Arquitectura de Computadoras
Pipelining

Facultad de Ingeniería
Universidad de la República

Instituto de Computación
Curso 2011

Algunas ideas para mejorar el rendimiento...

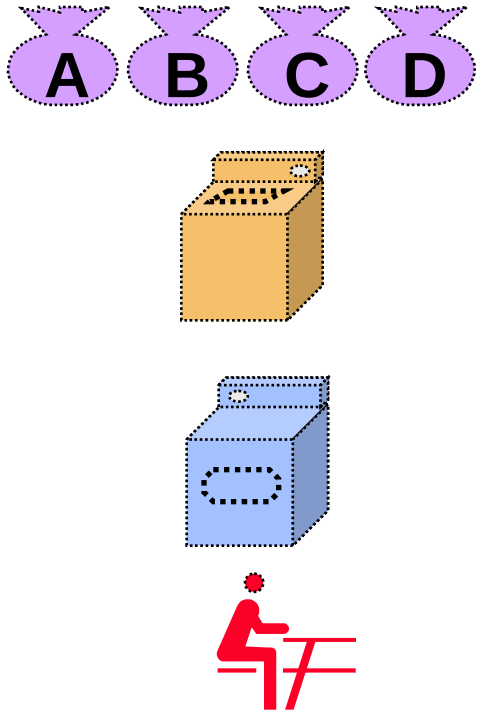
- Obvio: incrementar la frecuencia del reloj
  - Mayor frecuencia –> mayor velocidad de procesamiento
  - Estado del Arte: >2GHz
  - Pero...los tiempos de acceso a memoria y E/S pueden ser un cuello de botella...
- Ancho de los registros
  - Mayor procesamiento por ciclo
- Ancho del bus de datos
  - Mayor tasa de transferencia
- Ancho del bus de direcciones
  - NO mejora la velocidad de acceso
  - Pero se puede direccionar más memoria directamente...
    - Programas mas grandes
    - Menos uso de la memoria virtual. OK!

Más ideas para mejorar el rendimiento

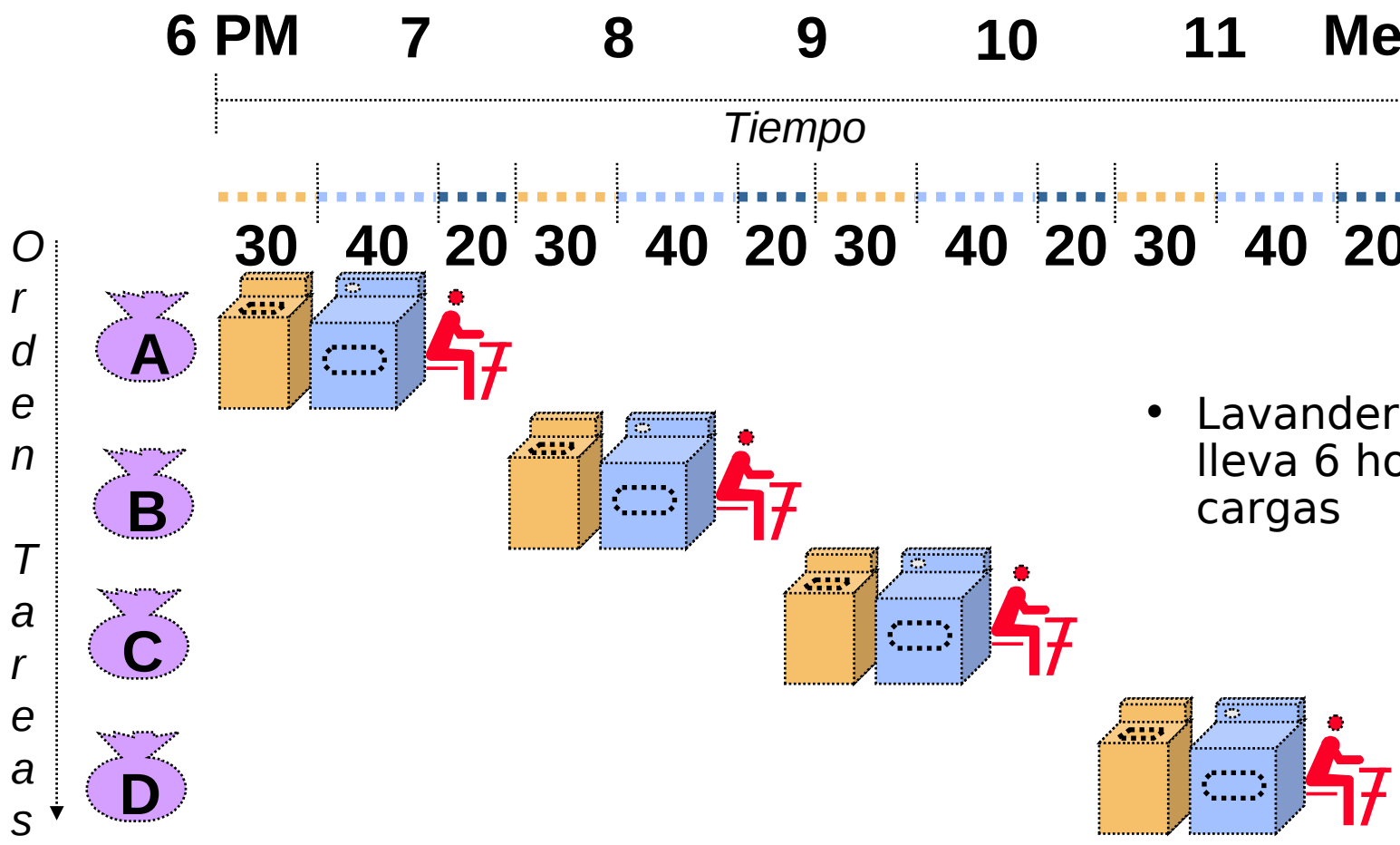
- (Obs: aún no manejamos definiciones formales asociadas al rendimiento de una arquitectura)
  - Pipelining
  - On board cache
  - On board L1 & L2 cache
  - Branch prediction
  - Data flow analysis
  - Ejecución especulativa
  - .....

Pipelining

- Ejemplo Lavandería
  - Ana, Beto, Carina, David tienen cada uno una carga de ropa para lavar, secar y ordenar
  - Lavadora lleva 30 minutos
  - Secadora lleva 40 minutes
  - “Ordenar” lleva 20 minutos

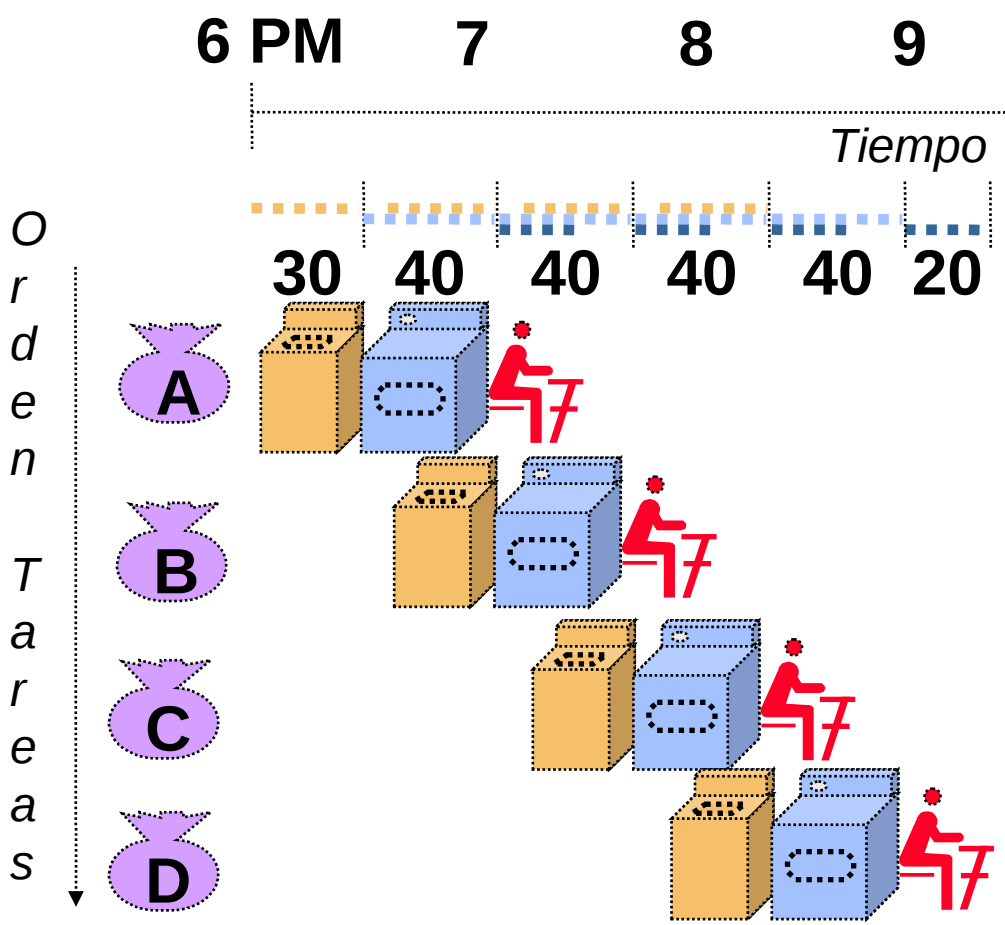


Lavandería Secuencial



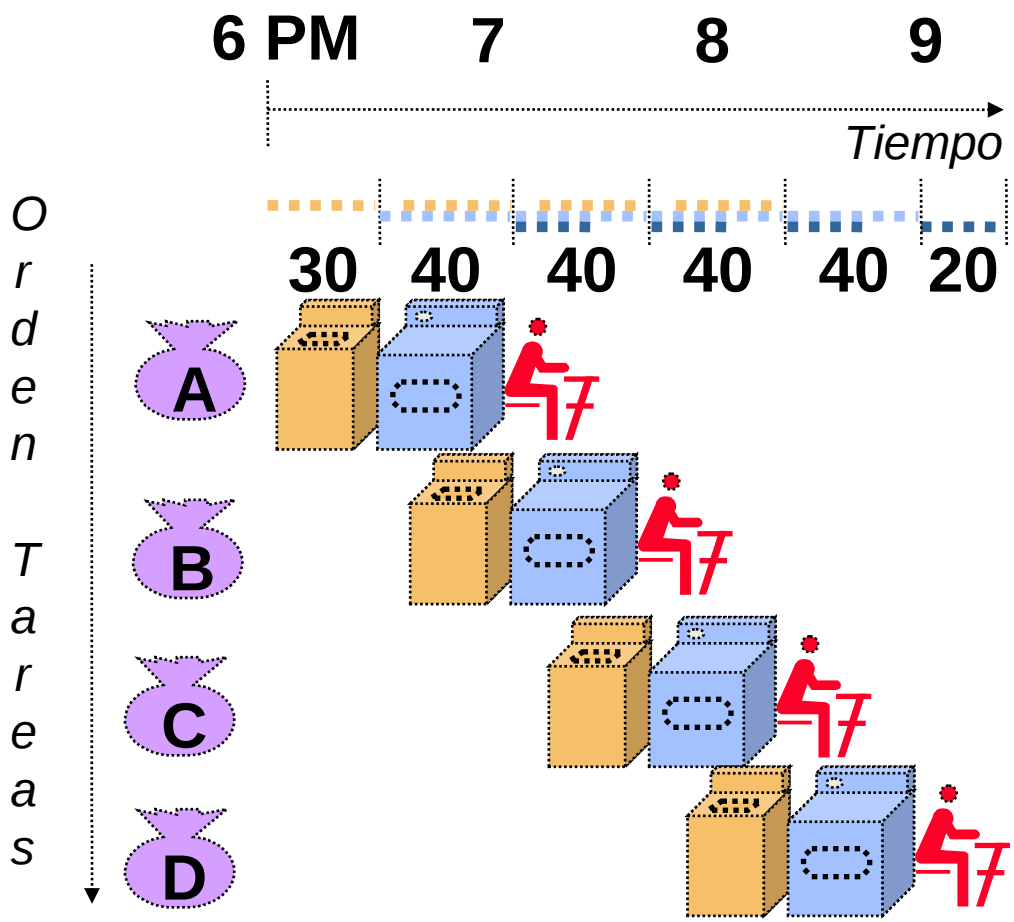
- Lavandería secuencial lleva 6 horas para 4 cargas

Lavandería en Pipeline



- Lavandería en Pipeline lleva 3.5 horas para 4 cargas!

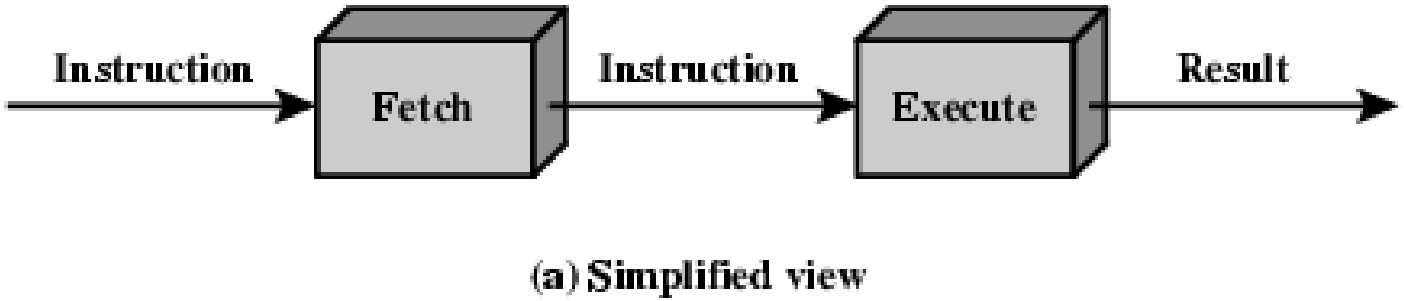
Lecciones del Pipeline



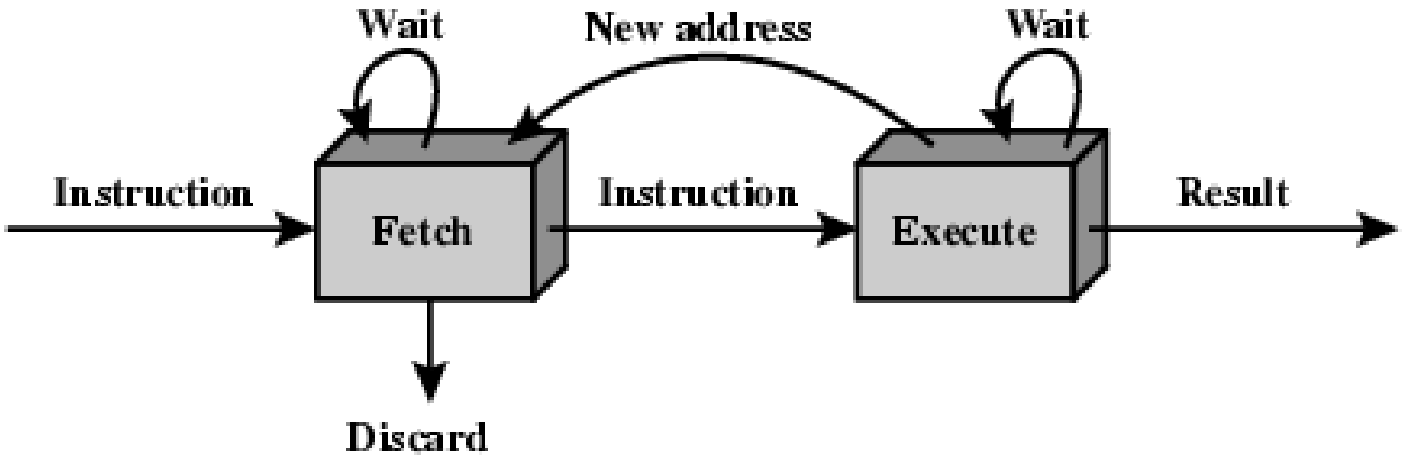
- Pipelining no mejora la latencia de cada tarea, sino el throughput de toda la carga de trabajo
- Velocidad del Pipeline limitada por el paso más lento
- Aceleración potencial = Cantidad de pasos del pipeline
- Tiempo para “llenar” y “vaciar” el pipeline reduce la aceleración

Pipeline primitivo:

Prefetch (o pipeline de dos etapas) (1/2)



(a) Simplified view



(b) Expanded view

Pipeline primitivo:

Prefetch (o pipeline de dos etapas) (2/2)

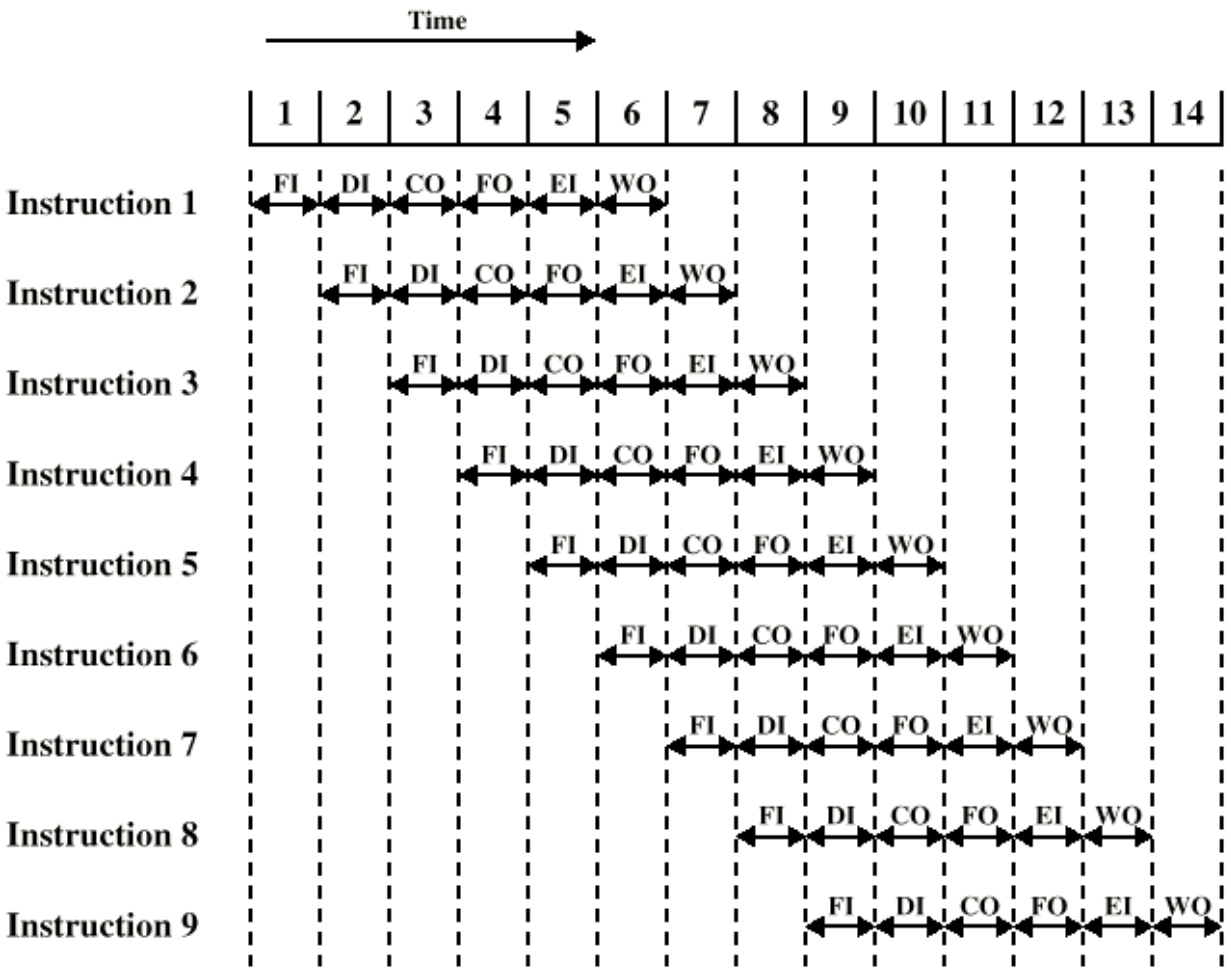
- Fetch accede a memoria principal
- Mientras la etapa de ejecución no accede a memoria, se puede buscar la próxima instrucción
  - Prefetch de instrucciones
- Mejora el rendimiento, pero no lo duplica:
  - Fetch usualmente más corto que la ejecución
  - Saltos condicionales e interrupciones pueden provocar descarte de las instrucciones cargadas mediante prefetch
- Más etapas de pipeline para mejorar performance?

Ejemplo de etapas de pipeline

- Fetch de instrucción (FI)
- Decodificar instrucción (DI)
- Calcular operandos (CO)
- Fetch de operandos (FO)
- Ejecutar instrucción y escribir registro (EI)
- Escribir operando en memoria (WO)

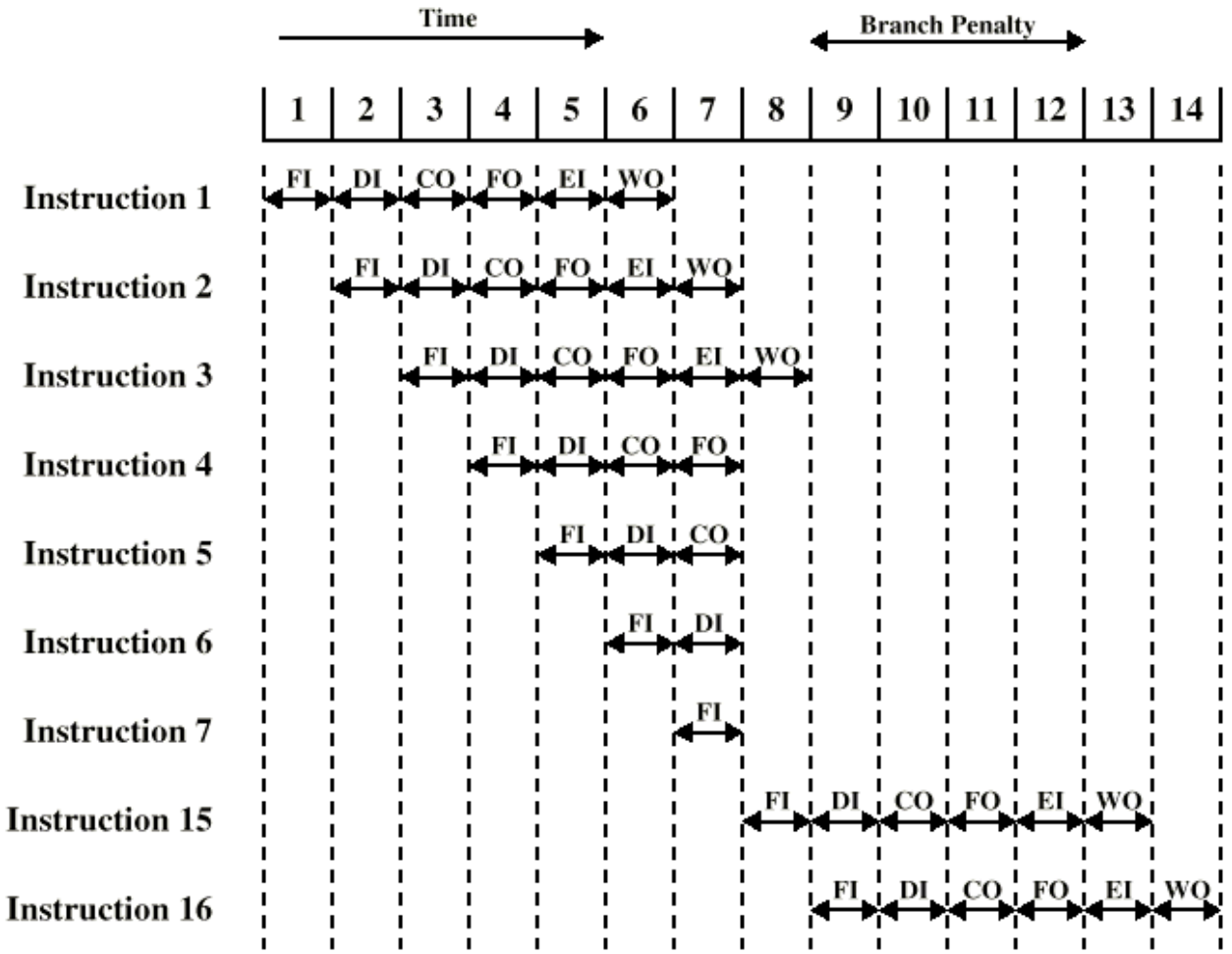
Tiempos del Pipeline

- Etapas de igual duración
- Sin saltos
- Sin conflictos de acceso a memoria y registros



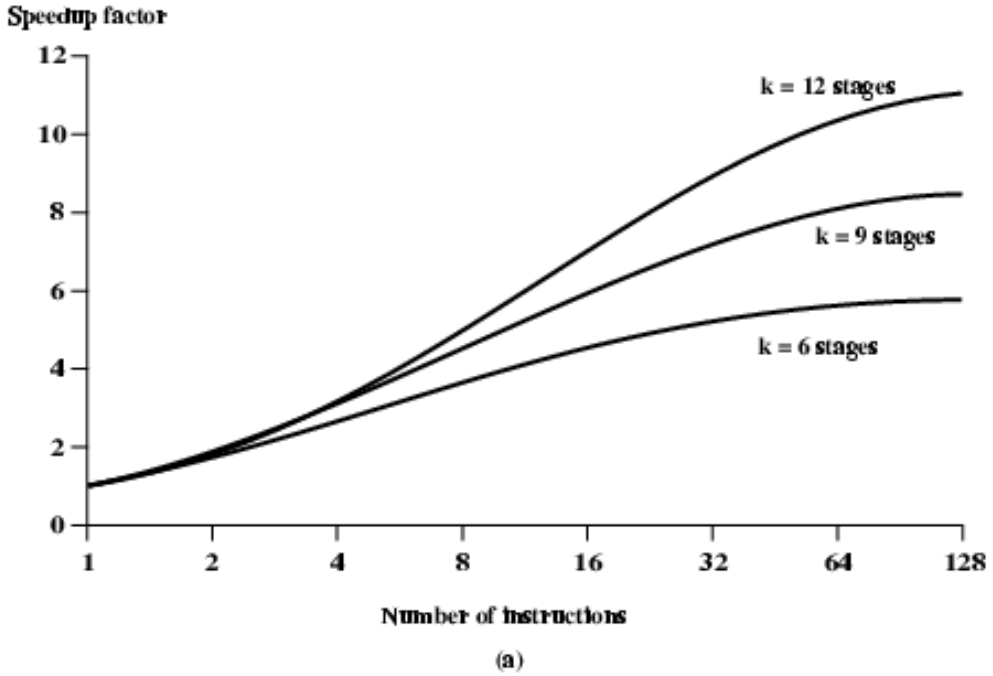
Penalización por saltos

- La I3 es una bifurcación condicional a I15
  - Si se toma el salto hay que vaciar el pipeline



Aceleración del Pipeline

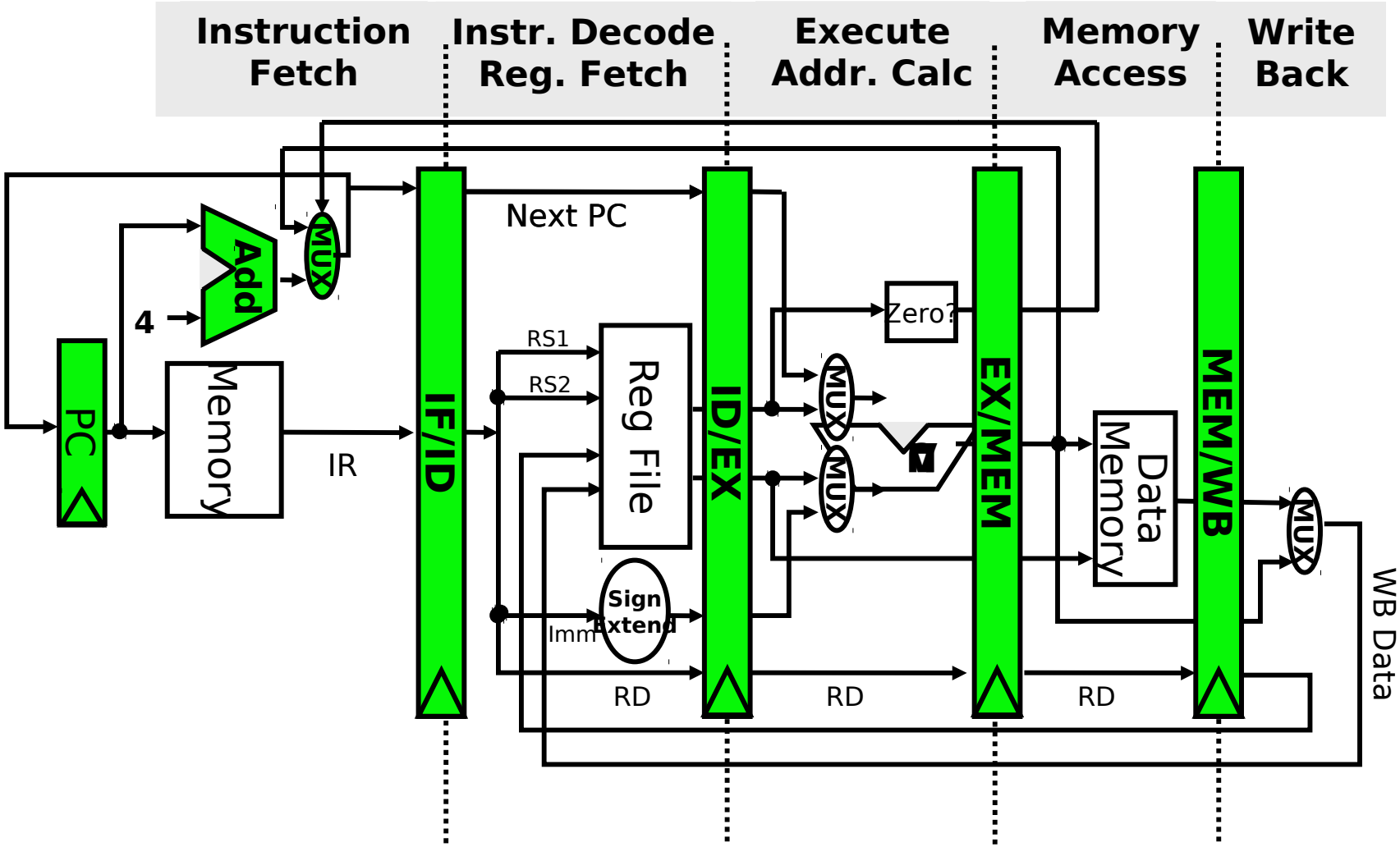
- Pipeline de k etapas, cada una de duración T
- Tiempo de ejecución de n instrucciones
  - Sin pipeline: $T_{tot} = nkT$
  - Con pipeline: $T_{tot} = [n + k - 1]T$
- Aceleración: $S = nk / [n + k - 1]$



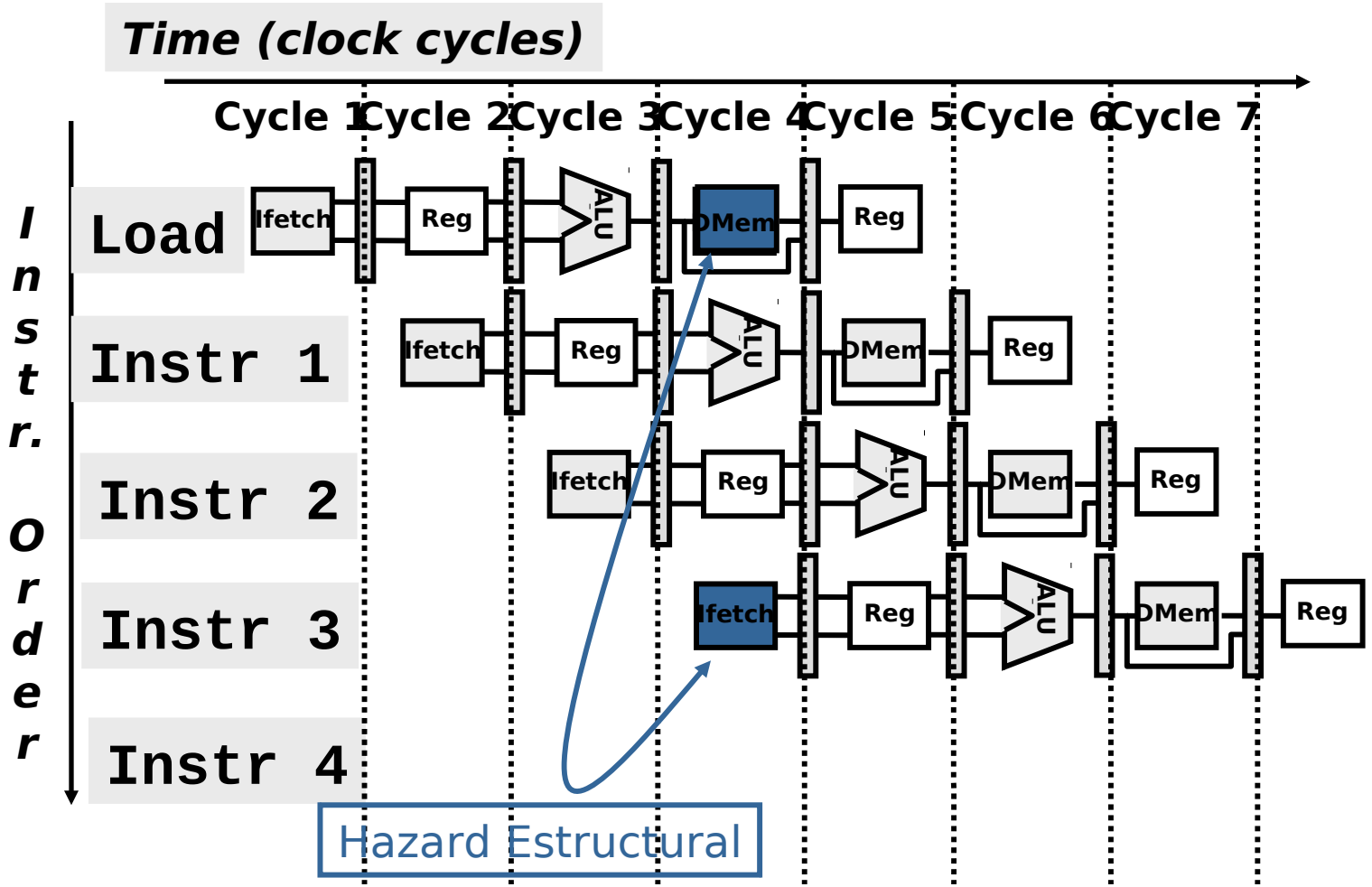
Problemas del pipeline

- Límites del pipeline: los hazards (obstáculos) pueden impedir que se alcance la aceleración teórica
  - Hazards estructurales: conflicto de HW entre dos o más etapas del pipeline para alguna combinación de instrucciones
  - Hazards de datos: dependencias de datos entre instrucciones. Ej., la ejecución de una instrucción depende de un resultado de otra previa, que aún está en el pipeline
  - Hazards de control: causados por instrucciones de salto u otras modificaciones del registro PC

Otro pipeline (Similar MIPS)



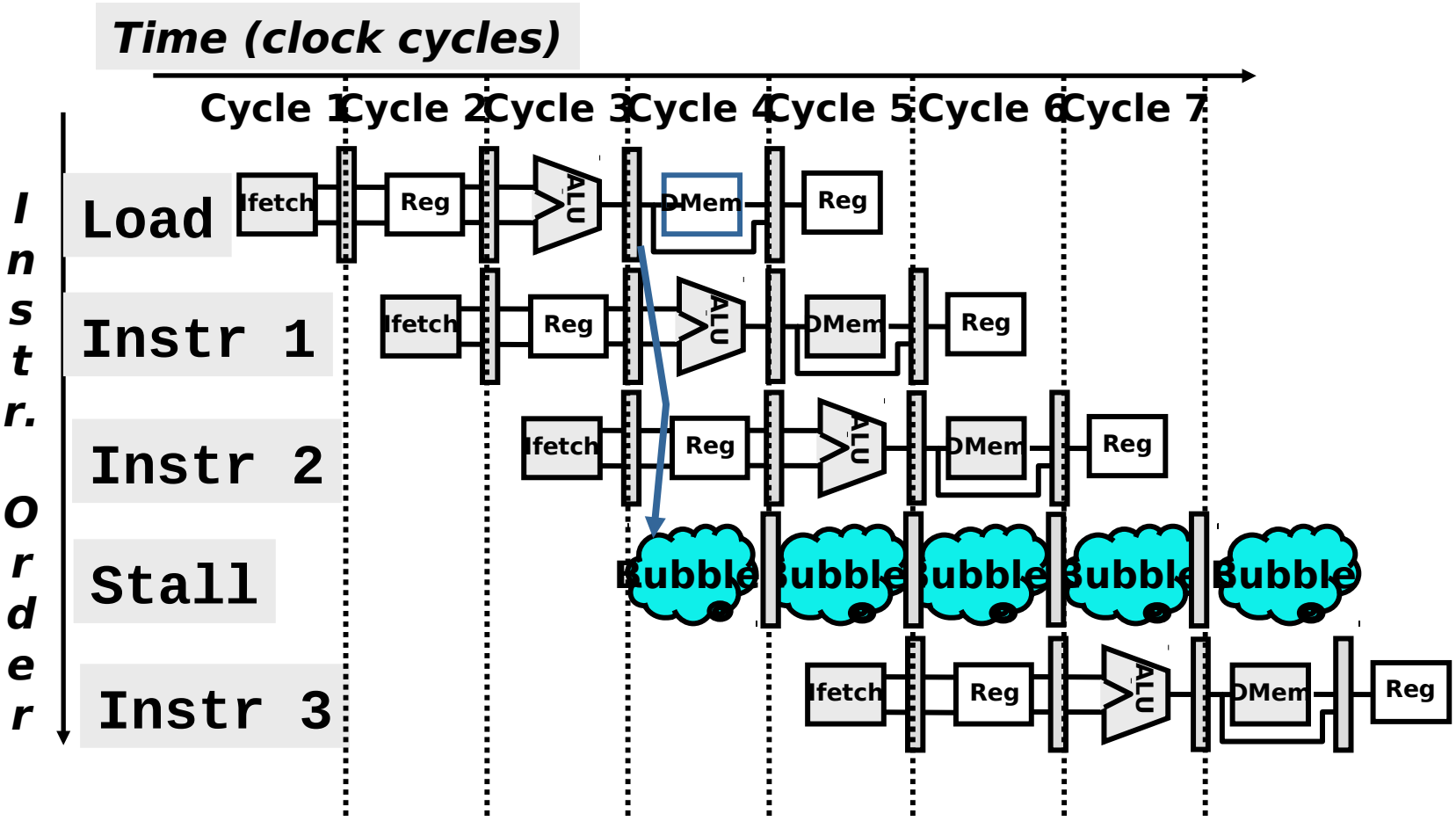
Ejemplo de Hazard Estructural:
Acceso a memoria



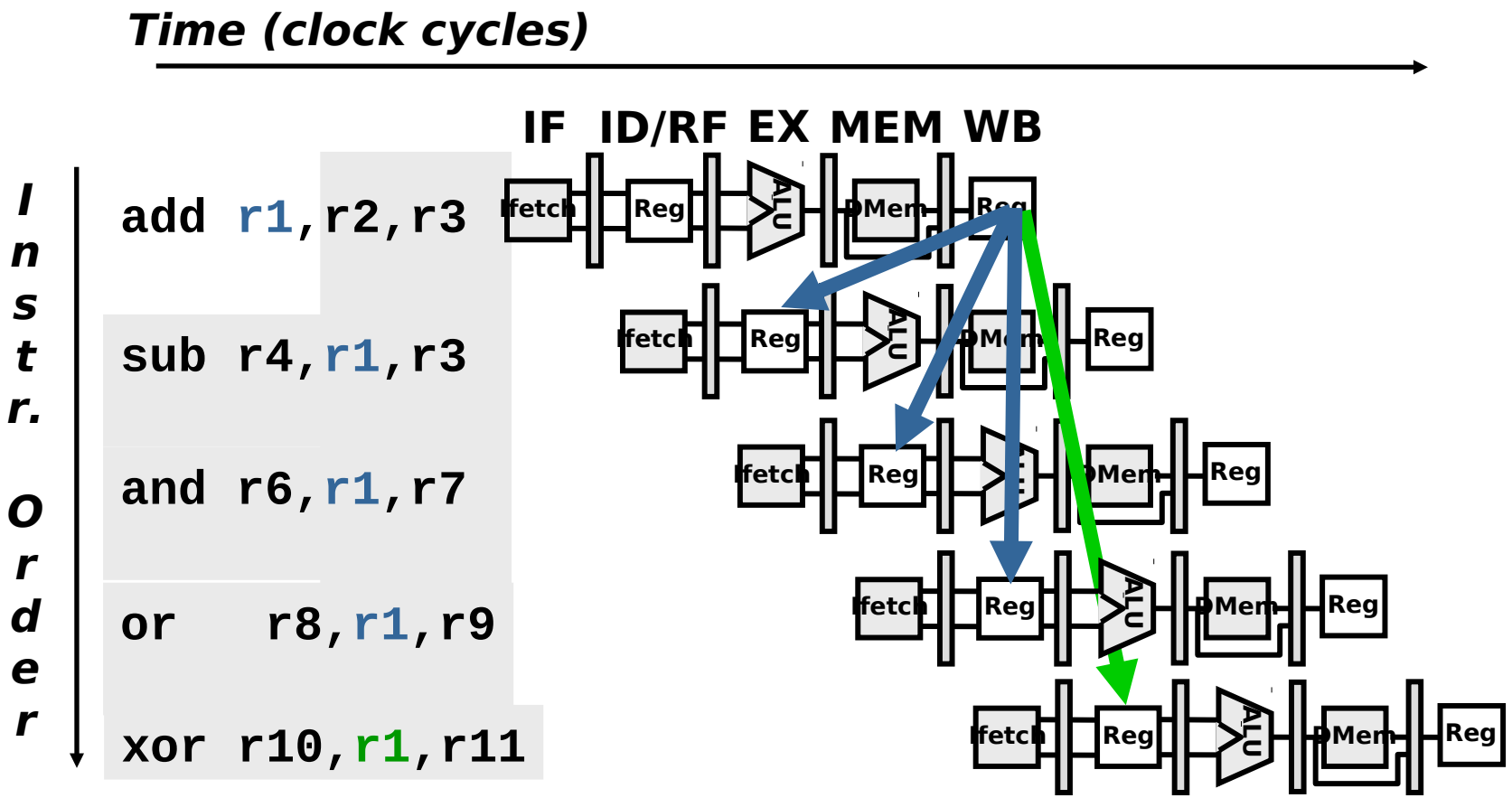
Como resolver un hazard estructural?

- Definición: intento de usar el mismo hardware para dos propósitos diferentes a la vez
- Esperar…
  - Se debe tener mecanismos para detectar el problema y “no hacer nada”
- Agregar más hardware…
  - Por ejemplo, agregar otro puerto de acceso a memoria
- Diseño del conjunto de instrucciones
  - Sabemos qué recursos va a necesitar cada instrucción

Esperar...


Time (clock cycles)
Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
Instr. Order
Load
Instr 1
Instr 2
Stall
Instr 3
Ifetch
Reg
ALU
DMem
Reg
Bubble

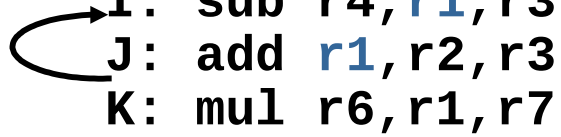
Hazards de Datos



Tipos de Hazards de Datos (1/2)

- Read After Write (RAW)
  Instr$_J$ intenta leer operando antes que Instr$_I$ lo escriba

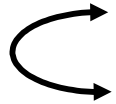
  - "Data Dependence" en nomenclatura del compilador
- Write After Read (WAR)
  Instr$_J$ escribe operando *antes* que la Instr$_I$ lo lea



  - "Anti-dependence" en nomenclatura del compilador
  - Debido al reuso del nombre "r1"
  - Imposible en el pipeline de ejemplo presentado
    - Lecturas en paso 2
    - Escrituras en paso 5

Tipos de Hazards de Datos (2/2)

- Write After Write (WAW)
  Instr$_J$ escribe operando *antes* que lo escriba Instr$_I$



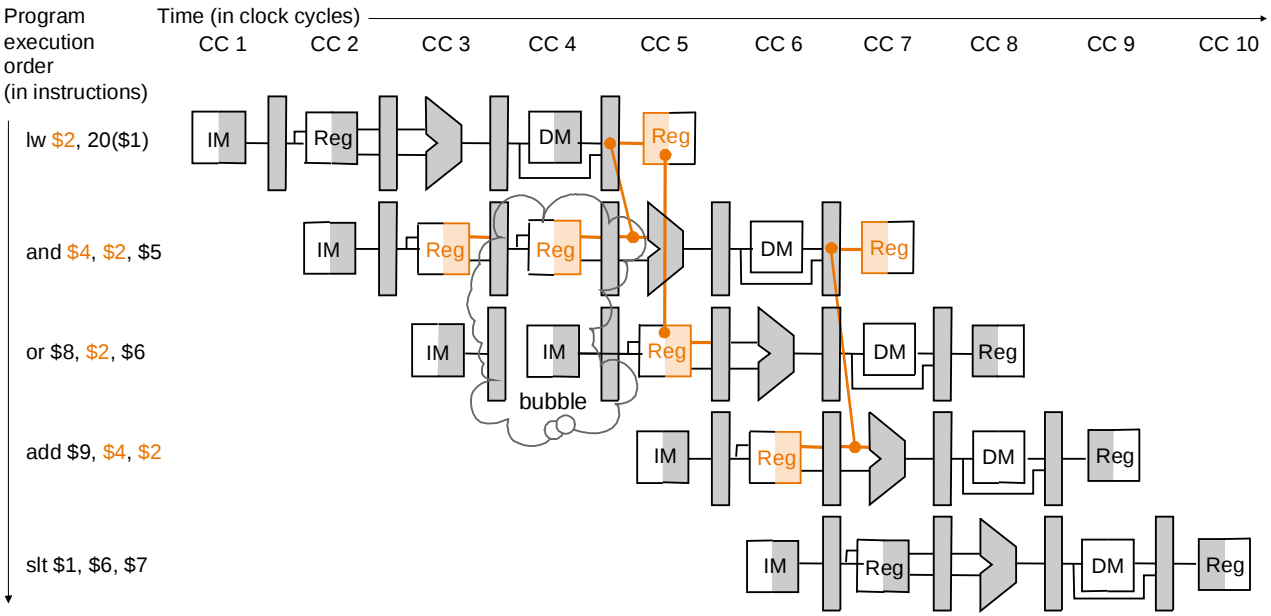
```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

  - "Output dependence" en nomenclatura del compilador
  - También causado por reuso del nombre "r1".
  - Imposible en el pipeline de ejemplo presentado
    - Escrituras en paso 5
- Casos WAR y WAW se dan en pipelines más complicados

- El compilador puede ayudar a evitar hazards de datos

Esperas

- Se para el pipeline manteniendo una instrucción en la misma etapa



Tratamiento de saltos

- Prefetch destino de la bifurcación
  - Además de las instrucciones que siguen el salto
  - Se guarda hasta que se ejecuta el salto
  - Usado en IBM 360
- Multiple Streams
  - Se duplican las etapas iniciales del pipeline
    - Propenso a generar hazards estructurales
  - Más saltos -> más pipelines?
  - Usado en IBM 370
- Delayed branching
  - Ejecutar siempre instrucción siguiente al salto
  - Involucra al programador / compilador
  - Usado en máquinas RISC
- Loop buffer
- Branch prediction

Branch Prediction (1/4)

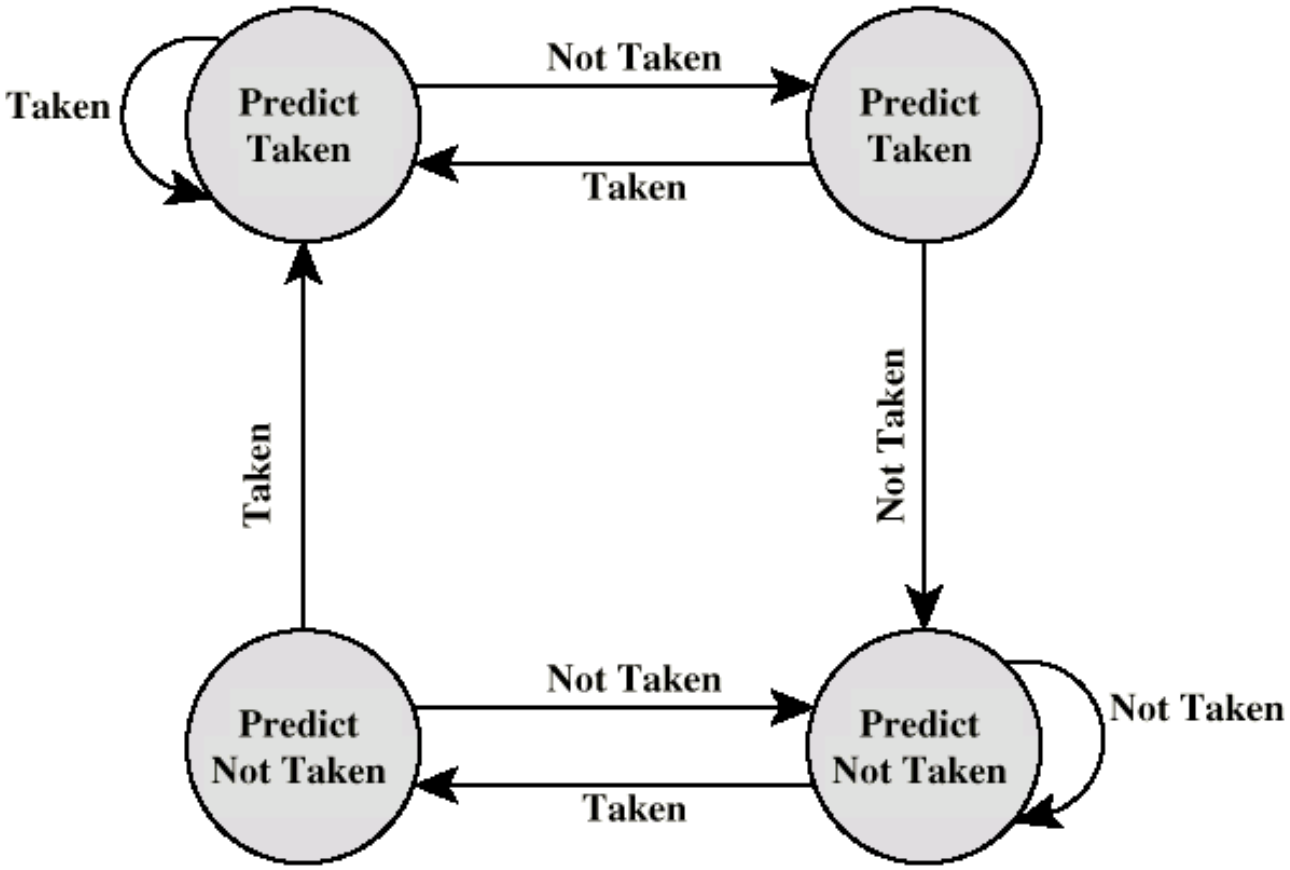
- Predecir nunca se toma el salto
  - Asume que el salto no se ejecuta
  - Siempre se carga la próxima instrucción
  - 68020 & VAX 11/780
  - VAX no hace el prefetch si se da un fallo de página
- Predecir siempre se toma el salto
  - Asume que el salto se ejecuta
  - Siempre se carga el destino del salto
  - Estadísticamente, los saltos condicionales se toman en más de un 50% de los casos, pero también es más probable que generen un fallo de página

Branch Prediction (2/4)

- Predecir por código de operación
  - Para algunas instrucciones de bifurcación, el salto se toma con más frecuencia que para otras
  - Existen reportes de hasta un 75% de aciertos
- Conmutar Taken/Not taken
  - Basado en historia reciente
  - Se necesita memoria para almacenar bits de estado asociado a cada instrucción de salto
  - Bueno para loops

Branch Prediction (3/4) : Diagrama de Estados

- En este ejemplo se usan dos bits de estado
- Debe fallar dos veces para cambiar la predicción
- Si se decide saltar hay que esperar a que la dirección de destino del salto esté disponible para poder hacer el fetch



Preguntas?