IV. Segmentación o Pipelining

Alternativas de Implementación de Procesador

1. Procesador Uniciclo

   $\Rightarrow$ CPI = 1 Pero Período de Reloj Grande

2. Procesador Multiciclo

   $\Rightarrow$ CPI > 1 Pero Período de Reloj más Pequeño

En la Implementación Multiciclo No Todas las Unidades Funcionales se Utilizan en Todos los Ciclos de Reloj. Por Ejemplo:

$\Rightarrow$ Lectura de Registros Sólo en Ciclo 2

$\Rightarrow$ Escritura de Registros Sólo en Ciclo 4 ó 5

¿Es Posible Utilizar Estos Recursos Ociosos Para Reducir el CPI del Procesador Multiciclo?

$\Rightarrow$ Paralelismo a Nivel de Instrucciones

$\Rightarrow$ Permitir que una Instrucción se Ejecute Mientras Aún Se Está Ejecutando Otra

Segmentación

¿Qué Hace Fácil la Segmentación?

⇒ Todas las Instrucciones Tienen el Mismo Largo

⇒ Existen Pocos Formatos de Instrucción

⇒ Los Operandos de Memoria Aparecen Sólo en *lw* y *sw*

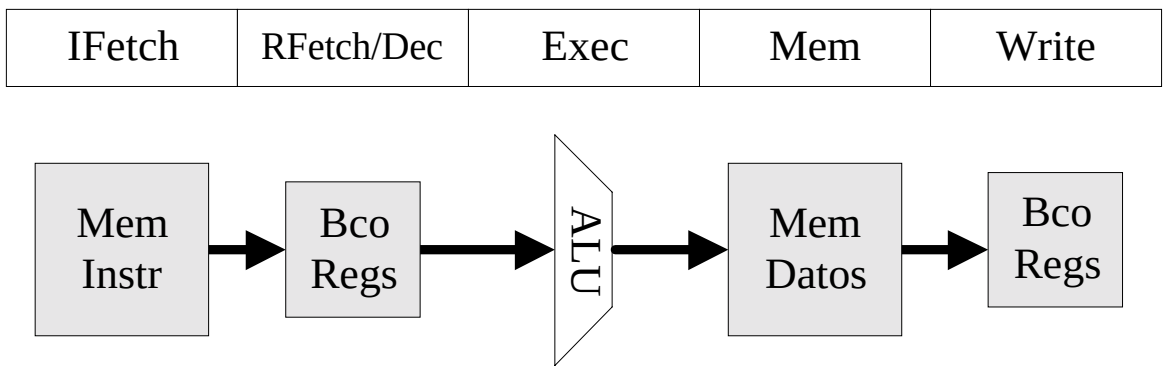
¿Qué Hace Difícil la Segmentación?

⇒ Peligros Estructurales: Competencias por Recursos

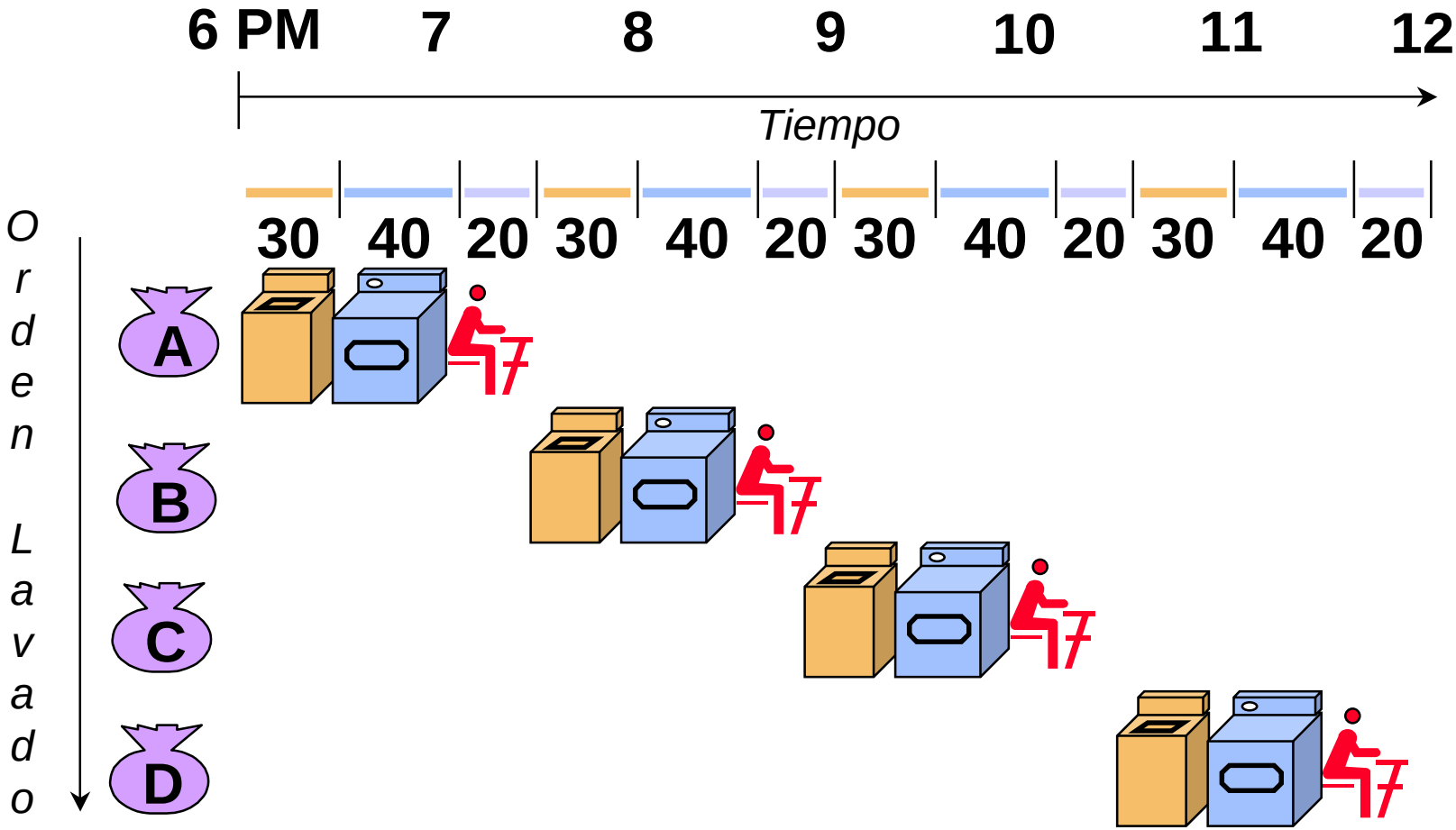
⇒ Peligros de Control: ¿Qué Pasa en los Branches?

⇒ Peligros de Datos: Dependencia de Datos Entre Instrucciones

Las 5 etapas de una Instrucción *lw*

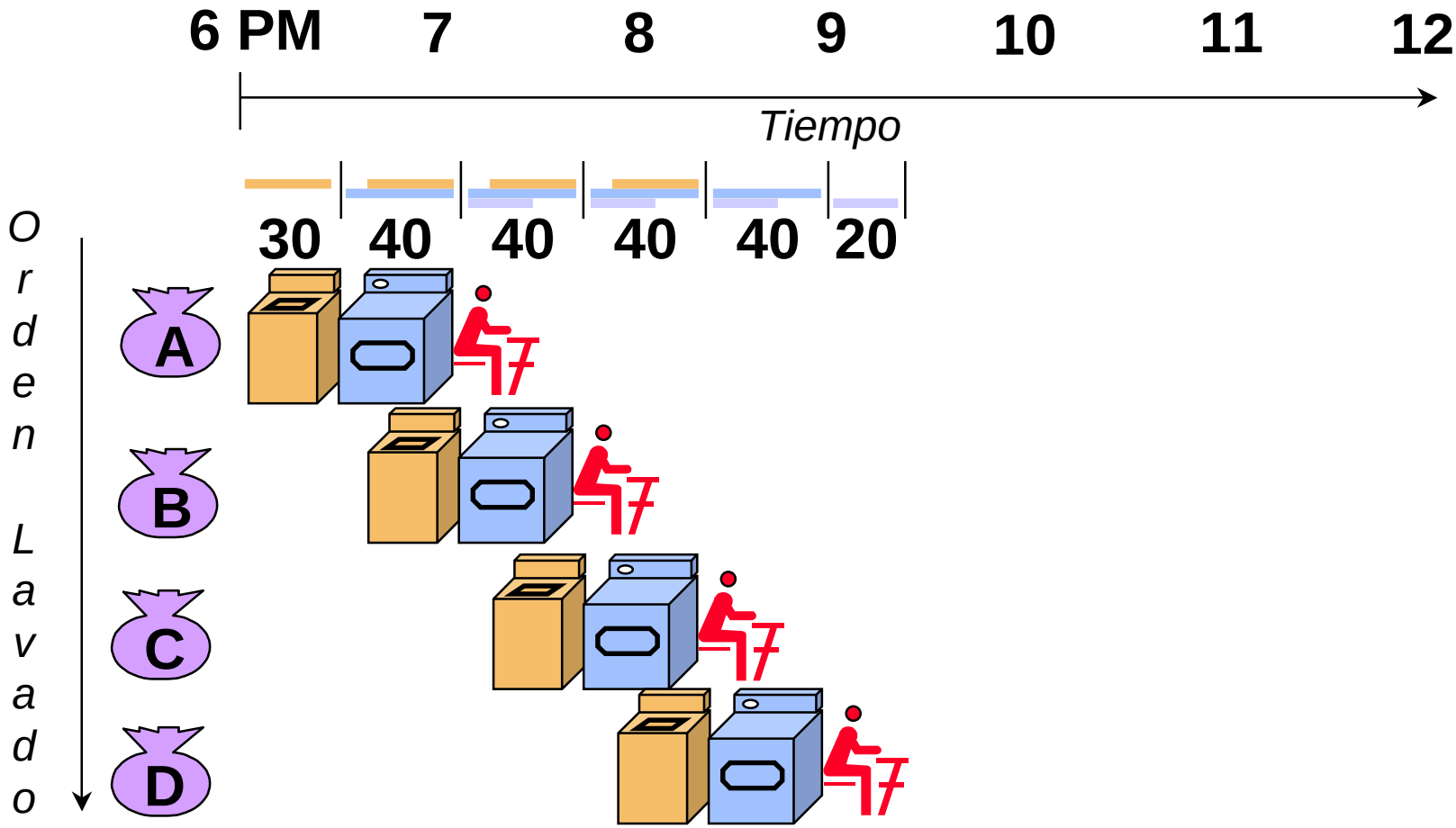


Segmentación



Una Lavandería Secuencial Tarda 6 Horas Para 4 Cargas de Lavado

Segmentación



Una Lavandería Segmentada Tarda 3.5 Horas Para las Mismas 4 Cargas de Lavado

Segmentación

La Idea Principal es que las Instrucciones Utilizan Recursos Distintos en Distintas Etapas de su Ejecución, por lo tanto, Es Posible Ejecutar Simultáneamente Múltiples Instrucciones, Si Éstas Se Encuentran Todas en Distintas Etapas de Ejecución

Técnica de Segmentación

⇒ En Cada Ciclo, Se Inicia la Ejecución de una Instrucción

⇒ Existen Múltiples Instrucciones en Ejecución, en Distintas Etapas

⇒ Es Posible Iniciar la Ejecución de una Instrucción en Cada Ciclo de Reloj

⇒ El CPI Efectivo es 1

Observaciones

⇒ La Segmentación no Mejora la Latencia Individual, Mejora el Throughput Global

⇒ Las Instrucciones Individuales Siguen Demorando lo Mismo

⇒ La Segmentación Está Limitada por la Etapa más Lenta

⇒ Si las Etapas Están Desbalanceadas la Mejora a Conseguir se Reduce

Segmentación de Instrucciones: Peligros Estructurales

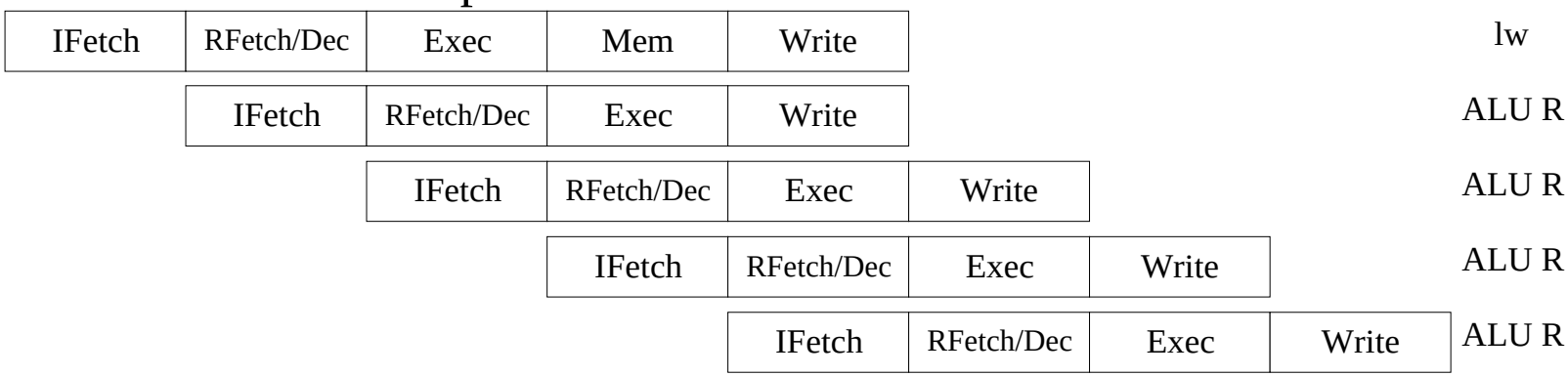
Ocurren Conflictos en Acceso a Recursos de Hardware en Distintas Etapas de la Ejecución (*structural hazards*)

Idea Principal:

⇒ Cada U.F. Puede ser Utilizada Sólo una Vez por Instrucción

⇒ Cada U.F. Debe ser Usada en el Mismo Estado por Todas las Instrucciones

Instrucciones ALU Tipo R



Conflicto en Escritura al Banco de Registros

Soluciones:

⇒ Definir un Estado Mem que no Hace Nada

⇒ Se Inserta un Ciclo Ocioso por Hardware

Segmentación de Instrucciones: Peligros Estructurales

Instrucciones ALU Tipo I
Conflicto en Escritura al Banco de Registros
Solución: Definir un Estado Mem que no Hace Nada

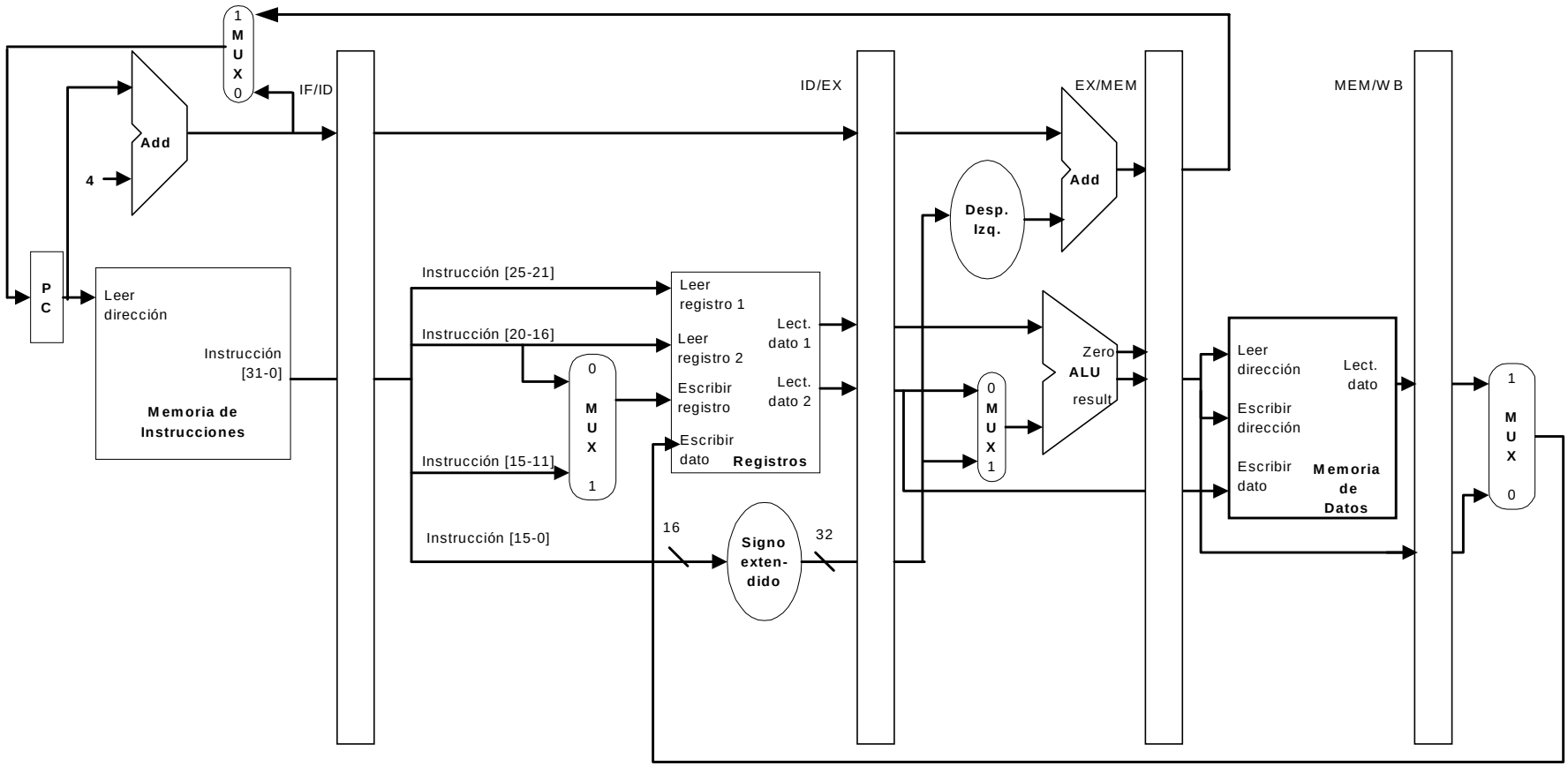
Instrucción *sw*
Definir un Estado Write que no Hace Nada

Instrucción *beq*
Evalúa Condición y Escribe el PC en Ciclo Exec
Lo Anterior Implica Tener una U.F. Para Cálculo de Dirección
Definir un Estado Mem y Uno Write que no se Hace Nada

Instrucción *jump*
Se Escribe el PC en Ciclo Exec
Se Define un Estado Mem y uno Write que no Hacen Nada

En Otras Palabras: Se Extienden Todas las Instrucciones a 5 Ciclos

Camino de Datos Segmentado

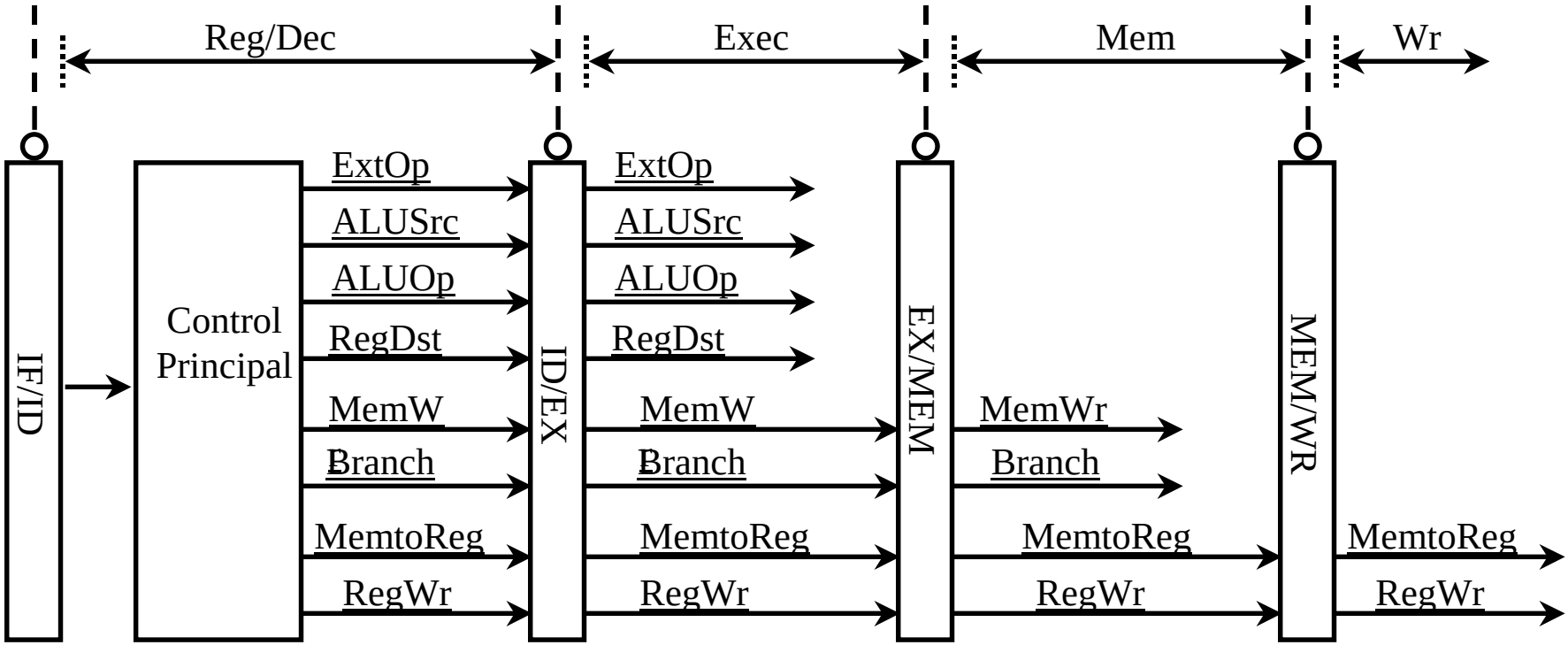


⇒ Similar al Procesador Uniciclo Pero se Agregan Registros de Pipeline

⇒ Cada U. F. Puede Ser Usada Sólo Una Vez Por Cada Instrucción

⇒ Cada U.F. Debe Ser Usada en la Misma Etapa por Todas las Instrucciones

Señales de Control Segmentadas



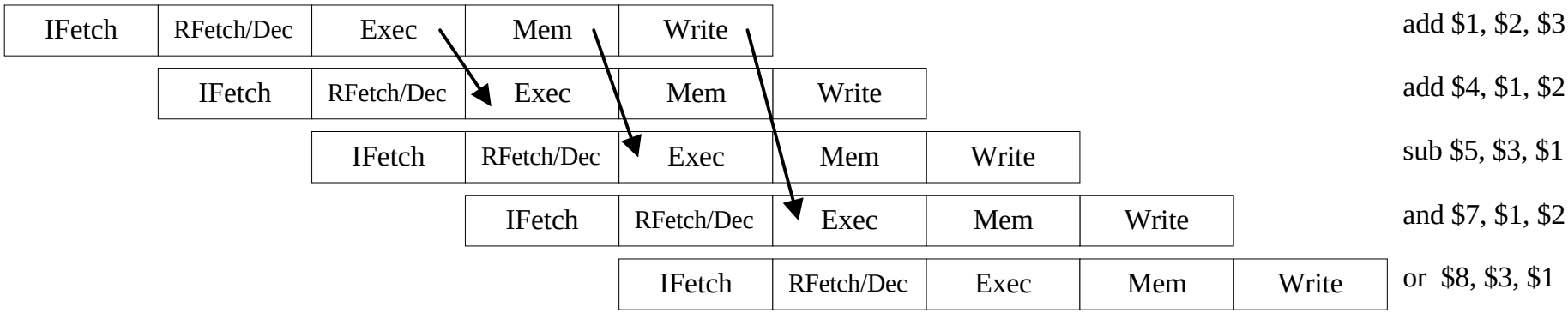
⇒ Las Señales de Control Deben Pasarse Entre Etapas

⇒ Utilizar los Mismos Registros de Pipeline Para Pasar las Señales

Segmentación de Instrucciones: Peligros de Datos

Dependencia de Datos Entre Instrucciones ALU

Un Conflicto de Datos Impide la Ejecución de las Siguientes Instrucciones



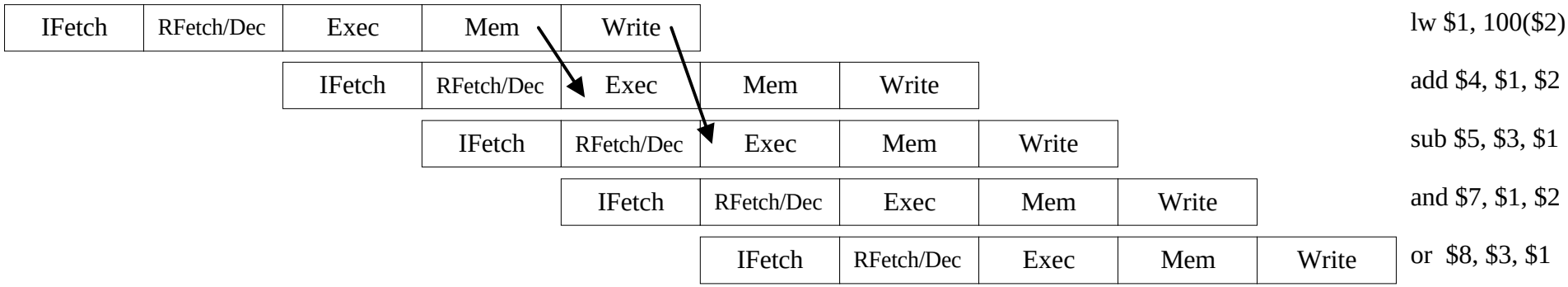
Solución Simple: Consiste en Insertar Burbujas en el Pipeline (Puede Bajar el Rendimiento si la Situación es Frecuente)

Solución por Forwarding: Detectar que Siguiente Instrucción Utiliza Resultado de Instrucción Actual y el Resultado de la ALU se Escribe en Registro Especiales del Pipelining, Esto Permite Eliminar las Burbujas (3)
El Forwarding se Implementa en el Estado *Exec* de la Instrucción que Produce el Dato y en el Estado *RFecthDec* de la Instrucción que lo Usa. Se Realiza lo Mismo Para las Instrucciones Siguientes, en los Estados *Mem* y *Write* de la Instrucción que Genera el Dato

Segmentación de Instrucciones: Peligros de Datos

Dependencia de Datos por Instrucción lw

Un Conflicto de Datos Impide la Ejecución de las Siguientes Instrucciones



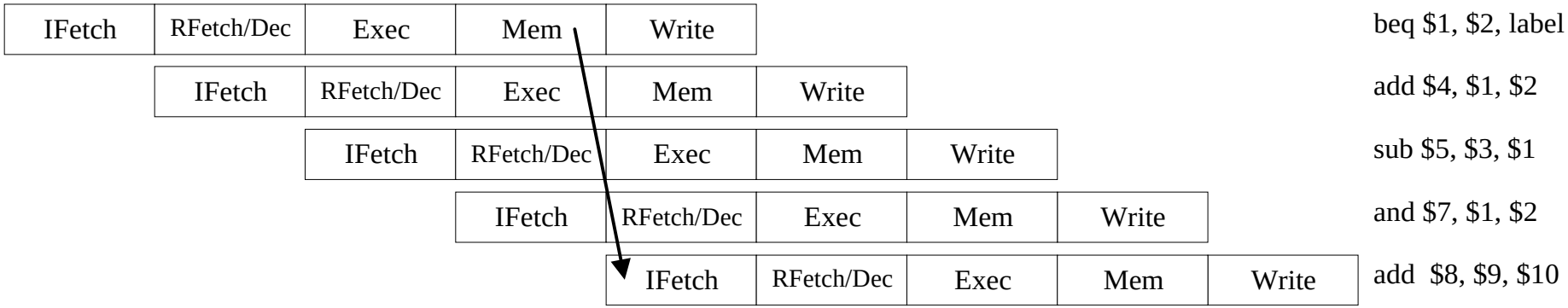
Solución por Forwarding: Permite Eliminar Sólo 2 Burbujas

Loads Retrasados: Si Se Expone el Comportamiento en la Arquitectura el Resultado de la Operación *lw* Está Disponible Dos Instrucciones Después lo que Permite al Compilador Intentar Colocar una Instrucción Útil Entre Medio, Si No Puede, Inserta un *nop* o el Hardware Puede Detectar la Situación e Insertar una Burbuja.

La Mayoría de los Procesadores Producen Esta Burbuja Por Hardware

Segmentación de Instrucciones: Peligros de Control

Dependencia de Control por Instrucción beq



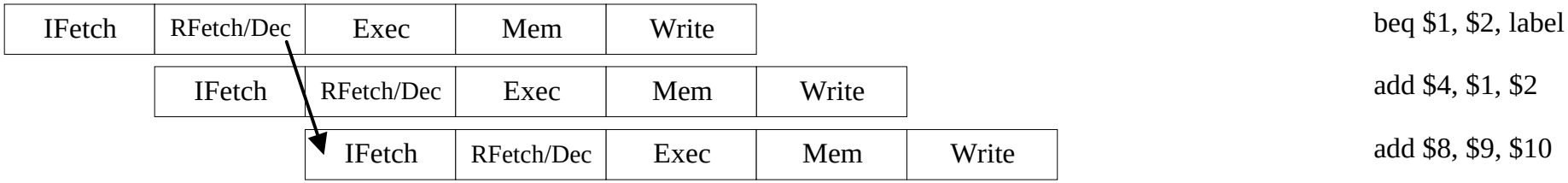
Retardo de 3 Instrucciones Antes de Hacer Efectivo el Salto
Para Saber si Ejecuto el Salto Necesito Conocer la Evaluación de la Condición y la Dirección Destino del Salto

¿Cómo Elimino el Retardo de 3 Instrucciones?
⇒ Utilizar Comparador Dedicado Para Resolver la Condición (*RFetch/Dec)*
⇒ Utilizar Sumador Dedicado Para Calcular Dirección de Salto (*RFetch/Dec)*

Con Esto Se Eliminan Dos Slots, Quedando un Retardo de una Instrucción

Segmentación de Instrucciones: Peligros de Control

Dependencia de Control por Instrucción beq



Saltos Retrasados (*Delayed Branches*)

⇒ Una Instrucción Anterior al Salto: Útil en Todos los Casos

⇒ Una Instrucción Posterior al Salto: Útil si el Salto no se Ejecuta

⇒ Una Instrucción de la Dirección Destino del Salto: Útil si el Salto se Ejecuta

Un Compilador Logra Llenar un 50% de los "*Delay Slots*"

Predicción de Saltos

Cuando Compilador no Puede Encontrar una Instrucción que Sea Útil Siempre, Intentar Predecir Resultado del Salto

⇒ Esquemas Estáticos: por el Compilador

⇒ Esquemas Dinámicos: en Tiempo de Ejecución por Hardware

Especialmente Útil en Procesadores Superescalares o Supersegmentados Donde el Costo del Salto es Mayor a una Instrucción

Segmentación y Excepciones

Ocurre una Excepción, Pero ¡¡¡ Se Están Ejecutando 5 Instrucciones !!!

⇒ ¿Cómo Detener el Pipeline? ¿Se Debe Reiniciar? ¿Cuál Instrucción Produjo la Excepción?

Ante una Excepción, el Control Debe Detectarla y Transferir el Control al Manejador de Excepciones Adecuado. Pero: en el Pipeline Puede Ocurrir Más de una Excepción Simultáneamente

⇒ Fetch: Page Fault; Acceso Desalineado a Memoria; Violación de Protección

⇒ Decode: Instrucción Inválida

⇒ Execution: Overflow Aritmético

⇒ Memory: Page Fault; Acceso Desalineado a Memoria

Excepciones Imprecisas

⇒El Problema Se Ignora

Arquitecturas *Precisas*

⇒ El Control Asegura Correcta Identificación (Solución Actual)