# Arquitectura e Ingeniería de Computadores

# Ejecución Superescalar
# ILP – Instruction-Level Paralellism

Lectura de Hennessy-Patterson:

1996 1ª Ed cap.6.8..

2017 6ª Ed cap.3.7- 3.8 -3.9:

Exploiting ILP using Multple issue and Static Scheduling /

Multple issue, Dynamic Scheduling and Speculation

Advanced Techniques for Instruction Delivery and Speculation

# Superescalar – Grado superescalar

- El término **superescalar** referencia a una máquina
  - capaz de ejecutar más de 1 Instrucción Por Ciclo (IPC)
  - de propósito general, o sea, procesa datos convencionales (escalares): enteros y reales (no p. vectorial, por ej).
- Un procesador superescalar (PSE) utiliza diferentes cauces para ejecutar más de una instrucción en paralelo, de modo independiente, utilizando unidades funcionales duplicadas.
- La existencia de varios cauces permite que exista más de un flujo de instrucciones en la máquina, y el grado en el que esto es posible es el grado superescalar.
- Por lo tanto el número de cauces es el grado superescalar.

# Reseña histórica

- Tjaden y Flynn en 1970 presentan el concepto en un trabajo, el cual es el punto de partida
- Se profundiza en esta idea en los años 80 y se implementan los primeros prototipos (IBM y DEC) a la vez que surgen las máquinas RISC, en la década de los 80
- El primer procesador superescalar se comercializa en 1989 y es el Intel 960CA, una máquina RISC.
- Surgen los primeros superescalares CISC en 1993

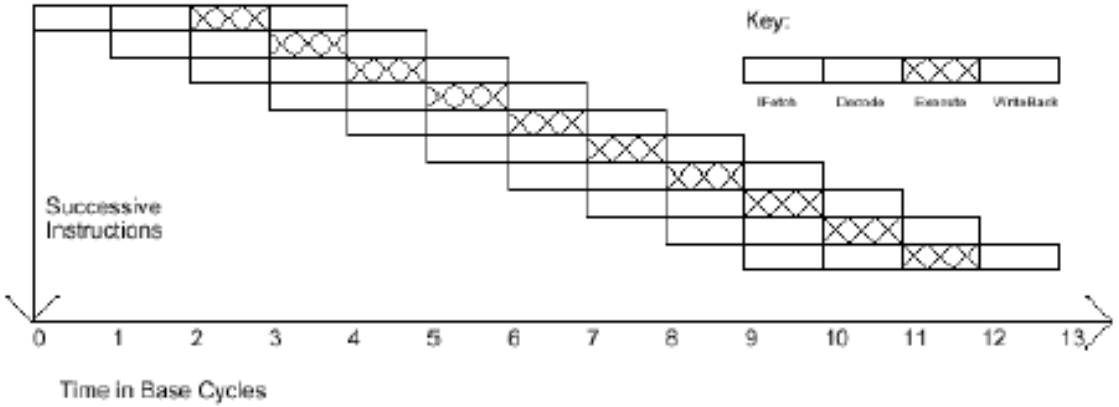
# ILP: Instruction-Level Paralelism

- El procesador superescalar saca provecho del paralelismo inherente en las instrucciones (ILP) de los programas
- La ILP hace referencia a la independencia existente entre instrucciones dentro de una secuencia, de modo que puedan ejecutarse en paralelo, concurrentemente.
  - El que, efectivamente, se ejecuten en paralelo depende de las posibilidades que ofrezcan las máquinas
- El paralelismo de la máquina (a nivel de instrucción) es la capacidad del procesador para explotar el ILP, y está en función de
  - su grado superescalar y de
  - los medios y sofisticación que tenga para localizar instrucciones que sean independientes.

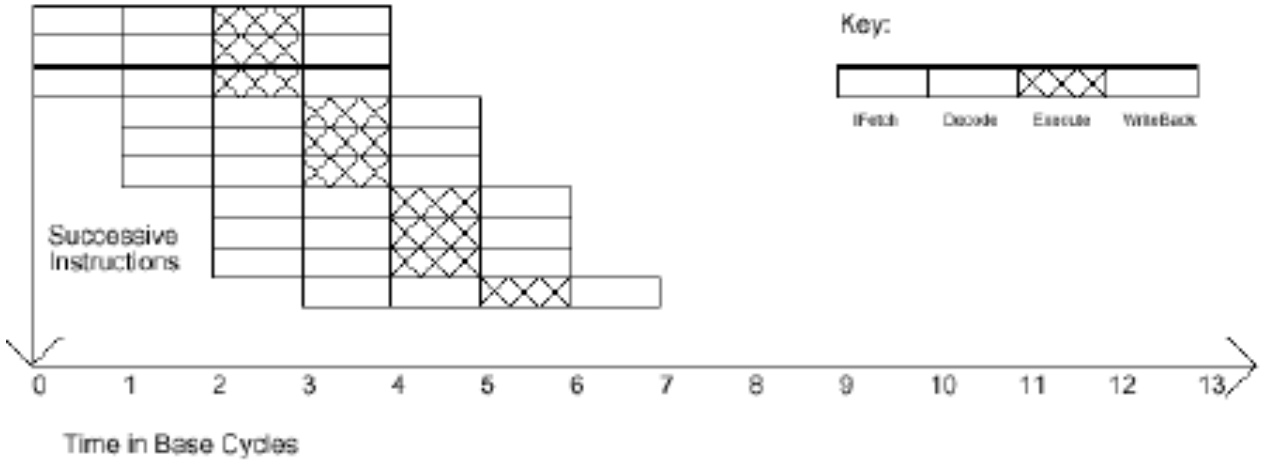
# Captación múltiple

- La existencia de varios cauces implica que el PSE capta (fetch) varias instrucciones simultáneamente y completa más de una instrucción por ciclo idealmente.
- → CPI < 1,  o sea IPC > 1
- Veamos una máquina de profundidad de cauce 4 que se adapta a una ejecución superescalar de grado 3
  - IPC = 3 se completan 3 instrucciones por ciclo
  - Nota:
    - para simplificar, el diagrama mostrado no muestra UFs multiciclo.
    - si intervienen operaciones con CF el diagrama sería menos regular

# Segmentación profundidad 4 -máquina base




# Superescalar grado 3



# ILP: Emisión múltiple

# Formas de explotar el ILP

- Hemos visto técnicas estáticas y dinámicas para eliminar atascos debidos a riesgos de datos y de control en un cauce segmentado para tender a un CPI =1
- Las mismas técnicas pueden aplicarse para los procesadores de emisión múltiple (CPI< 1) o superescalares
- PSEs se clasifican en 3 tipos:
  - PSEs. planificados estáticamente
    - ejecución en orden
  - PSEs. planificados dinámicamente
    - ejecución fuera de orden
  - PSEs. VLIW (Very Long Instruction Word)
    - ejecutan un paquete de instrucciones de tamaño fijo

| Common name | Issue structure | Hazard detection | Scheduling | Distinguishing characteristic | Examples |
|---|---|---|---|---|---|
| Superscalar (static) | Dynamic | Hardware | Static | In-order execution | Mostly in the embedded space: MIPS and ARM, including the Cortex-A53 |
| Superscalar (dynamic) | Dynamic | Hardware | Dynamic | Some out-of-order execution, but no speculation | None at the present |
| Superscalar (speculative) | Dynamic | Hardware | Dynamic with speculation | Out-of-order execution with speculation | Intel Core i3, i5, i7; AMD Phenom; IBM Power 7 |
| VLIW/LIW | Static | Primarily software | Static | All hazards determined and indicated by compiler (often implicitly) | Most examples are in signal processing, such as the TI C6x |
| EPIC | Primarily static | Primarily software | Mostly static | All hazards determined and indicated explicitly by the compiler | Itanium |

**Figure 3.19 The five primary approaches in use for multiple-issue processors and the primary characteristics that distinguish them.** This chapter has focused on the hardware-intensive techniques, which are all some form of superscalar. Appendix H focuses on compiler-based approaches. The EPIC approach, as embodied in the IA-64 architecture, extends many of the concepts of the early VLIW approaches, providing a blend of static and dynamic approaches.

Hennessy-Patterson, 2017

# VLIW: Very Long Instruction Word

- Emiten un número fijo de instrucciones formateadas como una instrucción larga o como un paquete
- Éstas se ejecutan en paralelo sin intervención del HW
- El compilador es el que decide y genera el conjunto
  - las técnicas de desenrollado de bucles y de optimización local y global de los optimizadores son necesarias para esta tecnología
- Existe una gran dependencia de la estructura de los cauces, sus latencias y número de UF’s
  - Limita la posibilidad de migración del código binario entre generaciones de máquinas
-

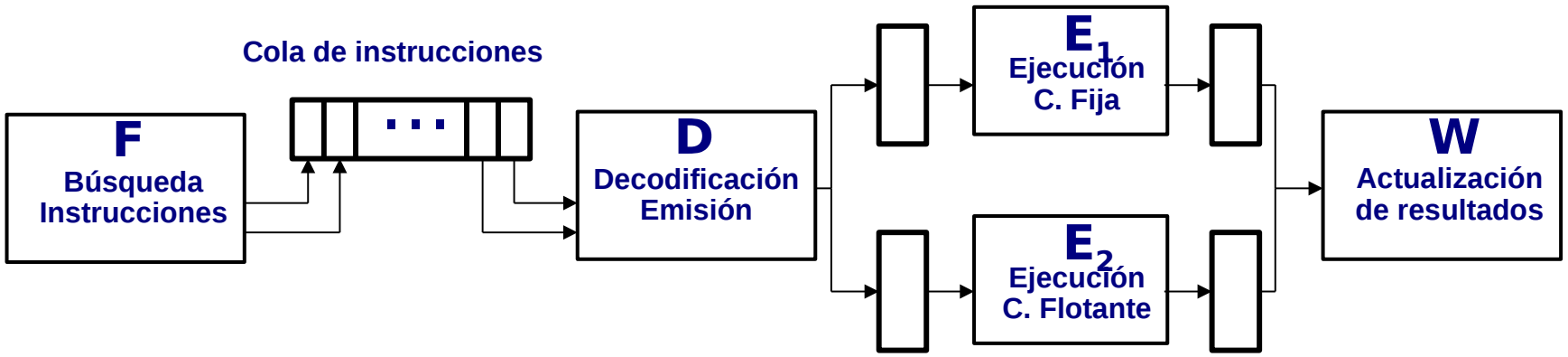
# VLIW

| Dirección | | | | |
|---|---|---|---|---|
| 0 | Instrucción | Instrucción | Instrucción | Instrucción |
| 1 | Instrucción | Instrucción | Instrucción | - |
| 2 | Instrucción | Instrucción | Instrucción | Instrucción |
| 3 | Instrucción | - | - | - |

# PSEs de planificación estática

- Aunque ambos usan planificación estática los procesadores superescalares de planificación estática pueden emitir un número variable de instrucciones por ciclo de reloj en contraste con los VLIW en que es fijo
- Ambos dependen del compilador para definir el paquete de instrucciones a lanzar en paralelo
- Comparten características similares de HW
- En máquinas de propósito general, es difícil mantener una ventana de emisión grande, por lo que
  - las implementaciones se suelen limitar a 2 instrucciones: una CF y otra entero, frecuentemente.
  - Para ventanas de instrucciones mayores, optan por VLIW o planificación dinámica

# Ejemplo de procesador con dos unidades de ejecución

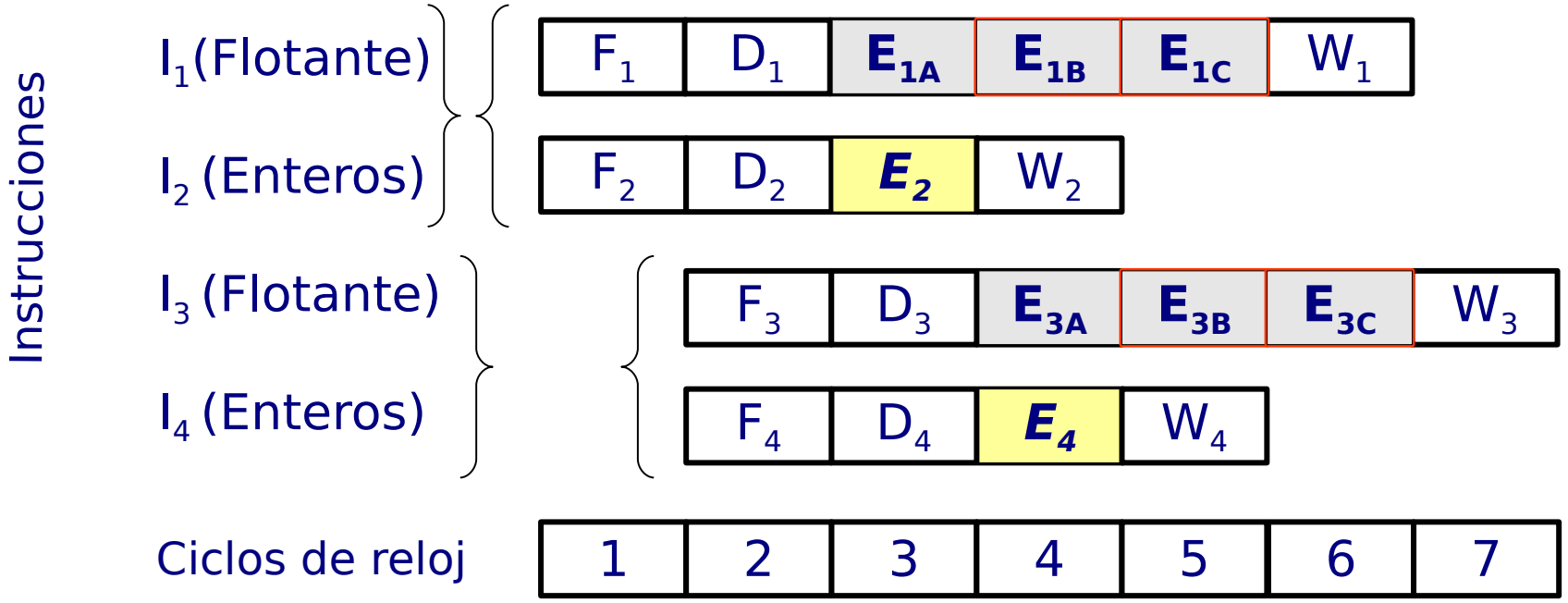


## *Ejemplo de procesador con dos unidades de ejecución*

La unidad de emisión puede tomar 2 instrucciones de la cola y decodificarlas

Si una es de enteros y otra de coma flotante (segmentada) y no existen riesgos → pueden emitirse y ejecutarse a la vez

Responsabilidad del compilador en el aprovechamiento del procesamiento superescalar: entrelazar Enteros y CF.



# PSEs con planificación dinámica

- Supongamos que extendemos Tomasulo + especulación ahora para una emisión múltiple de instrucciones
- La organización básica es la misma solo que es necesario soportar el procesamiento de múltiples instrucciones por ciclo de reloj:
  - extender la lógica de emisión
    - compleja: ir a más de 4 i/ciclo hace que la etapa de emisión sea un cuello de botella: muchas comprobaciones en ¡1 solo ciclo de reloj!
  - extender la lógica de completado a más de una instrucción

# Ejemplo

```
Loop:  ld x2,0(x1)        //x2=array element
       addi x2,x2,1       //increment x2
       sd x2,0(x1)        //store result
       addi x1,x1,8       //increment pointer
       bne x2,x3,Loop     //branch if not last
```

x1,x2… architectural registers

# Ejemplo – dual issue, sin especulación

| Iteration number | Instructions | Issues at clock cycle number | Executes at clock cycle number | Memory access at clock cycle number | Write CDB at clock cycle number | Comment |
|---|---|---|---|---|---|---|
| 1 | ld x2,0(x1) | 1 | 2 | 3 | 4 | First issue |
| 1 | addi x2,x2,1 | 1 | 5 | | 6 | Wait for ld |
| 1 | sd x2,0(x1) | 2 | 3 | 7 | | Wait for addi |
| 1 | addi x1,x1,8 | 2 | 3 | | 4 | Execute directly |
| 1 | bne x2,x3,Loop | 3 | 7 | | | Wait for addi |
| 2 | ld x2,0(x1) | 4 | 8 | 9 | 10 | Wait for bne |
| 2 | addi x2,x2,1 | 4 | 11 | | 12 | Wait for ld |
| 2 | sd x2,0(x1) | 5 | 9 | 13 | | Wait for addi |
| 2 | addi x1,x1,8 | 5 | 8 | | 9 | Wait for bne |
| 2 | bne x2,x3,Loop | 6 | 13 | | | Wait for addi |
| 3 | ld x2,0(x1) | 7 | 14 | 15 | 16 | Wait for bne |
| 3 | addi x2,x2,1 | 7 | 17 | | 18 | Wait for ld |
| 3 | sd x2,0(x1) | 8 | 15 | 19 | | Wait for addi |
| 3 | addi x1,x1,8 | 8 | 14 | | 15 | Wait for bne |
| 3 | bne x2,x3,Loop | 9 | 19 | | | Wait for addi |

# Ejemplo con especulación

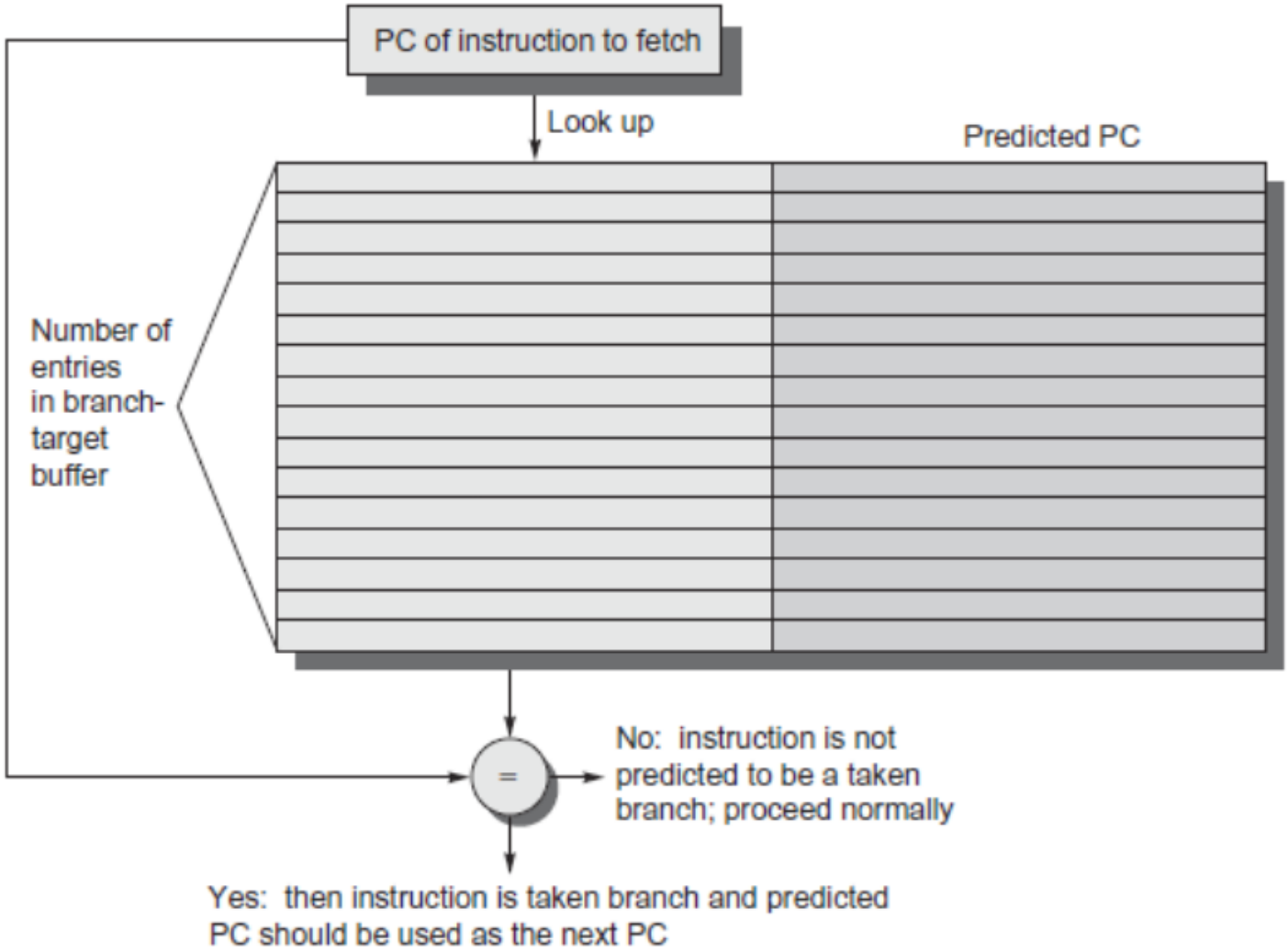
commit in order

| Iteration number | Instructions | Issues at clock number | Executes at clock number | Read access at clock number | Write CDB at clock number | Commits at clock number | Comment |
|---|---|---|---|---|---|---|---|
| 1 | ld x2,0(x1) | 1 | 2 | 3 | 4 | 5 | First issue |
| 1 | addi x2,x2,1 | 1 | 5 | | 6 | 7 | Wait for ld |
| 1 | sd x2,0(x1) | 2 | 3 | | | 7 | Wait for addi |
| 1 | addi x1,x1,8 | 2 | 3 | | 4 | 8 | Commit in order |
| 1 | bne x2,x3,Loop | 3 | 7 | | | 8 | Wait for addi |
| 2 | ld x2,0(x1) | 4 | 5 | 6 | 7 | 9 | No execute delay |
| 2 | addi x2,x2,1 | 4 | 8 | | 9 | 10 | Wait for ld |
| 2 | sd x2,0(x1) | 5 | 6 | | | 10 | Wait for addi |
| 2 | addi x1,x1,8 | 5 | 6 | | 7 | 11 | Commit in order |
| 2 | bne x2,x3,Loop | 6 | 10 | | | 11 | Wait for addi |
| 3 | ld x2,0(x1) | 7 | 8 | 9 | 10 | 12 | Earliest possible |
| 3 | addi x2,x2,1 | 7 | 11 | | 12 | 13 | Wait for ld |
| 3 | sd x2,0(x1) | 8 | 9 | | | 13 | Wait for addi |
| 3 | addi x1,x1,8 | 8 | 9 | | 10 | 14 | Executes earlier |
| 3 | bne x2,x3,Loop | 9 | 13 | | | 14 | Wait for addi |

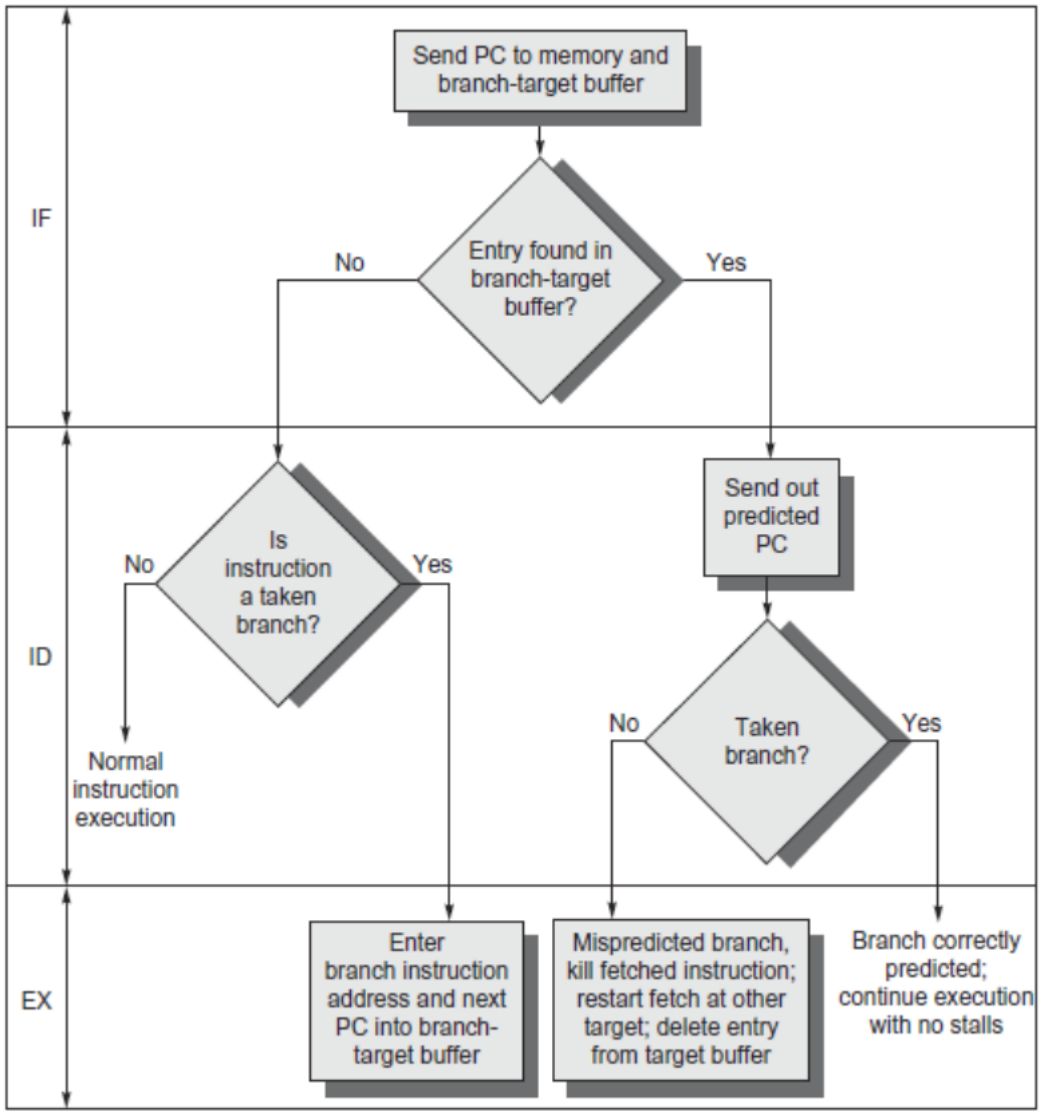
# PSEs dinámicos avanzados

- Los PSEs modernos también incluyen un buffer de pre-fetch ya que es necesario buscar ~ 4 i/ciclo
  - Buses más “anchos” para el tráfico con memoria
- Manejo de saltos más sofisticado
  - Se identifican las instrucciones de salto en el buffer de pre-fetch y se calculan las direcciones de salto
  - Estas se guardan en una caché especial llamada “branch target buffer”
  - Si el salto se toma, se obtiene la dirección de esta caché directamente, sin ciclos de penalización
- Alternativa a ROB: renombrado de registros
  - Registros físicos reemplazan a los reg. de la arquitectura
  - contienen los valores temporales
  - simplifica la etapa de commit

# Branch Target Buffer



# Branch Target Buffer



# Plegado de Salto

- Es una técnica de optimización
- “Branch folding”
- Se incluye la instrucción del destino del salto en el Branch Target Buffer para poder gestionar mejor los tiempos de decodificación asociados a buffers grandes

  (Búsquedas en cachés asociativas mayores)

# Renombrado de Registros

Ej: Si el destino de I2 es R5, TW2 es considerado como R5 durante ciclos 5 y 6 (adelantamientos etc) y solo en estos ciclos.

El registro temporal $TW_i$ se usa sólo por las instrucciones posteriores a i.

**Uso de registros temporales**