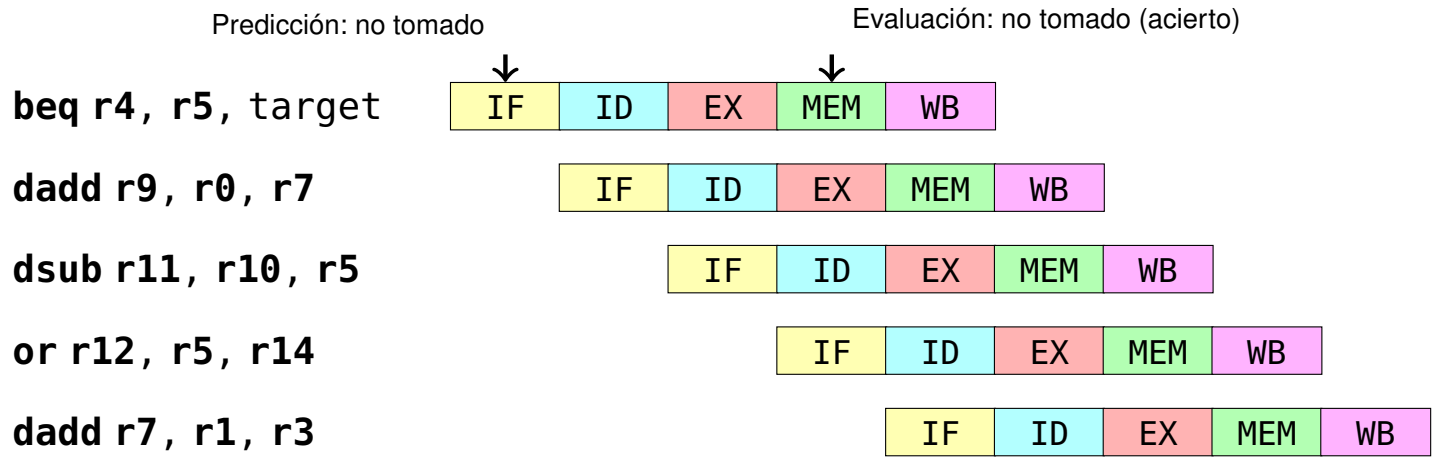
Algoritmo: siempre no tomado

- Simple
- Asume que nunca se toma el salto

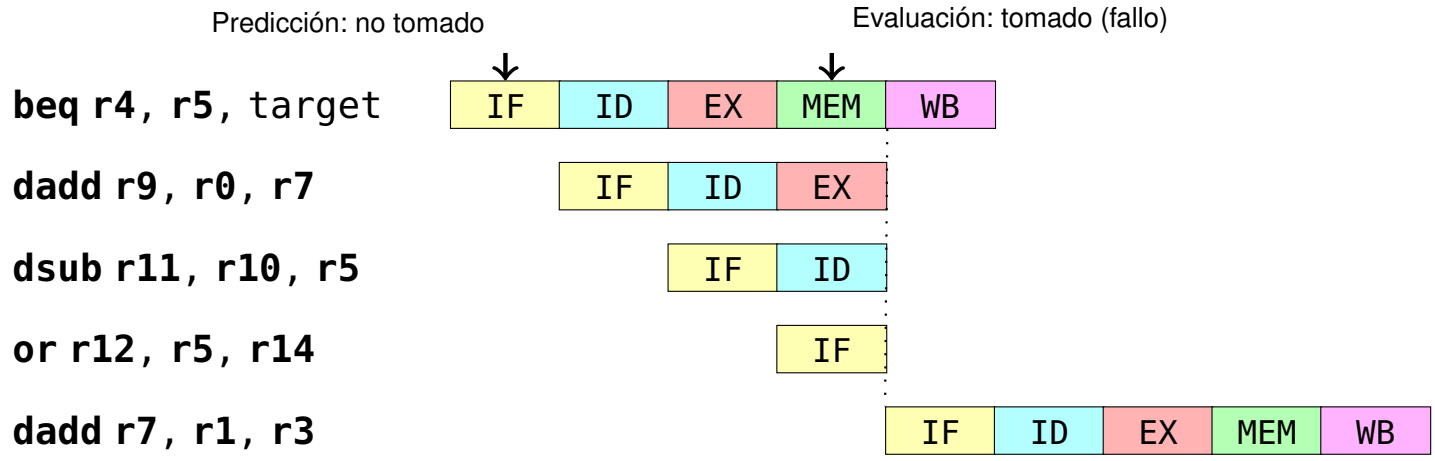
Predicción de saltos

Ejemplo

Acierto: no hay salto



Fallo: hay salto



Predicción de saltos

Siempre no tomado

- Baja efectividad (≈33 %)
- Predicción estática
- Posible: colaborar con el compilador para especular o no

Ejemplo

```
  .....
  ori   r1, r0, 100 ; r1 = 100
startf:
  daddi r1, r1, -1  ; r1 = r1-1
  bnez  r1, endf    ; jump to startf if r1 is not zero
endf:
  .....
```

Predicción de saltos

Predictor dinámico de dos bits

- Estado: dos bits (contador)
- Aumenta si tomado / decrementa si no tomado
- BHT (Branch History Table)
- BTB (Branch Target Buffer)
- Acierto $\approx$90 %

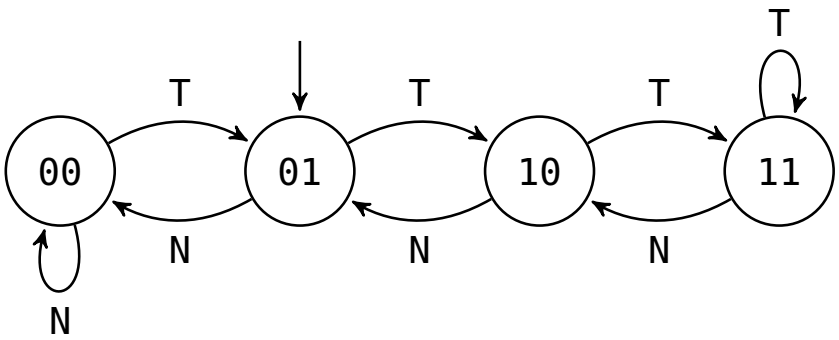
Primera ejecución de salto

Nueva entrada en tabla BHT: BHT + BTB (simplificacion)

- Dirección memoria de instrucción (acceso en `IF`)
- Dirección destino salto (cuando se evalúa)
- Bits de predicción (por defecto `01`)

Predicción de saltos

Estados

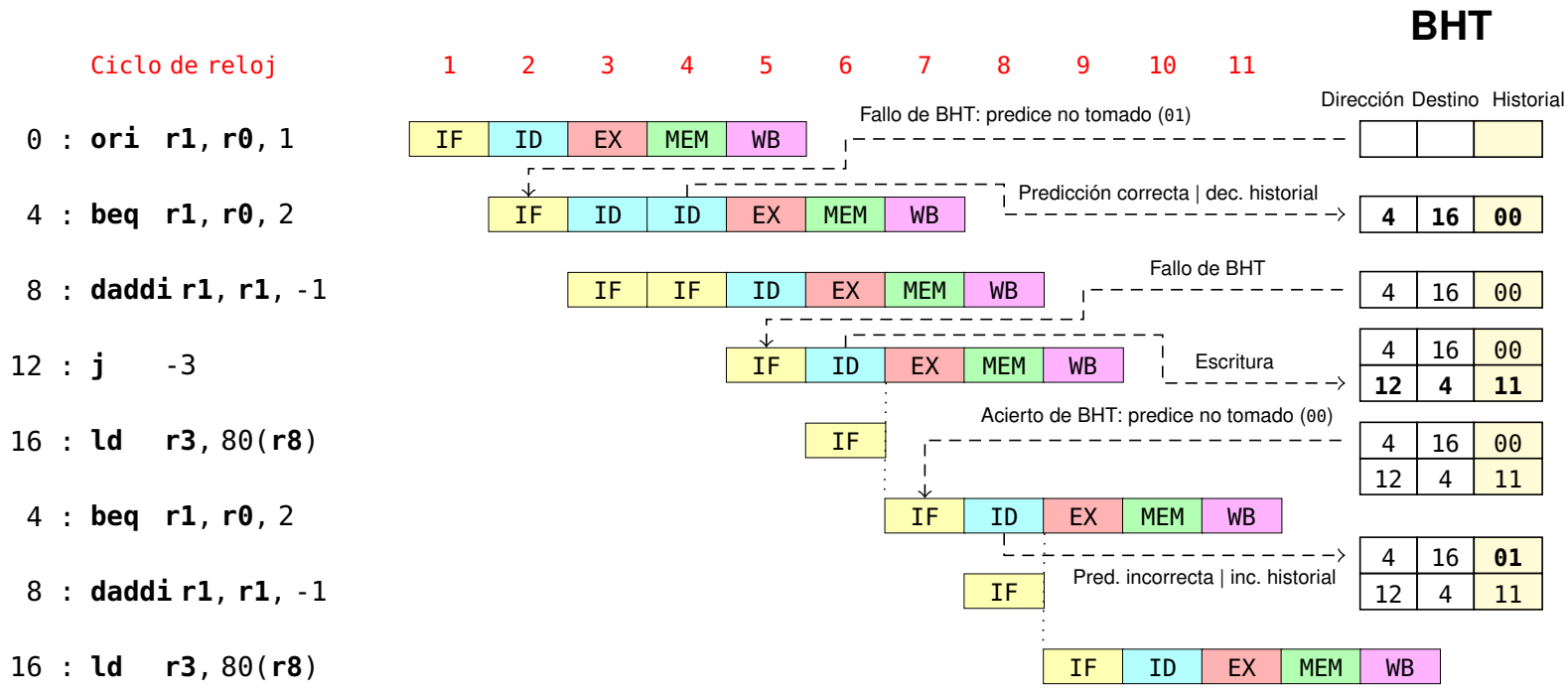


BHT

Dirección	Destino del salto	Historial
1010101...	0011101...	01
0111010...	1100010...	10
1010101...	0111011...	11

Predicción de saltos

Ejemplo



Profundidad segmentación

Aumentar número de etapas

- ✔ Más etapas $\Rightarrow$ menor tiempo de ciclo
- ✔ Mayor producitividad
- ✘ Más retardo por registros de segmentación
- ✘ Replicar recursos hardware (multipuerto)
- ✘ Efectos de riesgos de control
- ✘ Mayor frecuencia $\Rightarrow$ mayor disipación calor

Típicamente: 10 a 20 etapas

Mejoras de rendimiento

$$T_{CPU} = Nº \text{ instrucciones} \times CPI \times T$$

CPU segmentada

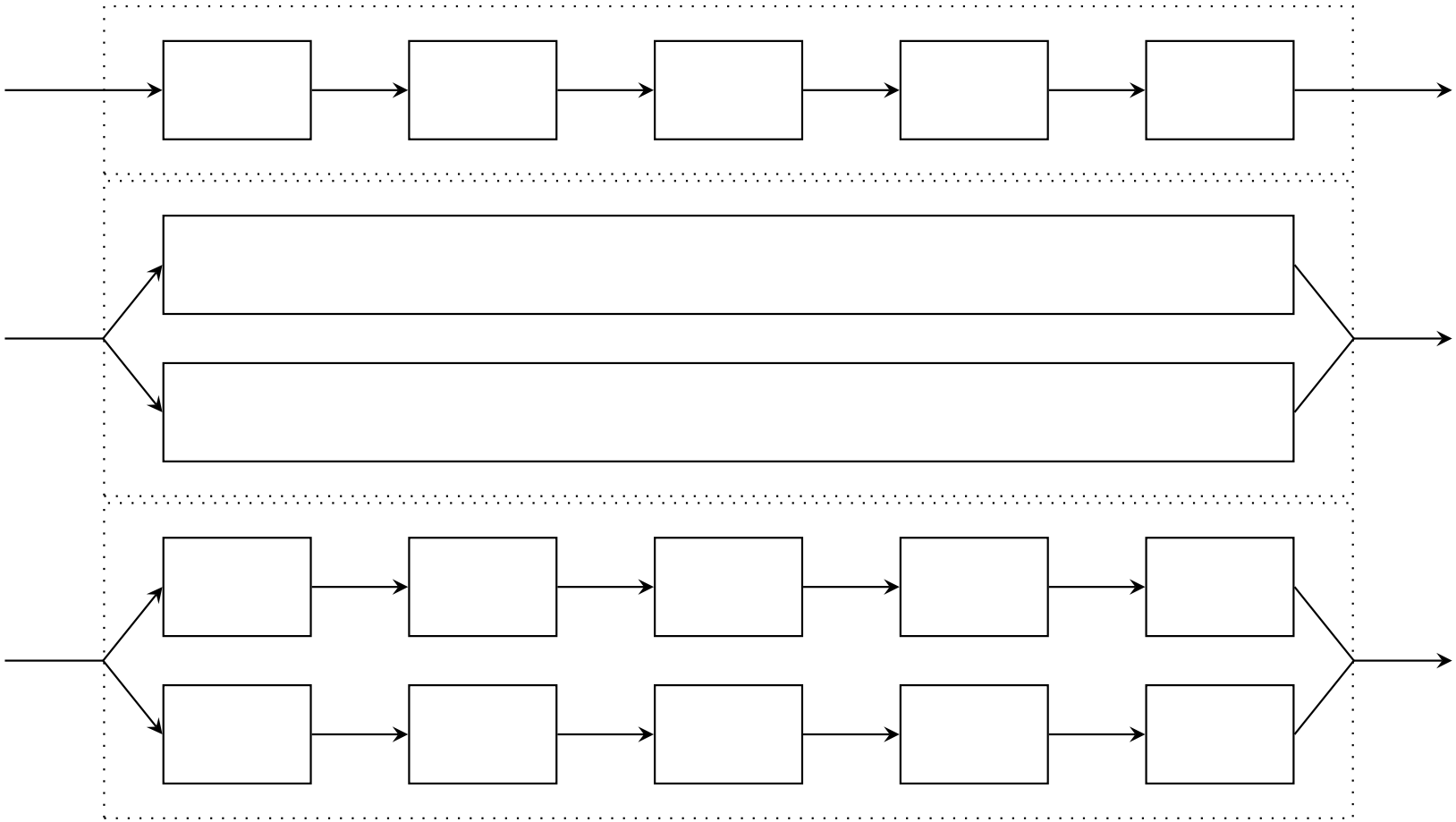
- Nº instrucciones: fijo por la ISA
- CPI ≥ 1
- T limitado por muro de potencia

Solución: nuevas microarquitecturas

CPI < 1

- Nueva métrica: IPC=CPI^{-1}
- Objetivo: IPC>1

Tipos de paralelismo



Emisión múltiple

Requisitos

Ancho de emisión

Número instrucciones simultáneas por etapa

- Mínimo para todas las etapas
- Algunas etapas mayor ancho

Planificación de instrucciones

- Decidir qué cauce usar (HW y/o SW)
- Resolver dependencias cruzadas

Emisión múltiple

VLIW (*Very Large Instruction Word*)

- Planificación estática (compilador)
- Compilador respeta dependencias
- Compilador se adapta a microarquitectura
- Necesario apoyo del compilador

Superescalar

- Planificación dinámica (CPU)
- CPU resuelve riesgos
- Conveniente apoyo del compilador pero no imprescindible

Paralelismo a nivel de instrucción (ILP)

ILP

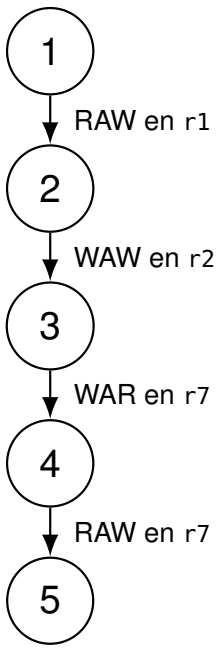
Independencia entre instrucciones

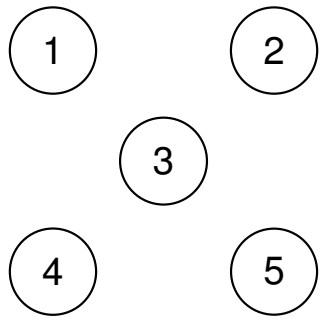
- Riesgos control minimizables (predicción)
- Riesgos RAW inevitables
- Límite máximo paralelismo (muro ILP)

Paralelismo a nivel de instrucción (ILP)

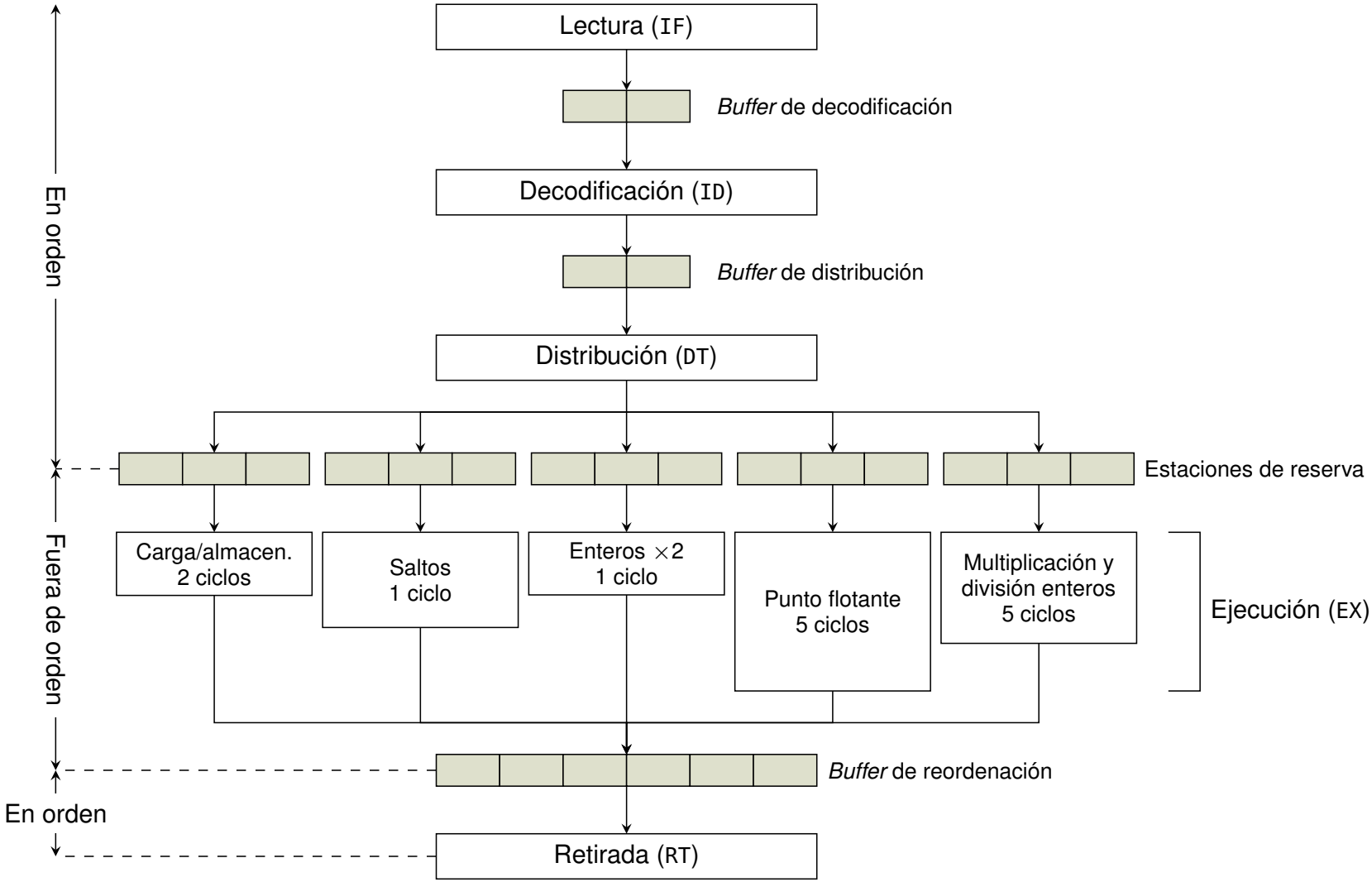
```
(1) dadd r1, r3, r4
(2) xor r2, r5, r1
(3) dsub r2, r4, r7
(4) movz r7, r4, r6
(5) slt r9, r7, r6
```

```
(1) dadd r3, r2, r1
(2) xor r6, r4, r1
(3) dsub r8, r4, r7
(4) movz r9, r4, r5
(5) slt r10, r7, r5
```

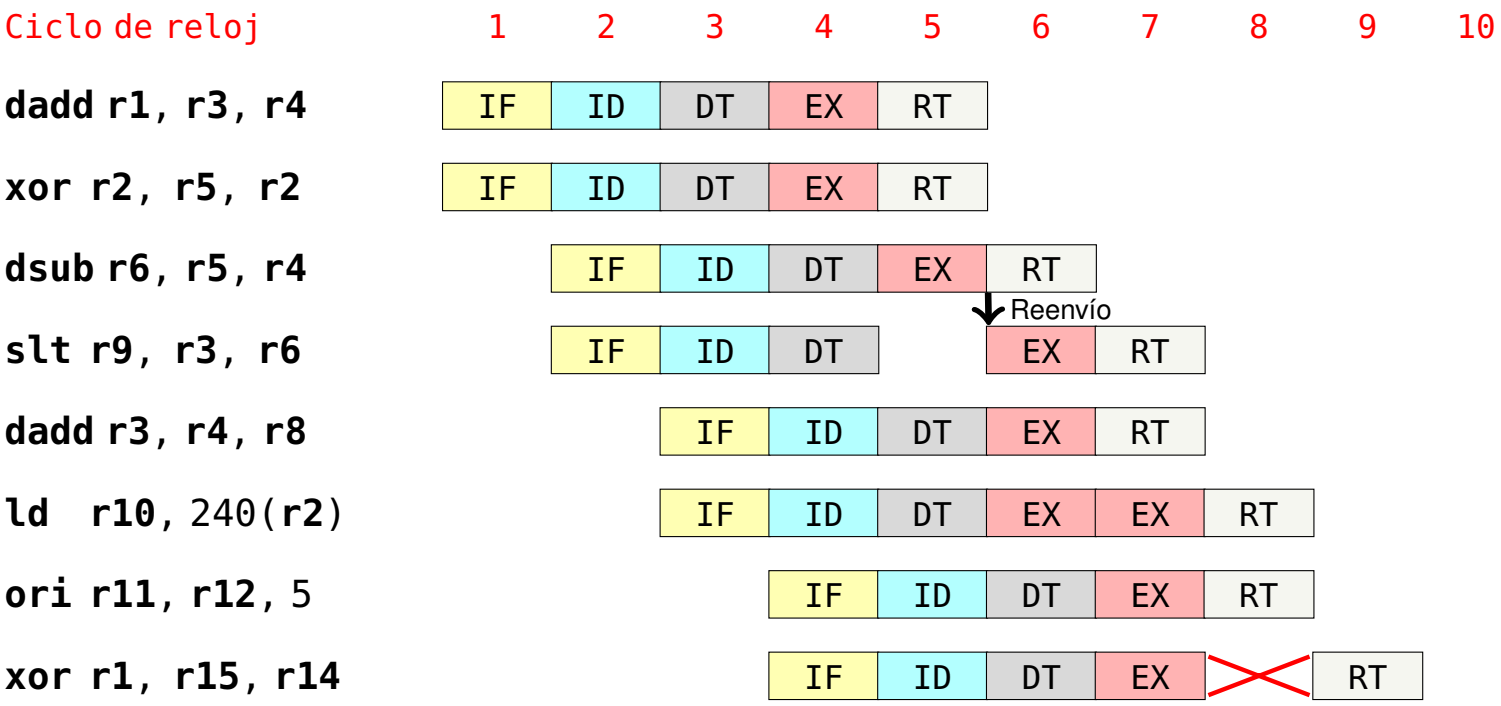




CPU superescalar



CPU superescalar



Seguridad

Ejecución especulativa

Revertir los efectos de las instrucciones

- Predicción de saltos (fallo)
- Ejecución OoO (excepción)

Problema

Deja huellas

- Puede ser utilizado para leer cualquier posición de memorua
- Utilizando canales laterales
- Ejemplos: Spectre, Meltdown

Construcción de procesador

Usando circuitos integrados

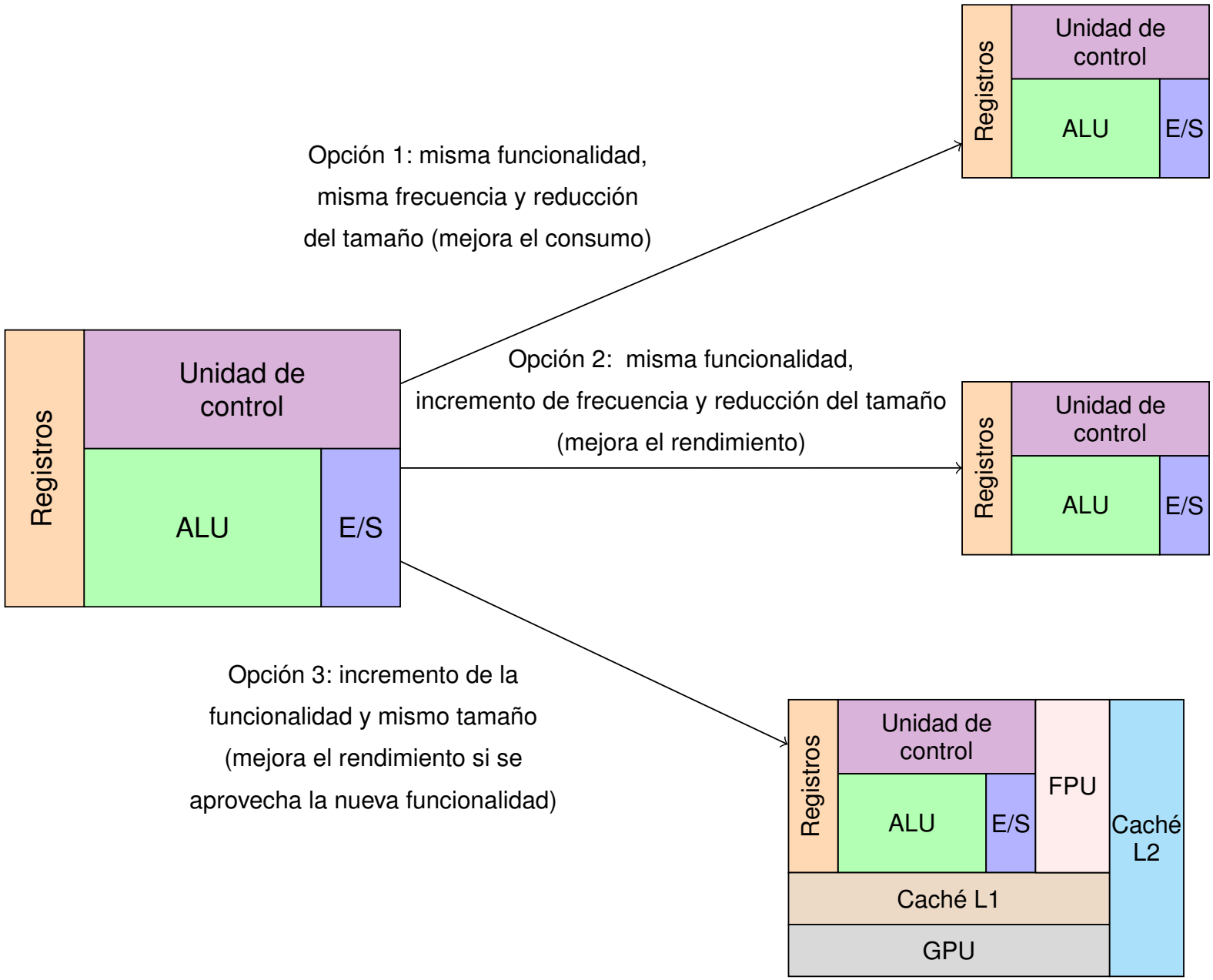
- Oblea de silicio
- Estampación litográfica/química
- Varios chips en la misma oblea
- Varias capas
- Mejorar tecnología de fabricación $\Rightarrow$ transistores más pequeños

Resumen

Ritmo exponencial:

- Aumento potencia computacional
- Reducción de costes

Mejora tecnología fabricación

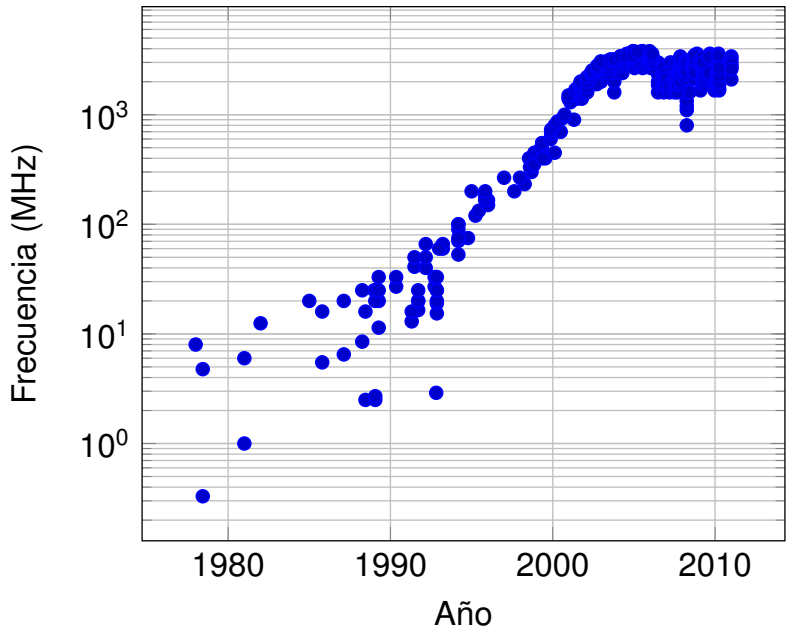


Power wall

$$T_{CPU} = \text{Nº instrucciones} \times \text{CPI} \times T$$

Límite al aumento de frecuencia

- Mayor consumo
- Mayor disipación de calor
- Mayor coste



ILP wall

$$T_{CPU} = \text{Nº instrucciones} \times CPI \times T$$

Límite al paralelismo de instrucciones

- Inherente a los programas
- Limitan el *pipeline* y la superescalabilidad

Solución

Añadir más núcleos

- Cambio de paradigma de programación
- *Memory wall*
- *Brick wall = Power wall + ILP wall + Memory wall*

Taxonomía de Flynn

Arquitecturas categorizadas según el número de flujos concurrentes de instrucciones y datos

1. SISD: Single Instruction, Single Data
   - CPU secuenciales y segmentadas
2. SIMD: Single Instruction, Multiple Data
   - GPU, instrucciones multimedia (MMX, SSE, . . .)
3. MISD: Multiple Instruction, Single Data
   - sistemas tolerantes a fallos
4. MIMD: Multiple Instruction, Multiple Data
   - Memoria distribuida: conjuntos CPU+memoria separados
   - Memoria compartida: varias CPU comparten memoria

Paralelismo a nivel de hilo

Hilo

Unidad mínima que un S.O. puede planificar

- un proceso puede tener varios hilos (al menos uno)

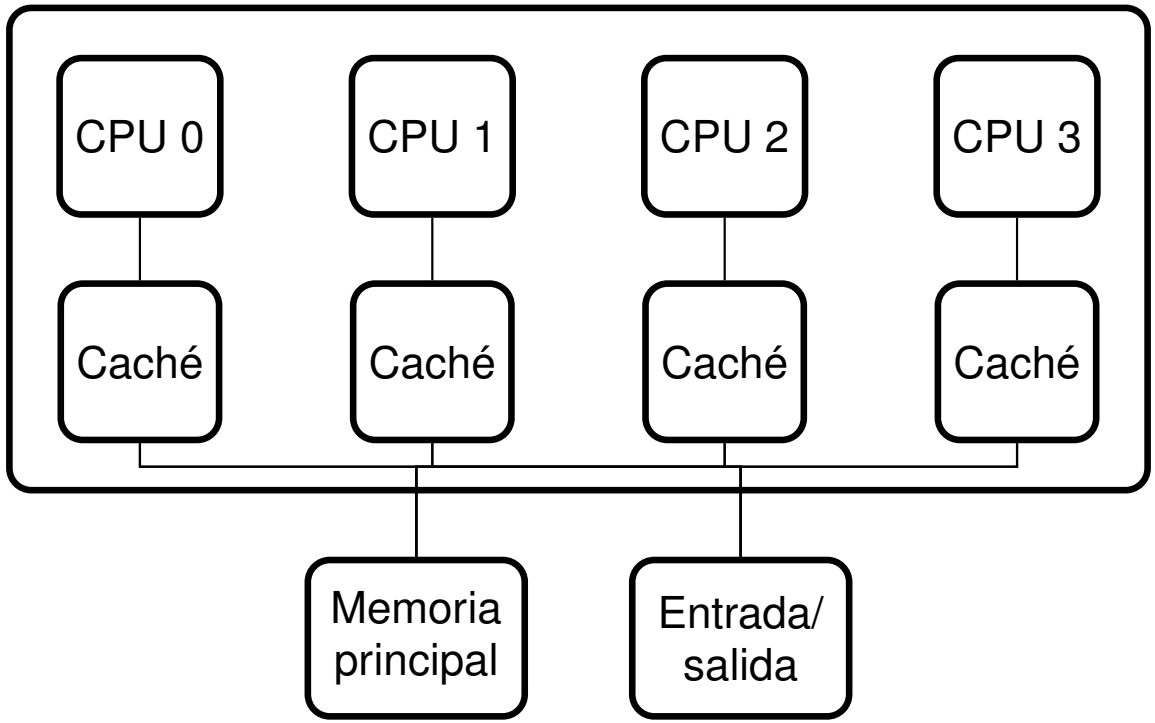
Paralelismo a nivel de hilo $\neq$ Paralelismo a nivel de instrucción (*pipeline* o arquitecturas superescalares)

Implicaciones

- Paralelismo a nivel de instrucción mejora la ejecución de cualquier programa
  - transparente al programador
- Programador debe implicarse para aprovechar paralelismo a nivel de hilo (TLP)
  - creación de hilos, sincronización, etc.

Sistemas multinúcleo

Los computadores actuales son MIMD con memoria compartida (*Shared Memory Multiprocessors*)



Simultaneos Multi-Threading (HyperThreading)

- Replicación de unidades funcionales
- Hilos concurrentes en un núcleo

S.O. Monotarea

Una tarea en ejecución en cada instante

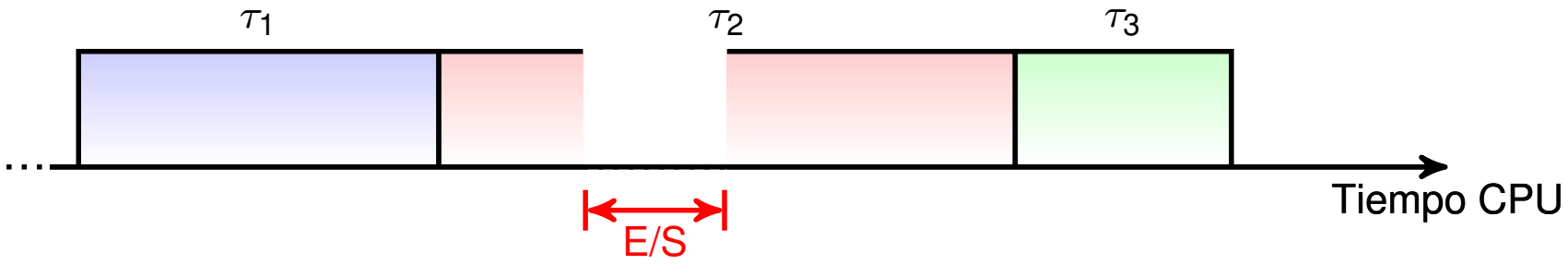
- CPU ejecuta un único programa
- memoria contiene código y datos del programa + S.O.
- interfaces de E/S disponibles para el programa

Desventajas

Utilización baja de los recursos

- uso de CPU $\Rightarrow$ interfaces de E/S desocupados
- E/S $\Rightarrow$ CPU desocupada

Ejemplo 3 tareas

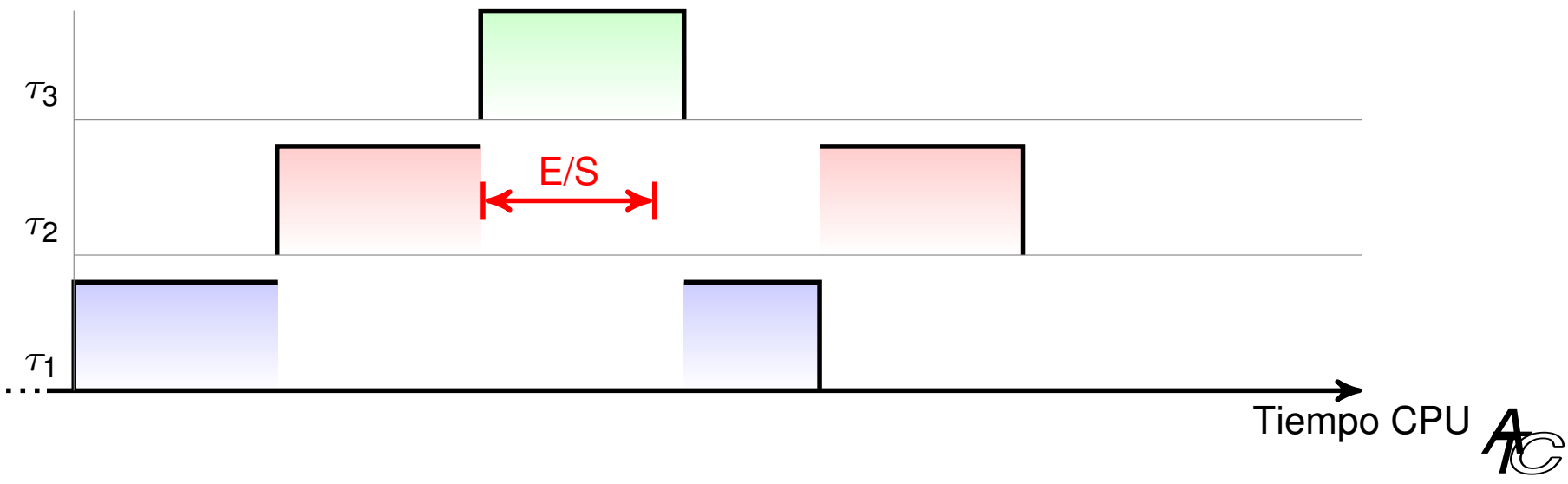


S.O. Multitarea

Varias tareas ejecutando simultáneamente

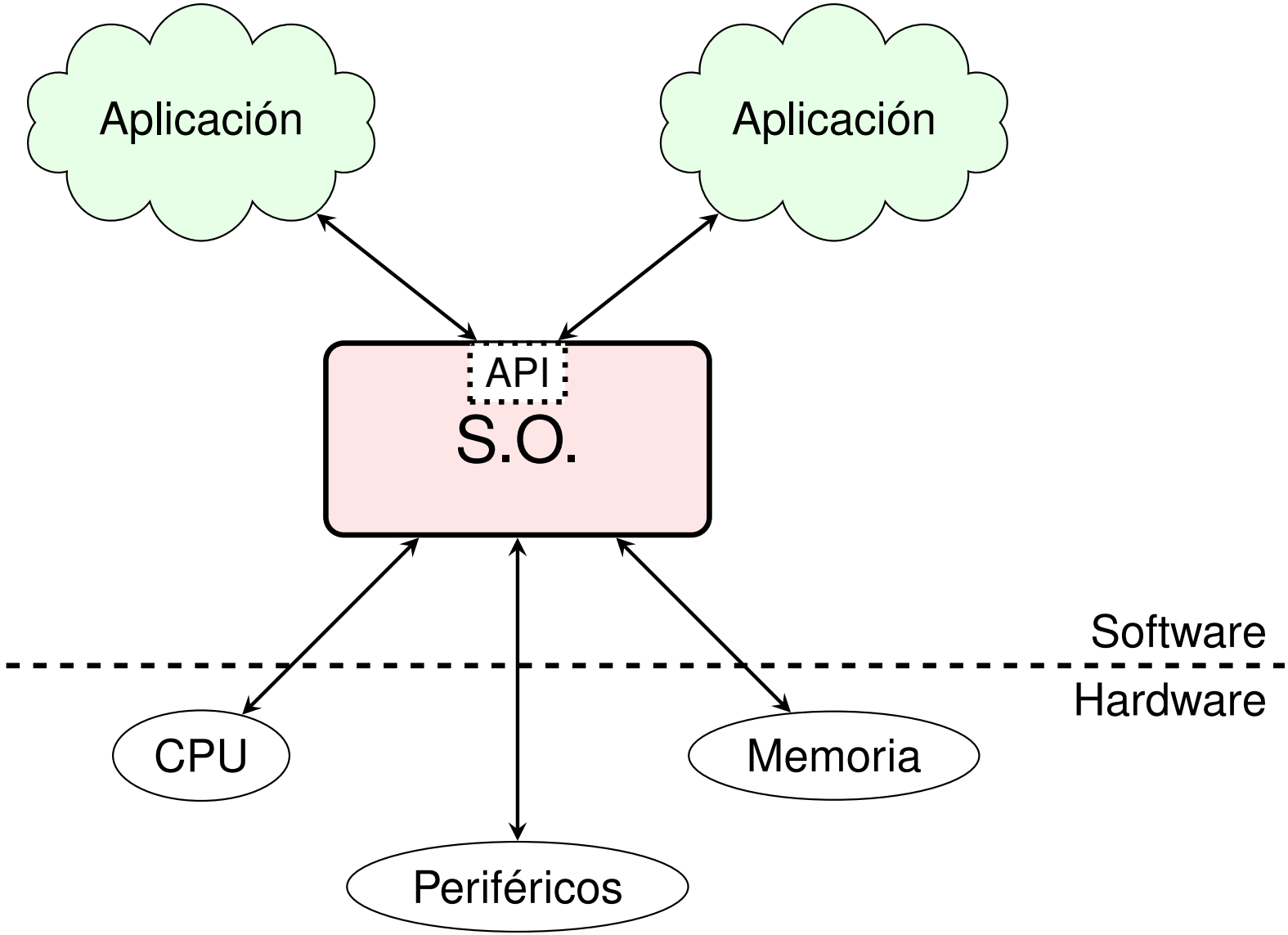
- ejecución paralela real o aparente $\Rightarrow$ varias CPU
- conmutación entre tareas (cambio de contexto)
- memoria contiene código y datos de varios programas + S.O.
- interfaces compartidos entre las tareas
- S.O. $\Rightarrow$ gestor de los recursos

✔ mayor utilización de los recursos



Definición de SO multitarea

Proporciona interfaz común a las aplicaciones (API)



Multitarea

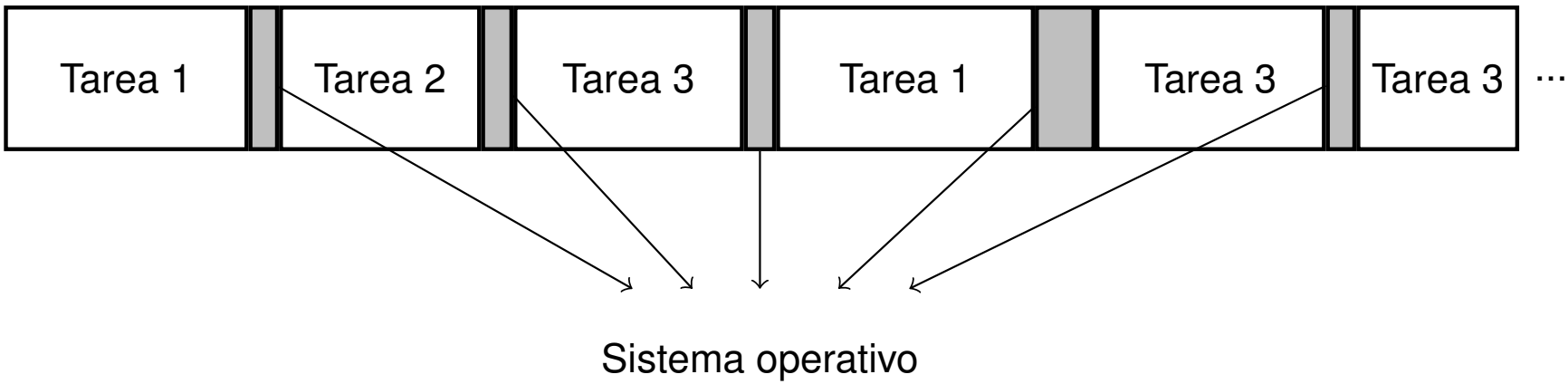
Definición

Capacidad para ejecutar varias tareas “simultáneamente”

- Problema: los recursos son únicos

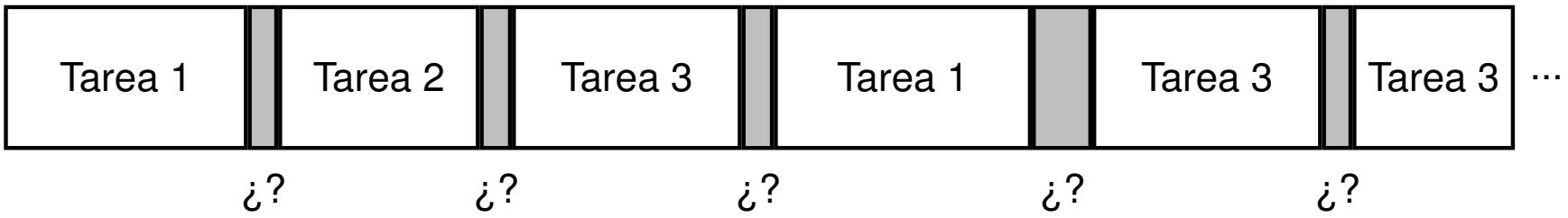
Solución

El S.O. gestiona los recursos y los comparte entre tareas



Si la asignación temporal es pequeña (*quantum*) se consigue concurrencia aparente

Mecanismos de transferencia



S.O. actúa como gestor de recursos

¿Bajo qué condiciones se transfiere el control al S.O.?

- Una tarea realiza una llamada a un servicio del S.O.
- Se produce una interrupción
- Se produce una excepción

Mecanismos de transferencia

- Cada caso implica la ejecución de un manejador distinto (rutinas de servicio)
- El manejador puede modificar el estado de las tareas
- Los mecanismos de transferencia pueden anidarse
  - Por ejemplo una excepción dentro de una interrupción
- Nomenclatura dependiente de la arquitectura

Añadir temporizador $\Rightarrow$ S.O. toma el control periodicamente

Soporte MIPS64 a los SO multitarea

Niveles de privilegio

Ejecución de instrucciones privilegiadas

- Usuario: `UM`=1, `ERL`=`EXL`=0
- Supervisor: `UM`=0, `ERL`=`EXL`=1

Memoria virtual paginada

Proteger al SO y a las tareas entre sí

- SO desde: 0x4000 0000 0000 0000

Recuperación estado ejecución

Proteger al SO y a las tareas entre sí.

- Registros `EPC` y `cause register`
- Instrucción `eret`

Virtualización

Definición

Ejecutar varios SO sobre máquina física

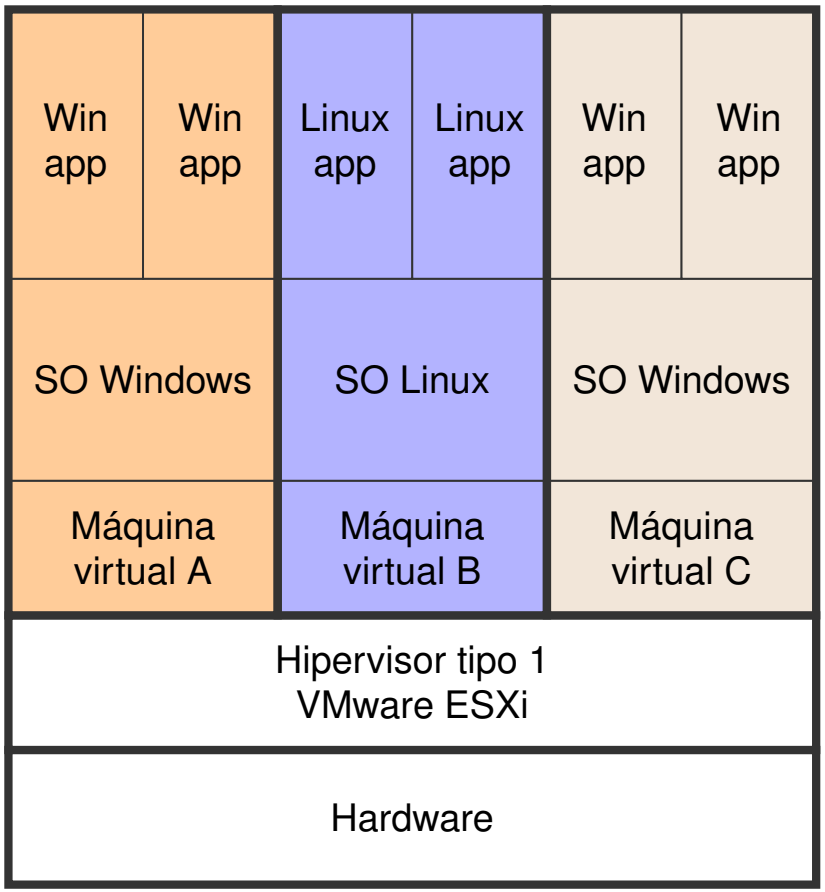
- Diferentes SO
- Reducción de costes y energía
- Portabilidad $\Rightarrow$ fácil administración
- Aislamiento $\Rightarrow$ seguridad

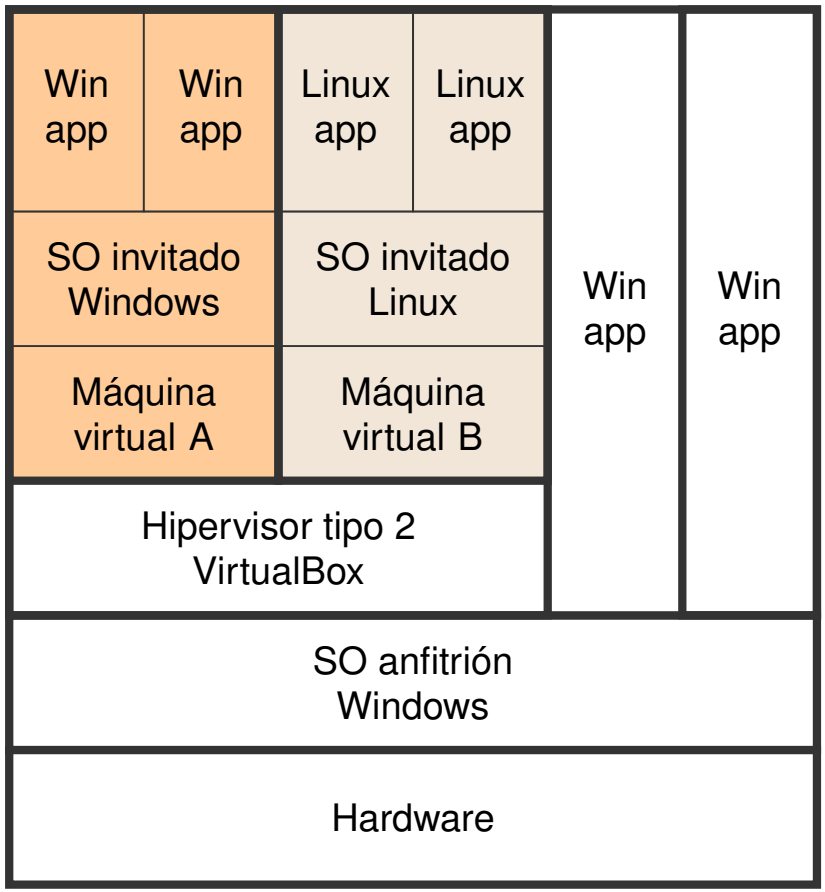
Hipervisor

Programa más privilegiado que el SO

- Gestor de recursos físicos entre máquinas virtuales

Hipervisor


Win app
Win app
Linux app
Linux app
Win app
Win app
SO Windows
SO Linux
SO Windows
Máquina virtual A
Máquina virtual B
Máquina virtual C
Hipervisor tipo 1
VMware ESXi
Hardware


Win app
Win app
Linux app
Linux app
Win app
Win app
SO invitado Windows
SO invitado Linux
Máquina virtual A
Máquina virtual B
Hipervisor tipo 2
VirtualBox
SO anfitrión
Windows
Hardware

Virtualización

CPU sin soporte

Sigue siendo posible

- SO invitado se ejecuta en modo de bajo privilegio
- Trap-and-emulate
- ✘ Instrucciones sensibles
  - Traducción binaria $\Rightarrow$ reduce rendimiento
  - paravirtualización $\Rightarrow$ SO invitado se modifica

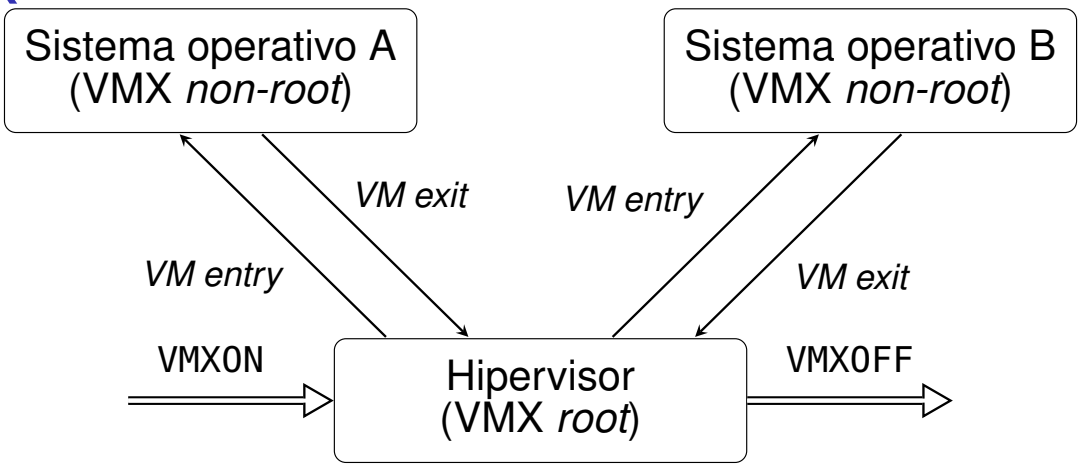
Actualmente

CPU proporciona soporte

- No es necesaria la traducción binaria
- Paravirtualización es opcional

Virtualización: soporte x86 (VT-x)

Modo VMX



Nuevas instrucciones

Rendimiento $\approx$ Máquina física

- `VMLAUNCH` $\Rightarrow$ inicia VM
- `VMRESUME` $\Rightarrow$ restaura VM