Arquitectura de Computadoras para Ingeniería

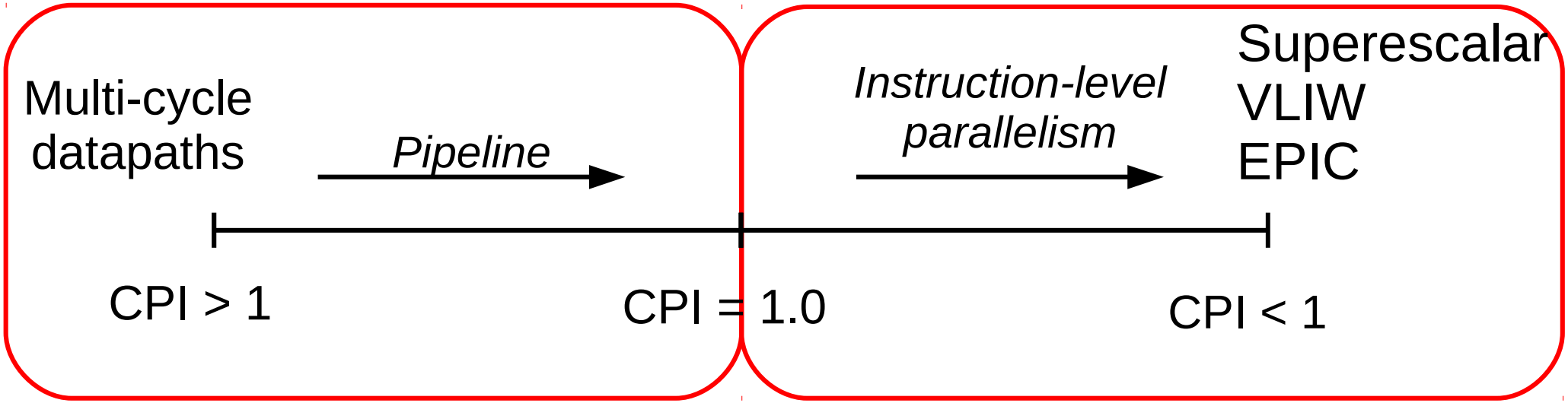
(Cód. 7526)
1° Cuatrimestre 2016

Dra. Dana K. Urribarri
DCIC - UNS

Instruction-level parallelism

Instruction-level parallelism

- El objetivo es aprovechar al máximo el paralelismo a nivel de instrucciones para ejecutar múltiples instrucciones por ciclo.



Mejorar el rendimiento

- El tiempo de ejecución de un programa es

$$EX_{CPU} = \#\,Instrucciones \times CPI \times período\,del\,reloj$$

- Para reducir EX_{CPU}
  - Cambiar la cantidad de instrucciones
  - Reducir el CPI
  - Reducir el período del reloj

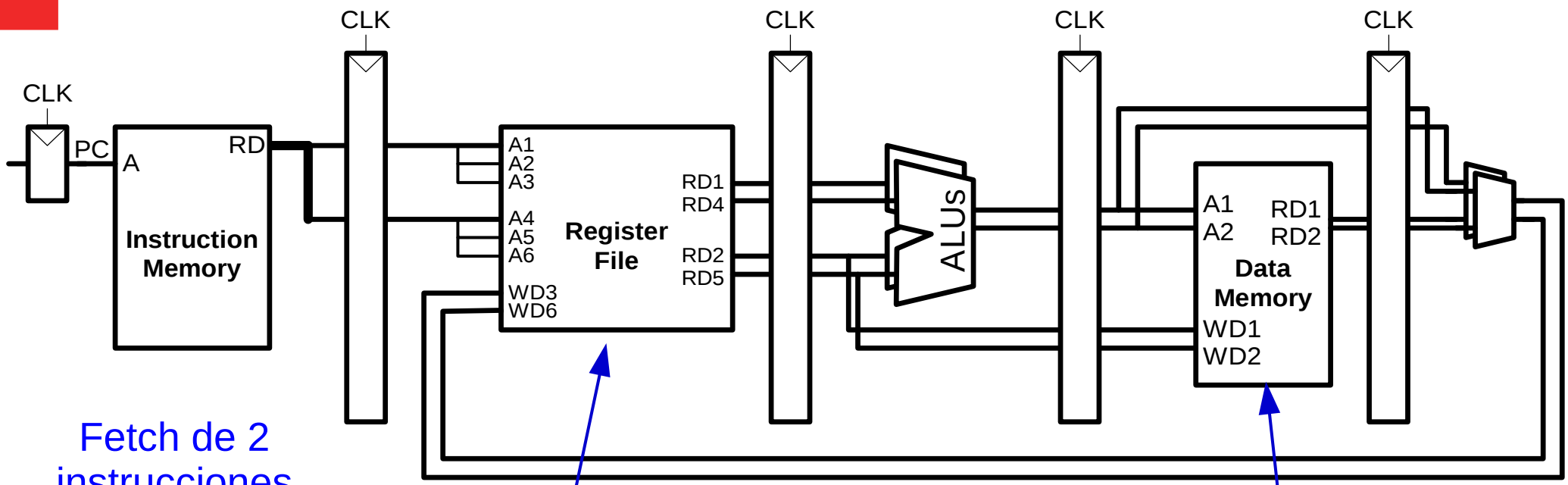
Queremos mejorar la microarquitectura, no cambiar la arquitectura.

Reducir CPI

- Reducir el CPI implica incrementar el IPC.
- Modificar la estructura del pipeline para permitir que más de una instrucción ejecute a la vez en la misma etapa.

- Procesador superescalar
  - Incrementa la cantidad de unidades funcionales.
  - Se *ensancha* el pipeline.

Reducir CPI



Fetch de 2 instrucciones

Banco de registros con 4 puertos de lectura y dos de escritura

Dos ALUs que operan en paralelo

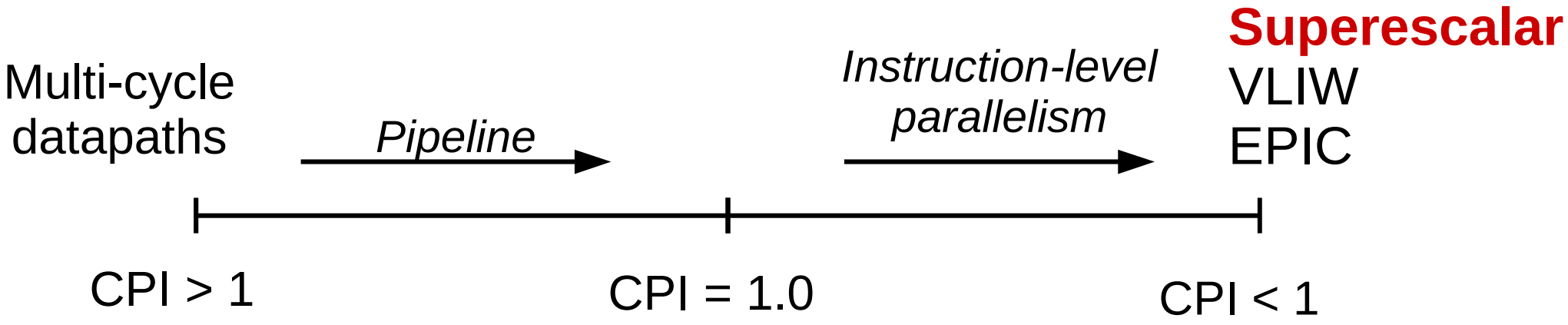
Memoria de datos que acepta dos accesos en simultáneo

$CPI_{ideal} = 0.5$

Reducir el período del reloj

- Reducir el período del reloj implica incrementar la frecuencia.
- Para esto, cada etapa del pipeline debe realizar menos trabajo.
- Las etapas se dividen en nuevas etapas y el pipeline se vuelve más *profundo*.

Instruction-level parallelism



Superescalar + Profundidad del pipeline

- Dividimos al pipeline en
  - Front-end → Fetch y Decode
  - Back-end → Execute, Memory y WriteBack

- Superescalar de grado *m* (*m*-way)
  - Front-end: Fetch y decode de *m* instrucciones a la vez
  - Back-end: Execute, memory y writeback de *m* instrucciones a la vez.

Categorías de Superescalar

- Las instrucciones se procesan en el *front-end* en el orden definido en el programa.
- Procesadores superescalares
  - En orden o estáticos
  - Fuera de orden o dinámicos

  se diferencian en el orden en que las instrucciones comienzan el procesamiento *back-end.*

Superescalar en orden o estáticos

- Las instrucciones dejan el *front-end* en el orden definido en el programa.
- Todas las dependencias de datos se resuelven antes de que la instrucción comience el procesamiento *back-end* (o podrán resolverse más adelante por *forwarding*).

Superescalar fuera de orden o dinámico

- Las instrucciones pueden terminar el procesamiento *front-end* y comenzar el *back-end* antes que instrucciones previas del orden establecido por el programa.
- Las instrucciones comienzan a ejecutar ni bien tiene los operandos disponibles.
- La etapa ~~WB~~ Commit debe respetar la semántica del programa.

  Los resultados deben almacenarse en el orden especificado por el programa.

Microarquitecturas superescalares

- En orden
  - CDC 6600 → Scoreboard
  - DEC Alpha 21164

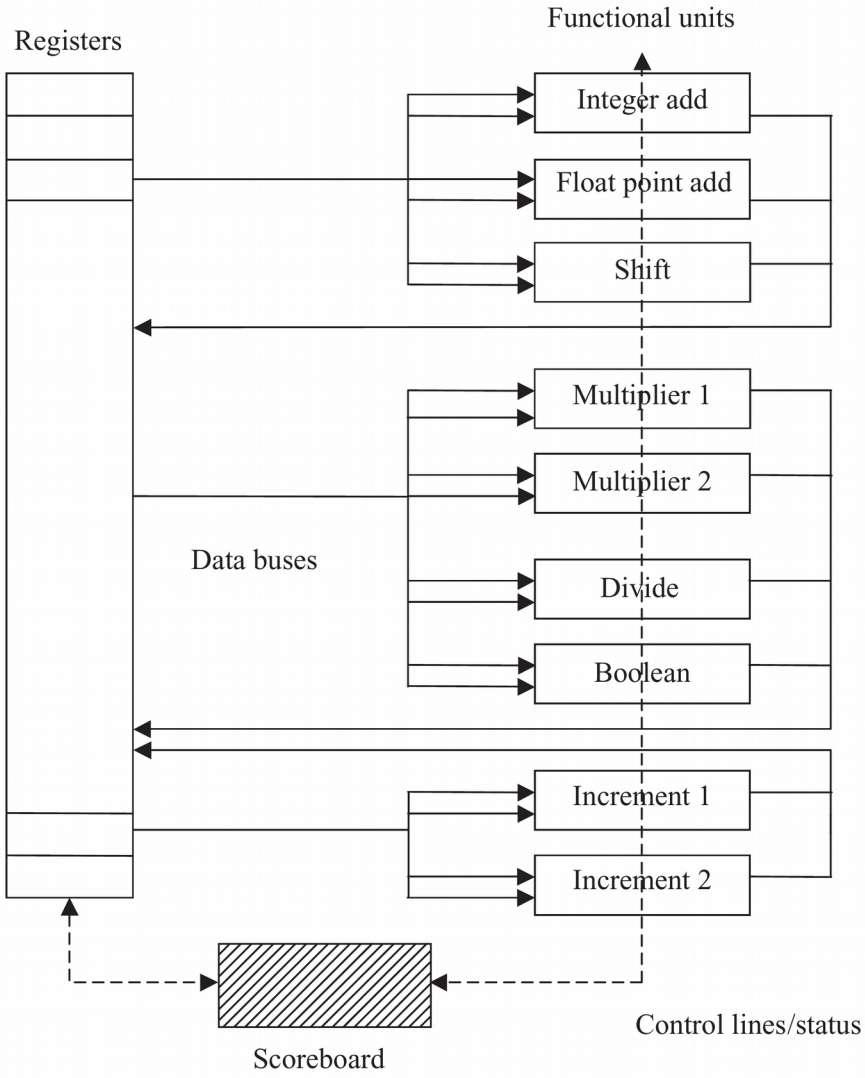
Scoreboard (CDC 6600)

- 1964: CDC 6600
  - Incorporó (entre otras cosas) unidades funcionales (UF) separadas.
  - Las UF no estaban en pipeline.
  - Operaban concurrentemente coordinadas por un *Scoreboard* (unidad de control).

Primera supercomputadora (1964)

Scoreboard (CDC 6600)

- La CDC 6600 tiene 10 unidades funcionales
  - Un *shifter*, sumador de enteros largos y sumador de punto flotante
  - Dos multiplicadores de punto flotante
  - Divisor en punto flotante y una unidad lógica
  - Dos incrementadores para calcular direcciones
  - Una unidad de saltos



Scoreboard (CDC 6600)

- Una vez que se hizo el *fetch* y el *decode* de la instrucción, el *Scoreboard* controla los siguientes pasos:

Completan el Decode

1) *Issue*: Si hay una UF libre y no hay conflictos WAW, se reserva la UF y la instrucción pasa a la siguiente etapa. Si no se cumple alguna condición, se frena la instrucción y las siguientes.

2) *Dispatch*: El *Scoreboard* controla que los operandos estén disponibles (ninguna instrucción previa va a escribirlo). Una vez que están disponibles (evita conflictos RAW) le indica a la UF que lea los operandos y comience la ejecución.

Scoreboard (CDC 6600)

3) *Execution*: La UF lee los operandos y ejecuta (uno o más ciclos, dependiendo de la latencia de la UF). Cuando la UF termina le avisa al *scoreboard* que está en condiciones de escribir el resultado.

4) *Write result*: Antes de escribir el *scoreboard* controla por conflictos WAR. Si existe uno, se frena la unidad hasta que se resuelvan.
Las instrucciones en progreso ya leyeron sus operandos y por lo tanto no ocasionan conflictos WAR.

Scoreboard (CDC 6600)

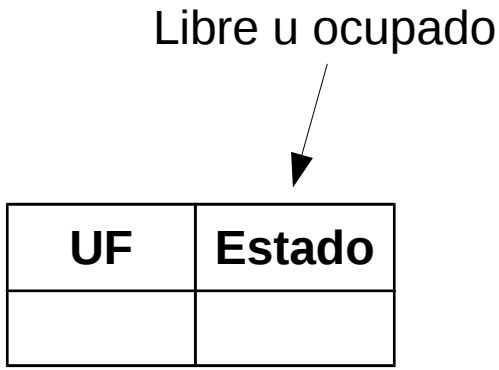
- El *scoreboard* necesita tener información de:
  - Para cada UF, si está libre u ocupada
  - Para cada instrucción
    - Nombre del registro resultado y de los fuentes
    - Nombre de las unidades que produzcan valores para los registros fuentes (puede no haber ninguna).
    - Flags que indiquen si los registros fuentes están disponibles.
    - El estado de la instrucción: en cuál de las 4 etapas está.
  - Por cada registros resultado, el nombre de la UF que produce su resultado (es redundante, pero facilita el hw)

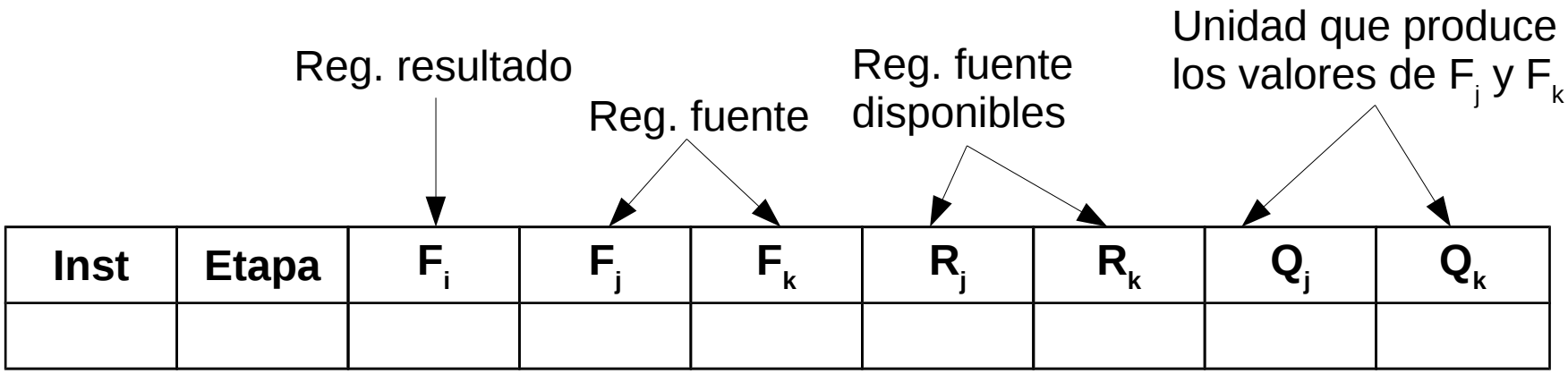
Scoreboard (CDC 6600)

Etapa	Condición	Acciones
Issue	Si alguna instrucción previa está esperando en esta etapa o la UF está ocupada o hay conflicto WAW, entonces *Stall* de la instrucción y de las siguientes.	Marcar la UF como ocupada. Actualizar los datos del Scoreboard.
Dispatch	Si alguno de los registros fuente no está disponibles, entonces *Stall* de la instrucción.	Enviar los datos a la UF.
Execute	Al final de la ejecución, pedir permiso para escribir el resultado.	
Write result	Si hay una instrucción previa que todavía no leyó los operandos y uno de esos operandos es el destino, entonces *Stall* de la instrucción.	Marcar la UF como libre. Marcar el registro destino como disponible.

Scoreboard (CDC 6600)

- 2 multiplicadores de latencia 6
- 1 sumador de latencia 1
  - *i1*: R4 ← R0 * R2
  - *i2*: R6 ← R4 * R8
  - *i3*: R8 ← R2 + R12
  - *i4*: R4 ← R14 + R16





Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Estado inicial

UF	Estado
mult1	libre
mult2	libre
add	libre

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 1

UF	Estado
mult1	ocupado
mult2	libre
add	libre

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Issue	4	0	2	Si	Si		

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 2

UF	Estado
mult1	ocupado
mult2	ocupado
add	libre

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Dispatch	4	0	2	Si	Si		
2	Issue	6	4	8	No	Si	mult1	

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 3

UF	Estado
mult1	ocupado
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Execute	4	0	2	Si	Si		
2	Dispatch-Stall	6	4	8	No	Si	mult1	
3	Issue	8	2	12	Si	Si		

No puede completar el *Dispatch* por conflicto RAW. R4 todavía no está disponible.

Va a estarlo recién en el ciclo 9.

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 4

UF	Estado
mult1	ocupado
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Execute	4	0	2	Si	Si		
2	Dispatch-Stall	6	4	8	No	Si	mult1	
3	Dispatch	8	2	12	Si	Si		
4	Issue-Stall							

No puede completar el *Issue* porque el sumador está ocupado y hay conflicto WAW de R4 con la instrucción 1.
→*Stall*

Scoreboard (CDC 6600)

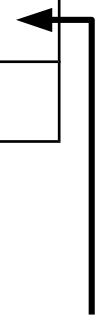
i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 5

UF	Estado
mult1	ocupado
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Execute	4	0	2	Si	Si		
2	Dispatch-Stall	6	4	8	No	Si	mult1	
3	Execute	8	2	12	Si	Si		
4	Issue-Stall							



Terminó de ejecutar y pide permiso para escribir.

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 6 y 7

UF	Estado
mult1	ocupado
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Execute	4	0	2	Si	Si		
2	Dispatch-Stall	6	4	8	No	Si	mult1	
3	Write result - Stall	8	2	12	Si	Si		
4	Issue-Stall							

Hay conflicto WAR de R8 con la instrucción 2.
→ *Stall*

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 8

UF	Estado
mult1	ocupado
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Execute	4	0	2	Si	Si		
2	Dispatch-Stall	6	4	8	No	Si	mult1	
3	Write result - Stall	8	2	12	Si	Si		
4	Issue-Stall							

1: Termina la ejecución y pide permiso para escribir.

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 9

UF	Estado
mult1	libre
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	Write Result	4	0	2	Si	Si		
2	Dispatch-Stall	6	4	8	Si	Si	mult1	
3	Write result - Stall	8	2	12	Si	Si		
4	Issue-Stall							

1: Libera la UF y escribe el resultado.

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 10

UF	Estado
mult1	libre
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Dispatch	6	4	8	Si	Si	mult1	
3	Write result - Stall	8	2	12	Si	Si		
4	Issue-Stall							

2: Puede leer los operandos y completar el *Dispatch*.

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 11...

UF	Estado
mult1	libre
mult2	ocupado
add	libre

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Execute	6	4	8	Si	Si	mult1	
3	Write Result	8	2	12	Si	Si		
4	Issue-Stall							

2: comienza a ejecutar. Va a terminar en el ciclo 16.

3: Ya no hay conflicto WAR con *i2*. Puede escribir el resultado y liberar el sumador.

4: Ya no hay conflicto con *i1*, pero el sumador se libera recién al final del ciclo.

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 12

UF	Estado
mult1	libre
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Execute	6	4	8	Si	Si		
3	------							
4	Issue	4	14	16	Si	Si		

4: Completa el Issue porque se libera el sumador

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 13

UF	Estado
mult1	libre
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Execute	6	4	8	Si	Si		
3	------							
4	Dispatch	4	14	16	Si	Si		

4: Puede leer los operandos y termina el *Dispatch*

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 14

UF	Estado
mult1	libre
mult2	ocupado
add	ocupado

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Execute	6	4	8	Si	Si		
3	------							
4	Execute	4	14	16	Si	Si		

4: Comienza con el *Execute* que es de un ciclo. Al finalizar pide permiso para escribir el resultado

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 15

UF	Estado
mult1	libre
mult2	ocupado
add	libre

Reg. Fuente disponibles

Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Execute	6	4	8	Si	Si		
3	------							
4	Write Result	4	14	16	Si	Si		

4: Como no hay conflictos (i2 ya terminó el *Dispatch*) escribe el resultado

Scoreboard (CDC 6600)

i1: R4 ← R0 * R2
i2: R6 ← R4 * R8
i3: R8 ← R2 + R12
i4: R4 ← R14 + R16

Ciclo 16

UF	Estado
mult1	libre
mult2	ocupado
add	libre

Reg. Fuente disponibles

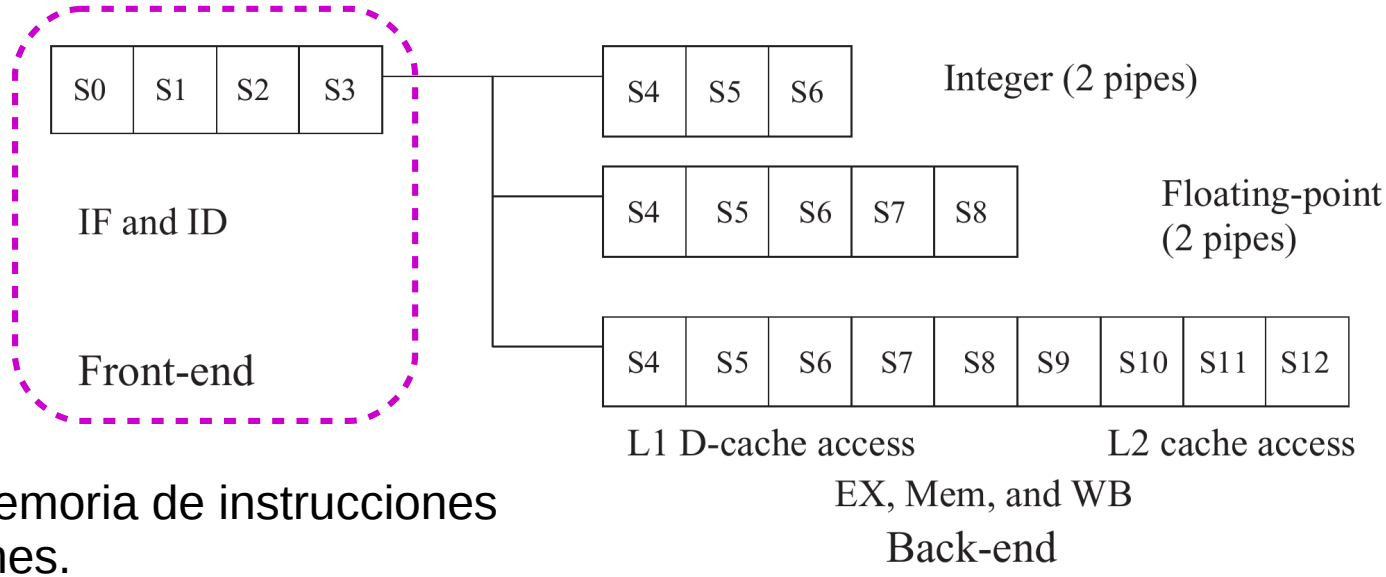
Inst	Etapa	F_i	F_j	F_k	R_j	R_k	Q_j	Q_k
1	------							
2	Execute	6	4	8	Si	Si		
3	------							
4	------							

2: Último ciclo de ejecución de *i2*. Pide permiso para escribir.

DEC Alpha 21164

- *Digital Equipment Corporation* 1994 – RISC
- En orden 4-way (pipeline de 4 instrucciones en paralelo)
- Tiene 4 unidades funcionales:
  - 2 UF en pipeline para operaciones aritméticas de enteros, operaciones lógicas y cálculos de direcciones.
    - 1 UF tiene un *shifter* y un multiplicador (parcialmente en pipeline, la latencia depende de la longitud de los operandos)
    - 1 UF para los branches.
  - 2 UF en pipeline para operaciones en punto flotante
    - 1 UF está dedicada a multiplicación
    - 1 UF realiza las demás operaciones incluyendo división (que no es no en pipeline ⇒ mayor latencia)

DEC Alpha 21164



Front-end (4 etapas)

- ***S0***: Accede a la memoria de instrucciones y trae 4 instrucciones.
- ***S1***: Realiza la predicción del *branch* y calcula la dirección de salto
- ***S2***: Dirige las instrucciones a la UF apropiada. Resuelve los conflictos estructurales.
- ***S3***: Resuelve conflictos de datos. Planifica *forwarding* y *stalls* por dependencias RAW y WAW.

La unidad principal de control es un *Scoreboard* y S3 se corresponde con la etapa *Issue*.

El *Scoreboard* controla el estado de los registros, el progreso de las instrucciones en el back-end y envía las señales de *forwarding* y escritura de resultados.

Microarquitecturas superescalares

- Fuera de orden
  - *Renombramiento de registros*
  - *Buffer de reordenamiento*
  - *Estaciones de reservación/Ventana de instrucciones*
  - IBM System 360/91 → Algoritmo de Tomasulo

Superescalar fuera de orden

- Las tareas del *front-end*
  - Fetch de múltiples instrucciones
  - Predicción de *branchs*
  - Decodificación de las instrucciones

  es independiente del procesamiento *back-end*.
- Procesadores en orden y fuera de orden difieren en cuándo las instrucciones dejan el *front-end* y cómo se procesan en el *back-end*.

Renombramiento de registros

- Los conflictos RAW son dependencias procedimentales.
  - Una dependencia RAW no debería frenar instrucciones siguientes que no dependan del valor que genera esa instrucción.

- Los conflictos WAR y WAW existen porque los recursos (registros) son escasos.
  - Son dependencias de nombre

Renombramiento de registros

- Registros lógicos o de arquitectura
  - Registros definidos por el ISA
- Registros físicos
  - Total de registros reales de almacenamiento

- Los registros lógicos en una instrucción se mapean a los registros físicos en la última etapa del *front-end*.
  - Los registros fuentes se reemplazan por los registros físicos a los que fueron mapeados
  - El registro destino se mapea a un registro físico sin usar.

Renombramiento de registros

Etapa de renombramiento

- i_x: $R_i \leftarrow R_j$ *op* R_k
  - $R_b \leftarrow$ Rename(R_j)
  - $R_c \leftarrow$ Rename(R_k)
  - $R_a \leftarrow$ Un registro físico libre
  - Rename(R_i) $\leftarrow R_a$
  - $i_x : R_a \leftarrow R_b$ *op* R_c

Renombramiento de registros

- *i1*: R1 ← R2/R3
- *i2*: R4 ← R1 + R5 → Dependencia RAW con i1
- *i3*: R5 ← R6 + R7 → Dependencia WAR con i2
- *i4*: R1 ← R8 + R9 → Dependencia WAW con i1

Scoreboard CDC 6600

i2 no termina la etapa *Dispatch* hasta que no termine la división de *i1*.

i3 no puede escribir hasta que *i2* no lea los operandos.

i4 no termina el *Issue* por conflicto WAW con *i1*.

Renombramiento de registros

- *i1*: R1 ← R2/R3
- *i2*: R4 ← R1 + R5 → Dependencia RAW con i1
- *i3*: R5 ← R6 + R7 → Dependencia WAR con i2
- *i4*: R1 ← R8 + R9 → Dependencia WAW con i1

- Luego del renombramiento de los registros
  - *i1*: Rc ← Ra/Rb
  - *i2*: Re ← Rc + Rd → Dependencia RAW con i1
  - *i3*: Rf ← Rg + Rh → Ya no hay dependencia
  - *i4*: Rk ← Ri + Rj → Ya no hay dependencia

Buffer de reordenamiento

Original	*Renombrado*
i1: R1 ← R2/R3	*i1*: Rc ← Ra/Rb
i2: R4 ← R1 + R5	*i2*: Re ← Rc + Rd
i3: R5 ← R6 + R7	*i3*: Rf ← Rg + Rh
i4: R1 ← R8 + R9	*i4*: Rk ← Ri + Rj

- Ya no hay dependencias WAW y WAR.
- Probablemente *i3* e *i4* terminen antes que *i1* e *i2*.
- Resultado de ejecución fuera de orden ≡ Resultado de ejecución secuencial
- Evitar que Rf y Rk se copien en R5 y R1 antes de que Rc y Re se copien en R1 y R4

Buffer de reordenamiento

- El *buffer de reordenamiento* (ROB – reorder buffer) es una cola FIFO donde los elementos son tuplas:
  - Un flag indicando si la instrucción completó la ejecución
  - El valor computado por la instrucción
  - El nombre del registro lógico donde debe almacenarse el resultado
  - El tipo de la instrucción: aritmética, load, store, branch, …

Buffer de reordenamiento

- Durante el renombramiento de una instrucción se inserta una entrada al ROB

  (false, N/A, R_i, op)

- Cuando se completa la instrucción, se modifica la entrada en el ROB con el resultado y se pone en *true* el flag indicando que terminó.
- Durante la etapa de *commit,* si la instrucción terminó y es la primera del ROB, se almacena el resultado en el registro destino.

Estaciones de reservación

- *Front-end* → renombrar registros e insertar instrucciones en el ROB
- Ultima etapa del *back-end* (*commit*) → almacenar resultados

- ¿Dónde esperan las instrucciones antes de ejecutar?
- ¿Cómo se sabe que una instrucción puede ejecutar?

Estaciones de reservación

- Luego de renombrar los registros e ingresar la entrada en el ROB, la instrucción se envía (despacha) a una *estación de reservación*.
- La estación de reservación debe contener:
  - La operación a realizar
  - El valor de los operandos o el nombre de los registros físicos (indicado por un *flag*)
  - El nombre del registro físico del resultado
  - La entrada en el ROB donde debe guardarse el resultado

Estaciones de reservación

- Las estaciones de reservación pueden ser:
  - Centralizadas (*Ventana de instrucción*)
  - Descentralizadas: asociadas a una UF (o a un grupo)

- Si en un dado ciclo más de una instrucción está lista para operar en la misma UF, se debe tomar una decisión de planificación:
  - La instrucción más vieja según el orden del programa
  - Operaciones identificadas como *críticas*

Estaciones de reservación

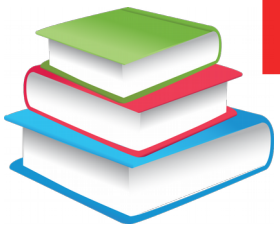
- Para ejecutar la instrucción hay que enviar los datos a la UF.
  - Es necesario detectar cuándo los operandos están disponibles.
- Se asocia un *ready bit* a cada registro físico para indicar si el valor de ese registro fue calculado o no.

Estaciones de reservación

- En la etapa de renombramiento, al mapear el registro destino de una instrucción se pone el *ready bit* en falso.
- Cuando se envía la instrucción a la estación de reservación, si el *ready bit* del registro fuente es
  - Verdadero: se le pasa el valor a la estación de reservación.
  - Falso: se le pasa el nombre del registro a la estación de reservación.

  Cuando todos los registros fuente tienen un valor, la instrucción está lista para pasar a la ejecución en la UF.
- Cuando una instrucción termina
  - Hace un *broadcast* a todas las estaciones de reservación del nombre del registro físico y el valor.
  - Se escribe el valor en el registro físico destino y se pone el *ready bit* en verdadero.

Bibliografía



- *Capítulo 3. Multiprocessor Architecture. From simple pipelines to chip multiprocessor*. Jean-Loup Baer. Cambridge University Press. 2010.
- *Capítulo 3 y Apéndice C.* John L. Hennessy & David A. Patterson. *Computer Architecture. A Quantitative Approach*. Elsevier Inc. 2012, 5ta Ed.

Suplementaria

- *Capítulo 7.* David Money Harris & Sarah L. Harris. *Digital Design and Computer Architecture*. Elsevier. 2013, 2da Ed.
- *Capítulo 4.* David A. Patterson & John L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface.* Elsevier Inc. 2014, 5ta Ed.