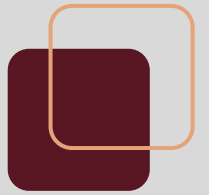
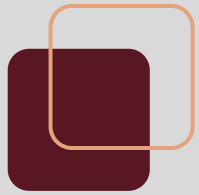


Aspectos avanzados de arquitectura de computadoras
Superescalares II

Facultad de Ingeniería - Universidad de la República

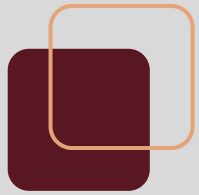
Curso 2017

Superescalares Out of Order (1/2)



- Más allá del hardware estudiado, generalmente se puede dividir la ejecución en cuatro etapas diferentes:
  - Fetch: Incluye la lectura de instrucciones desde memoria y la decodificación de las mismas. Esta etapa se realiza en orden.
  - Issue/Dispatch: Esta acción implica reservar una unidad funcional para la ejecución de la instrucción y enviar la misma para que comience a ejecutar. Puede ser realizada fuera de orden.

Superescalares Out of Order (2/2)



- Execute: Implica la ejecución de la instrucción, y posiblemente el writeback en el banco de registros físico. Esta etapa puede realizarse fuera de orden.
- Commit: Esta etapa implica la realización de acciones definitivas en el hardware, como escrituras en memoria y la escritura de resultados en el banco de registros de la arquitectura. Esta etapa debe realizarse en orden.

Out of Order Issue (1/3)

- Motivación:

  mul R2, R3, R4

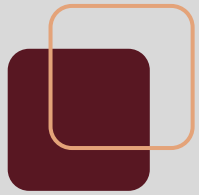
  add R4, R2, R6

  mul R5, R7, R8

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
mul R2, R3, R4	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C								
add R4, R2, R6		IF	ID	ID	ID	ID	ID	ID	ID	EX	MEM	WB	C							
mul R5, R7, R8			IF	IF	IF	IF	ID	IF	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C

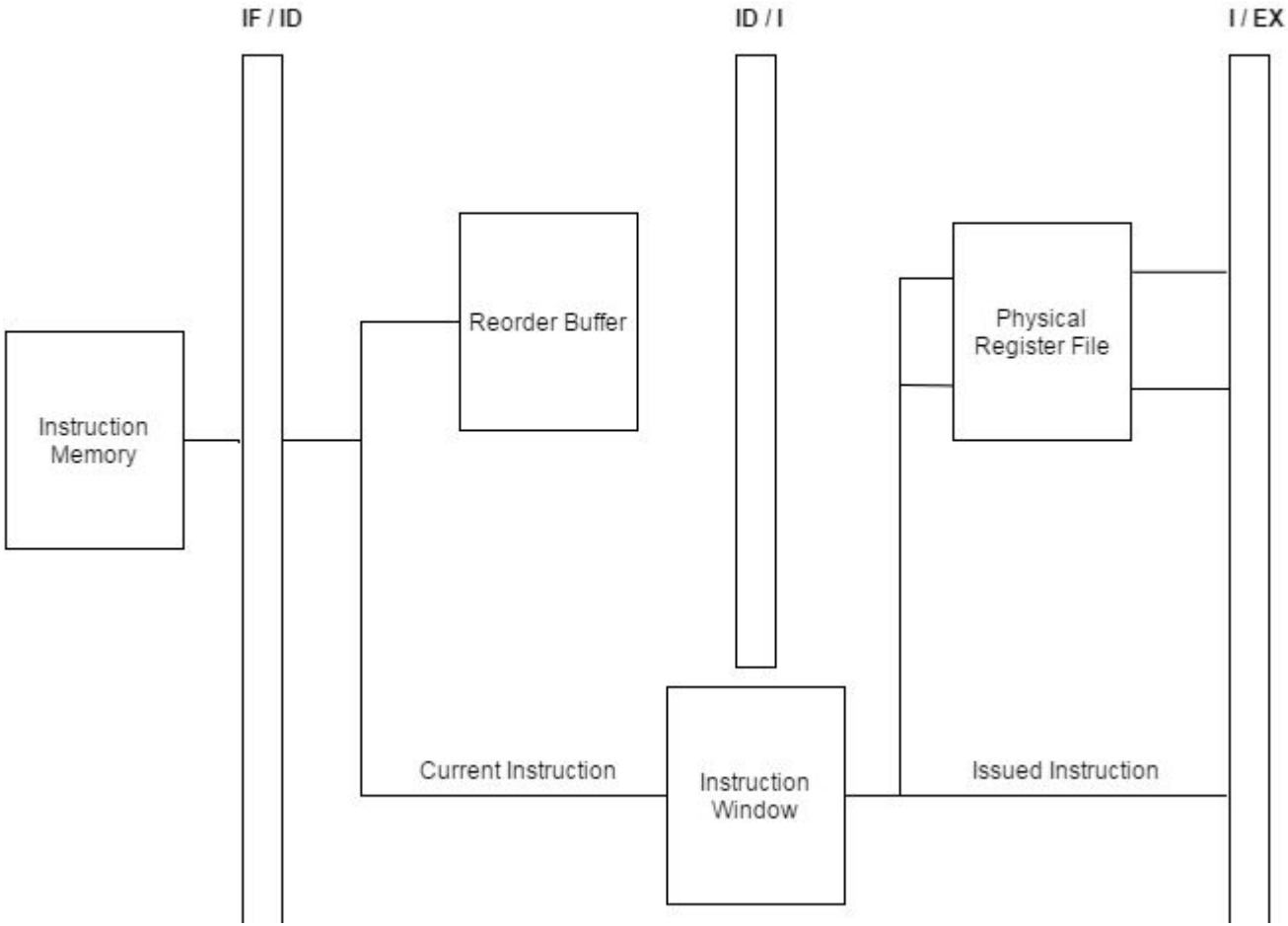
La tercer instrucción no depende de ninguna anterior. La razón por la que no comienza antes es que la emisión de instrucciones se realiza en orden!

Out of Order Issue (2/3)

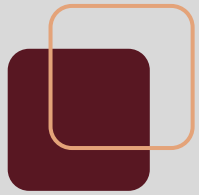


- Para lograr la emisión fuera de orden, se agrega una estructura de datos, llamada 'ventana de instrucciones' (instruction queue, issue queue), donde entran las instrucciones decodificadas *en orden*, y son retiradas *fuera de orden*, a medida que las mismas pueden ser ejecutadas sin provocar hazards.
- Para la lectura de la *instruction window* agregaremos una nueva etapa denominada *issue stage (I)*

Out of Order Issue (3/3)



Instruction Window (1/4)

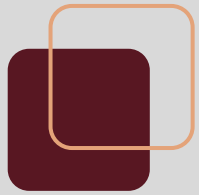


- Ejemplo para MIPS

OpCode	Inmediato	Spec?	V	Registro Destino	V	P	Registro Origen 1	V	P	Registro Origen 2
add	-	0	1	R1	1	0	R2	1	1	R5
subi	0x100	0	1	R3	1	1	R5	0	-	-
...										

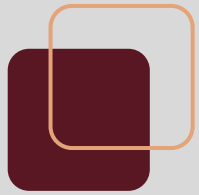
- *OpCode*: código de operación de la instrucción a ejecutar
- *Inmediato:* si la instrucción tiene un inmediato, se guarda el valor del mismo.

Instruction Window (2/4)



- *Spec?*: Indica si la instrucción es especulativa (más adelante)
- *Registros*: código del registro al cual se hace referencia en al instrucción.
- *V*: indica si el registro correspondiente es utilizado en la instrucción. Ejemplo: el registro destino sería *inválido* para una instrucción 'load'.
- *P:* indica si el registro correspondiente está pendiente de ser calculado por una instrucción anterior.

Instruction Window (3/4)



- ¿Cuándo una instrucción está lista para ser ejecutada?
  - Todos los registros que son válidos están disponibles (P = 0).
  - La emisión de la instrucción no provoca hazards estructurales (ver Scoreboard).

Instruction Window (4/4)

- ¿Cómo ejecuta ahora el código motivador?

  mul R2, R3, R4

  add R4, R2, R6

  mul R5, R7, R8

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mul R2, R3, R4	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C		
add R4, R2, R6		IF	ID							I	EX	MEM	WB	C	
mul R5, R7, R8			IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C

WAR Hazards (1/2)

- La emisión fuera de orden abre (en este pipeline), la posibilidad de hazards Write after Read

  mul R2, R3, R4

  add R3, R4, R2

  add R4, R6, R7

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mul R2, R3, R4	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C		
add R3, R4, R2		IF	ID							I	EX	MEM	WB	C	
add R4, R6, R7			IF	ID	I	EX	MEM	WB							C

El valor de R3 resulta incorrecto pues se utiliza un valor de R4 calculado posteriormente en el código!

WAR Hazards (2/2)

- Soluciones?
  - Esperar

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mul R2, R3, R4	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C		
add R3, R4, R2		IF	ID							I	EX	MEM	WB	C	
add R4, R6, R7			IF	ID				I	EX	MEM	WB				C

    - Se debe agregar lógica de control que permita determinar cuándo puede ejecutar la instrucción sorteando el hazard.
  - Register Renaming

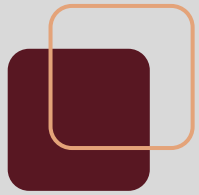
Register Renaming (1/6)

- El problema con los hazards WAW y WAR surge por reutilizar registros.
- Si se utiliza un registro adicional, el problema se sortea:

```
mul R2, R3, R4
add R3, R4, R2
add R11, R6, R7
```

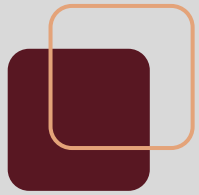
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mul R2, R3, R4	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C		
add R3, R4, R2		IF	ID							I	EX	MEM	WB	C	
add R11, R6, R7			IF	ID	I	EX	MEM	WB							C

Register Renaming (2/6)



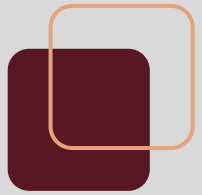
- El ejemplo anterior soluciona el problema, pero para que la instrucción esté modificada, exige trabajo del compilador.
- La técnica de *register renaming* consiste en generar el mismo efecto, pero dinámicamente a través del hardware.
- Al entrar en la *ventana de instrucciones*, se mapea el registro destino a uno de los registros del banco de registros físico.

Register Renaming (3/6)



- Dado que el renombrado se realiza en hardware, los registros del banco de registros físico no son visibles para el programador.
- Para obtener buena performance, el banco de registros físico debe ser más grande que el de la arquitectura.
- Se debe mantener el mapeo de qué registro físico contiene el valor de cada registro de la arquitectura (*Rename Table*)

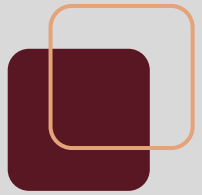
Register Renaming (4/6)
Rename Table



Registro Real (Arquitectura)	Pendiente	Registro Físico
R1	0	PR1
R2	1	PR63
...	...	...
R31	0	PR38

- Pendiente: Indica si hay una instrucción ejecutándose que escribe a este registro.
- Registro Físico: Código del registro físico al cual está mapeado este registro.

Register Renaming (5/6)
Free List



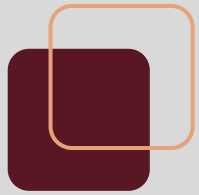
- Adicionalmente a la *Rename Table*, se mantiene una arreglo de bits para indicar cuáles registros físicos están libres para realizar el renombrado.

Free List

PR1	PR2	PR3	PR4	...	...	...	...	...	...	...	PRN
0	1	1	1								0

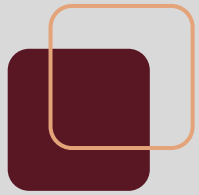
(PRN: Physical Register N)

Register Renaming (6/6)
Algoritmo de Renombrado



- Al ingresar una instrucción a la ventana de instrucciones, se realiza el renombrado del registro destino. Se elige un registro físico de la lista de registros físicos libres y se actualiza la tabla de renombrado.
- Adicionalmente, se utiliza la *rename table* para verificar cuáles registros físicos contienen los operandos origen de la instrucción.

Ejecución Especulativa (1/5)



- La emisión fuera de orden plantea nuevos problemas para la ejecución de instrucciones.

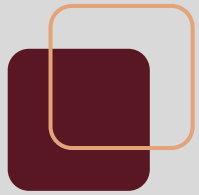
```
    mul   R3, R5, R6
    beqz R3, target  ; salta a target si R3 = 0
    addu R2, R4, R5
    addu R3, R6, R8
    addu R5, R8, R8
    addu R8, R4, R2
target:
    subu R2, R4, R5
```

Ejecución Especulativa (2/5)

	1	2	3	4	5	6	7	8	9	10	11	12	13
mul R3, R5, R6	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	
beqz R3, target		IF	ID							I	EX	MEM	WB
addu R2, R4, R5			IF	ID	I	EX	MEM	WB					
addu R3, R6, R8				IF	ID	I	EX	MEM	WB				
addu R5, R8, R8					IF	ID	I	EX	MEM	WB			
addu R8, R4, R2						IF	ID	I	EX	MEM	WB		
...													
...													
subu R2, R4, R5												IF	ID

- Si no se toman precauciones, las instrucciones posteriores al salto que no debían ser ejecutadas podrían impactar sus resultados!

Ejecución Especulativa (3/5)



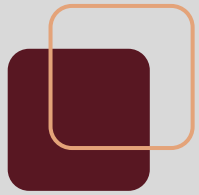
- Cuando un salto condicional (branch) entra en la ventana de instrucciones, las siguientes instrucciones son marcadas como *especulativas*, pues hasta que el resultado del salto no es calculado, es posible que las mismas no deban ser ejecutadas.
- Al conocerse el resultado del salto, se eliminan las instrucciones incorrectamente cargadas y se pone en 0 el bit especulativo de las correctas.

Ejecución Especulativa (4/5)

- Otra de las funciones de la etapa de commit es evitar que instrucciones especulativas impacten sus resultados.

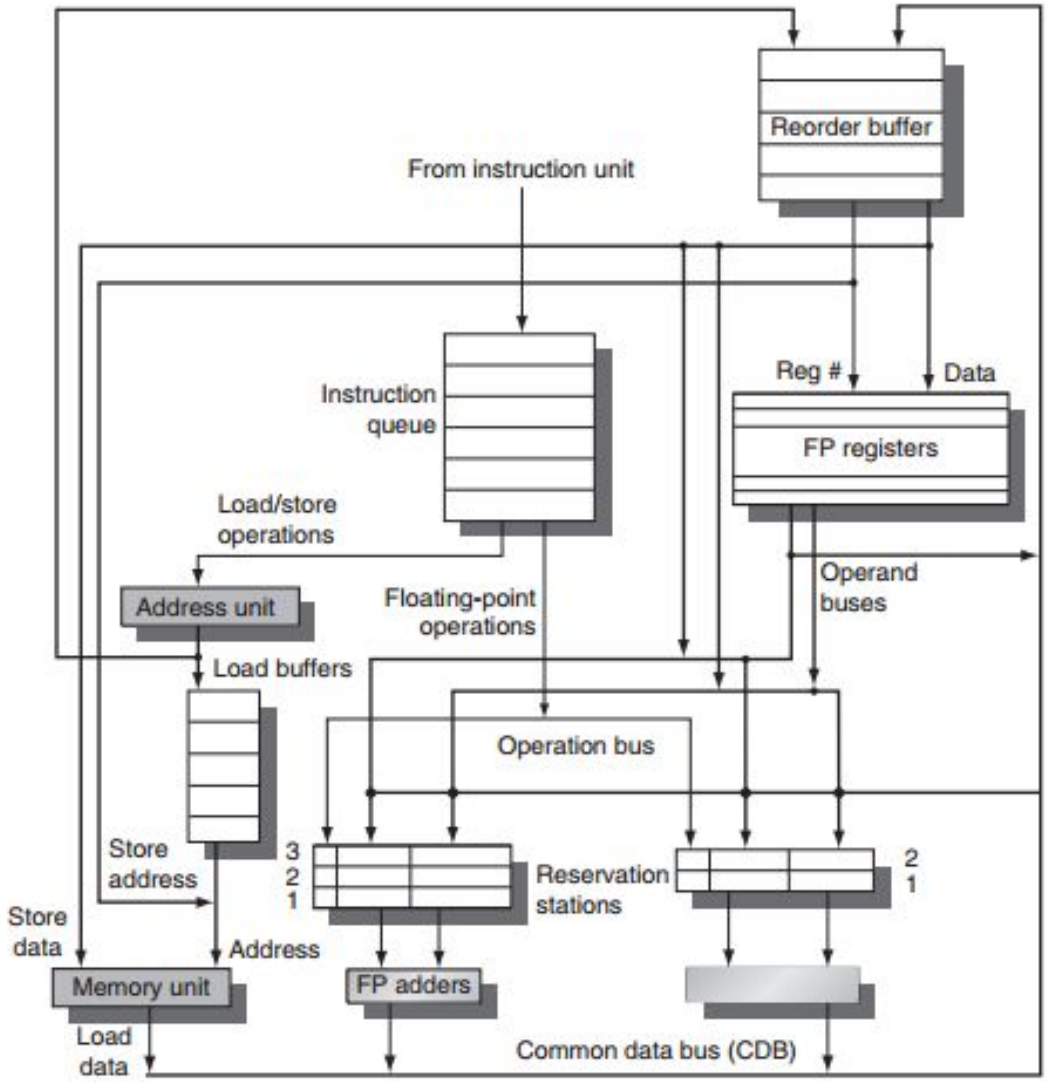
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
mul R3, R5, R6	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C				
beqz R3, target		IF	ID							I	EX	MEM	WB	C			
addu R2, R4, R5			IF	ID	I	EX	MEM	WB									
addu R3, R6, R8				IF	ID	I	EX	MEM	WB								
addu R5, R8, R8					IF	ID	I	EX	MEM	WB							
addu R8, R4, R2						IF	ID	I	EX	MEM	WB						
...																	
...																	
subu R2, R4, R5												IF	ID	I	EX	MEM	C

Ejecución Especulativa (5/5)

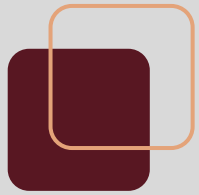


- Si bien una instrucción puede modificar el *banco de registros físico* en la etapa de write back siendo especulativa, no podrá impactar en el banco de registros de la arquitectura pues esperará en el reorder buffer hasta que el salto resuelva, pues este es anterior en el orden lógico del programa.
- Cuando la instrucción es desechada, los valores incorrectamente escritos en el banco de registros físico deben ser reemplazados desde el banco de registros real.

Algoritmo de Tomasulo (1/4)

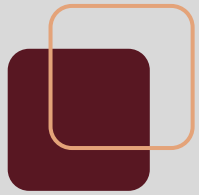


Algoritmo de Tomasulo (2/4)



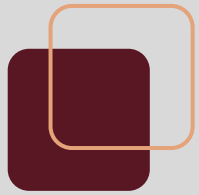
- La IBM 360/91 (1967) fue la primera en implementar el algoritmo de Tomasulo. Novedoso por incluir Register Renaming y *Reservation Stations*.
- Las Reservation Stations cumplen el rol del scoreboard y parte del de la ventana de instrucciones ya visto. Adicionalmente proveen la técnica de register renaming.

Algoritmo de Tomasulo Reservation Stations (3/4)



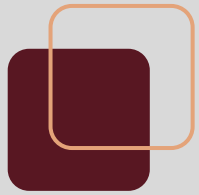
- Una reservation station contiene la información de una instrucción pendiente de ejecución, y guarda sus operandos origen hasta que todos estén disponibles.
- Una vez que todos los operandos son obtenidos, la instrucción puede comenzar a ejecutar.
- El renombrado de registros se provee renombrando el registro destino al número de *reservation station* que contiene la instrucción.

Algoritmo de Tomasulo
Common Data Bus (4/4)



- La actualización de las reservation stations se realiza a través de un bus común, el cual realiza un *broadcast* de los resultados que están siendo guardados en los registros.
- Resultado: La lógica de búsqueda de operandos es distribuida, así como la de *forwarding*, ya que cada reservation station tiene la información de qué registros precisa.

Fin



¿Preguntas?