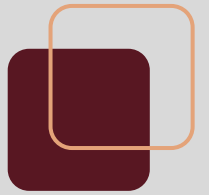
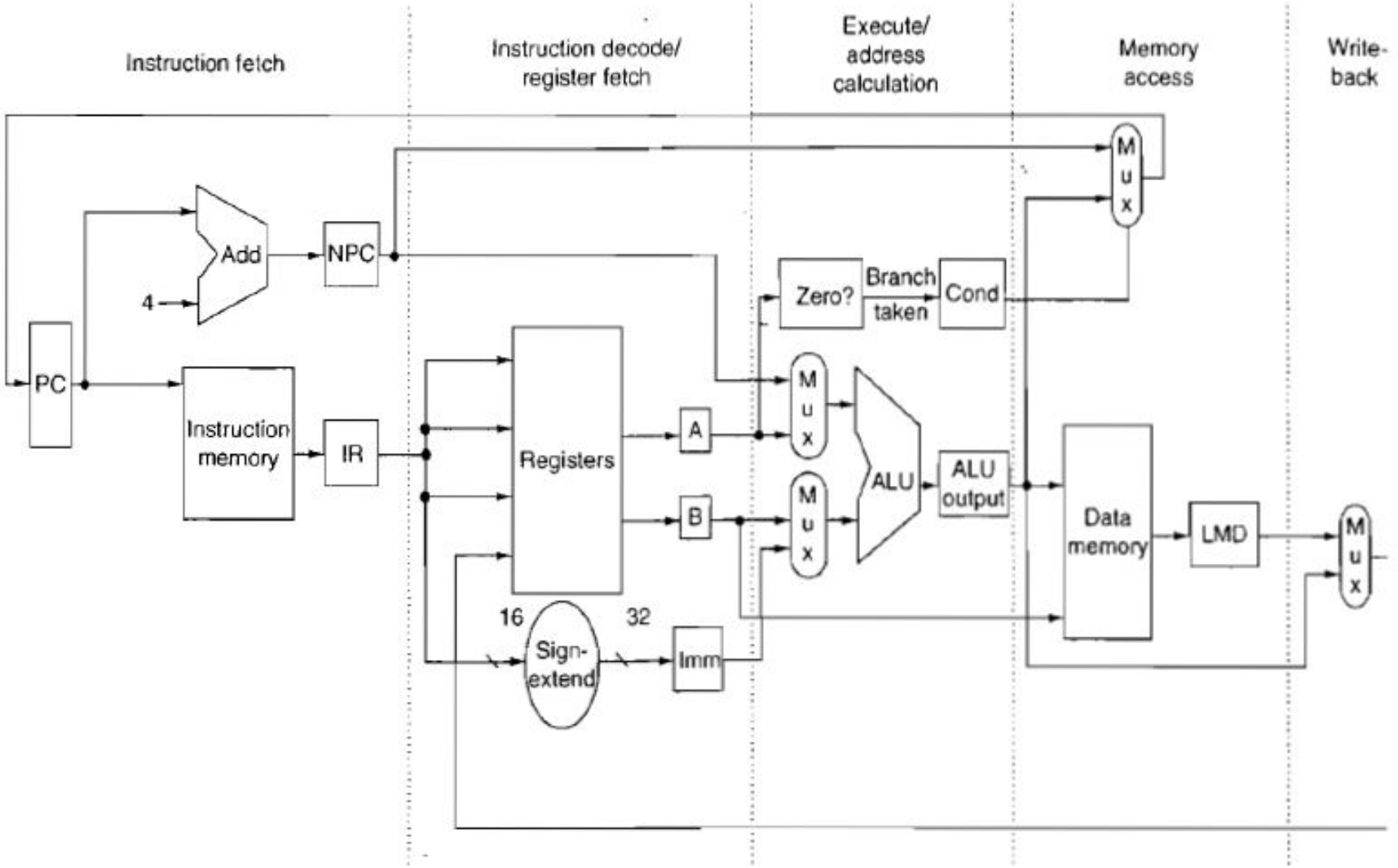


Arquitectura de computadoras
Superescalares

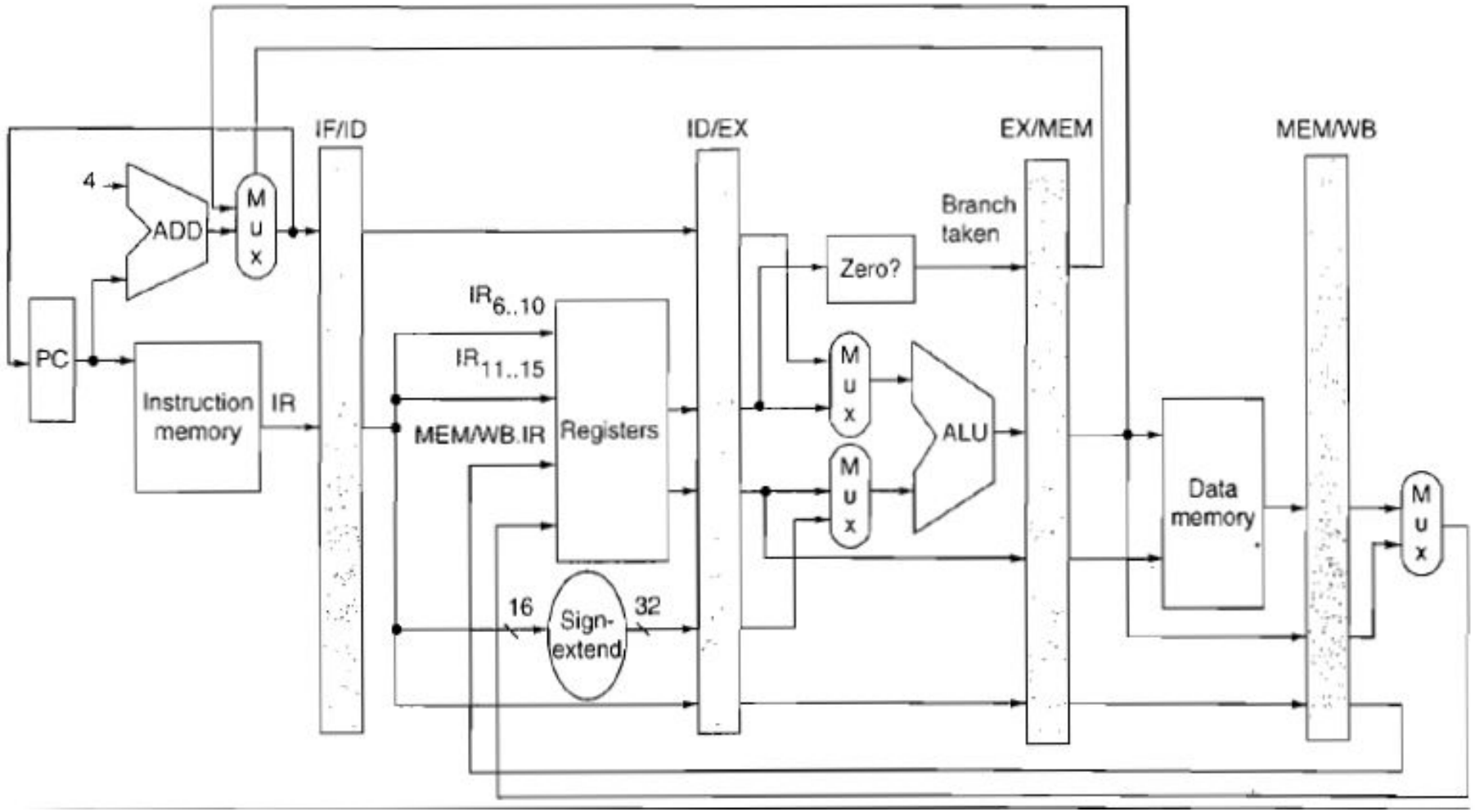
Facultad de Ingeniería - Universidad de la República

Curso 2024

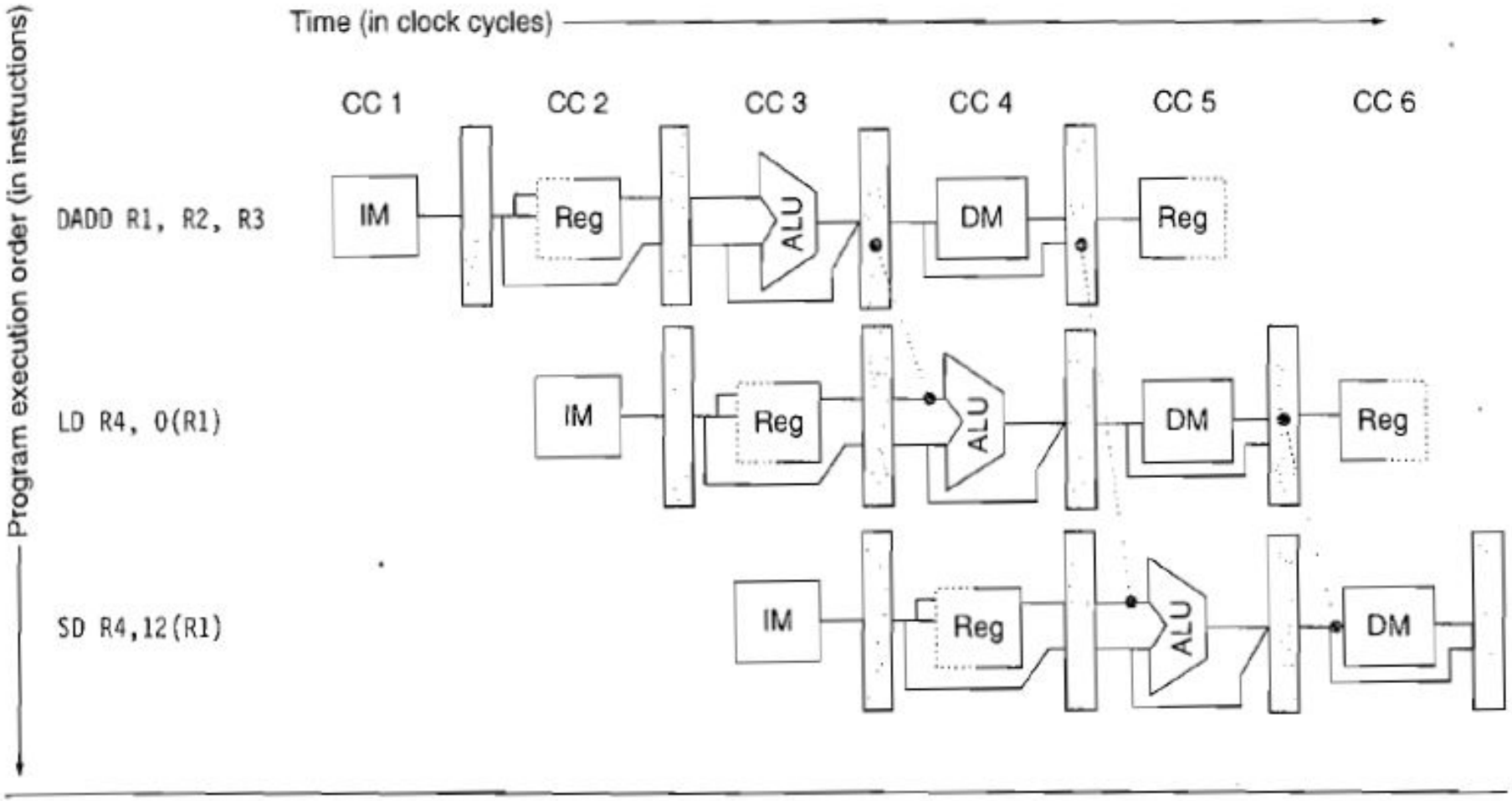
CPU MIPS SIN Pipeline



Pipeline MIPS



Pipeline MIPS: Forwarding

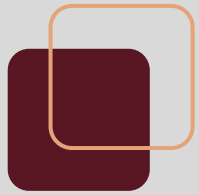


Diagramas de tiempos

- La ejecución del pipeline se puede expresar con un diagrama de tiempo vs instrucciones.

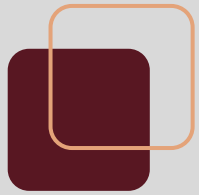
	Clock number								
Instruction number	1	2	3	4	5	6	7	8	9
Instruction i	IF	ID	EX	MEM	WB				
Instruction $i+1$		IF	ID	EX	MEM	WB			
Instruction $i+2$			IF	ID	EX	MEM	WB		
Instruction $i+3$				IF	ID	EX	MEM	WB	
Instruction $i+4$					IF	ID	EX	MEM	WB

Operaciones Multiciclo (1/11)



- Una simplificación introducida en el pipeline MIPS de 5 etapas es que todas las operaciones tienen la misma duración en etapa de ejecución.
- En la práctica esto no ocurre, ya que por ejemplo, se diferencian:
  - Operaciones de ALU enteras
  - Multiplicaciones/Divisiones enteras
  - Operaciones de ALU PF

Operaciones Multiciclo (2/11)



- Al agregar operaciones multiciclo, hay dos alternativas:
  - Alargar el ciclo de reloj para mantener el pipeline.
  - Separar las unidades funcionales como diferentes *paths* de ejecución, cada una con diferente cantidad de etapas de duración.

Operaciones Multiciclo (3/11)

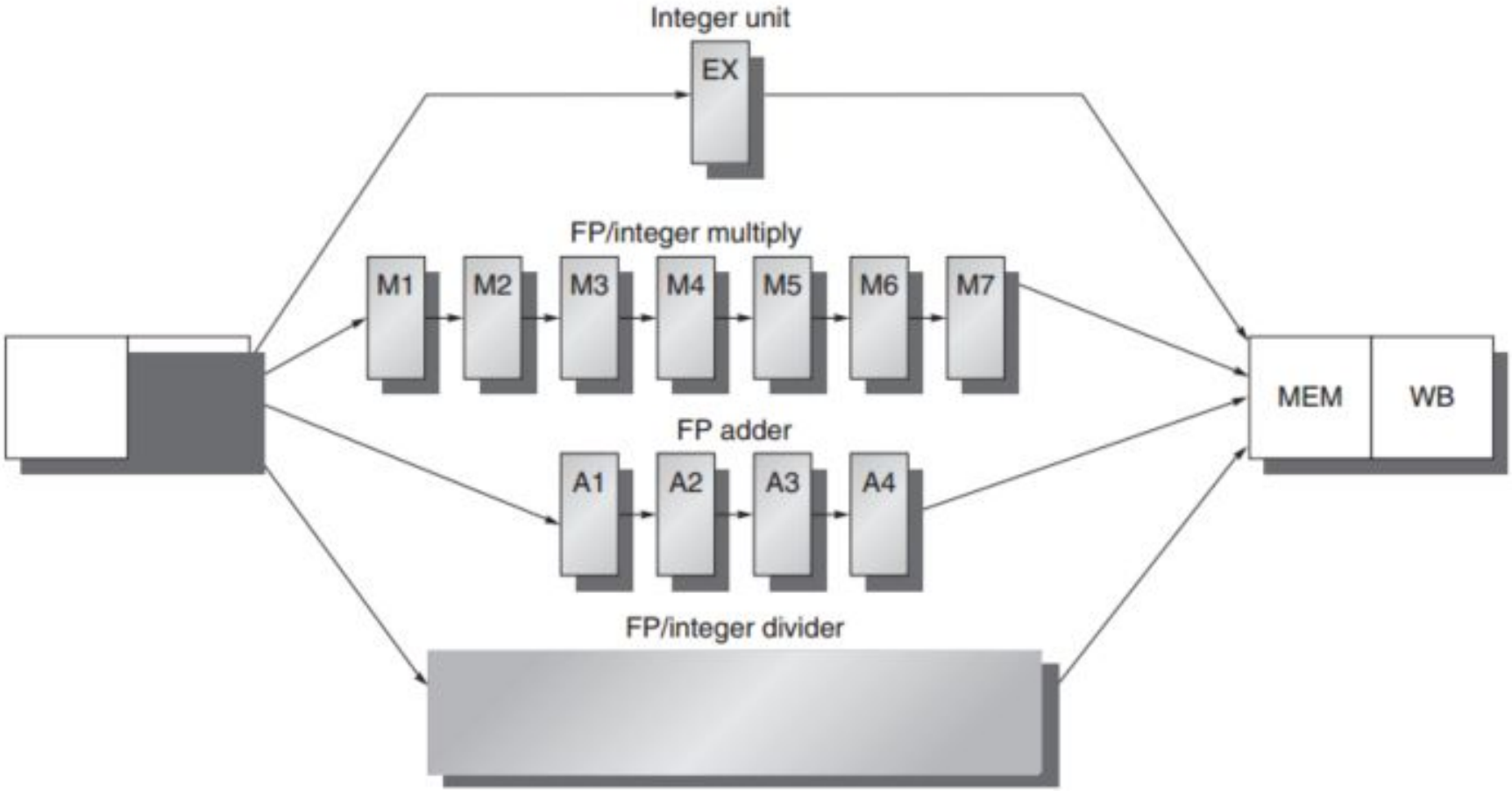
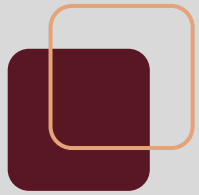


Imagen tomada del libro Computer Architecture, a Quantitative Approach. John L. Hennessy y David Patterson (2011)

Operaciones Multiciclo (4/11)



- El pipeline presentado tiene las siguientes unidades funcionales:
  - ALU entera (1 ciclo)
  - Multiplicador FP/Entero (7 ciclos, en pipeline)
  - Sumador FP (4 ciclos, en pipeline)
  - Divisor (24 ciclos, sin pipeline)

Operaciones Multiciclo (5/11)

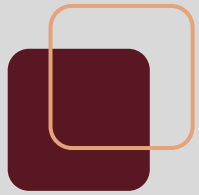
- ¿Cómo se ve la ejecución ahora?

MUL.D	IF	ID	*M1*	M2	M3	M4	M5	M6	M7	MEM	WB
ADD.D		IF	ID	*A1*	A2	A3	A4	MEM	WB		
L.D			IF	ID	*EX*	MEM	WB				
S.D				IF	ID	*EX*	*MEM*	WB			

- Como las unidades funcionales tienen diferentes latencias, las instrucciones pueden finalizar en diferente orden al que aparecen en el programa (*finalización fuera de orden*)

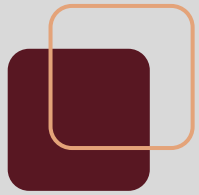
Imagen tomada del libro Computer Architecture, a Quantitative Approach. John L. Hennessy y David Patterson (2011)

Operaciones Multiciclo (6/11)
Nuevos Posibles Hazards



- Como los largos de las instrucciones son diferentes, es posible que múltiples instrucciones lleguen a la etapa MEM a la misma vez.
- Es posible tener hazards WAW dado que las instrucciones no terminan su ejecución en orden.
- Dado que el divisor no está en pipeline, pueden ocurrir hazards estructurales (múltiples divisiones deben esperar)

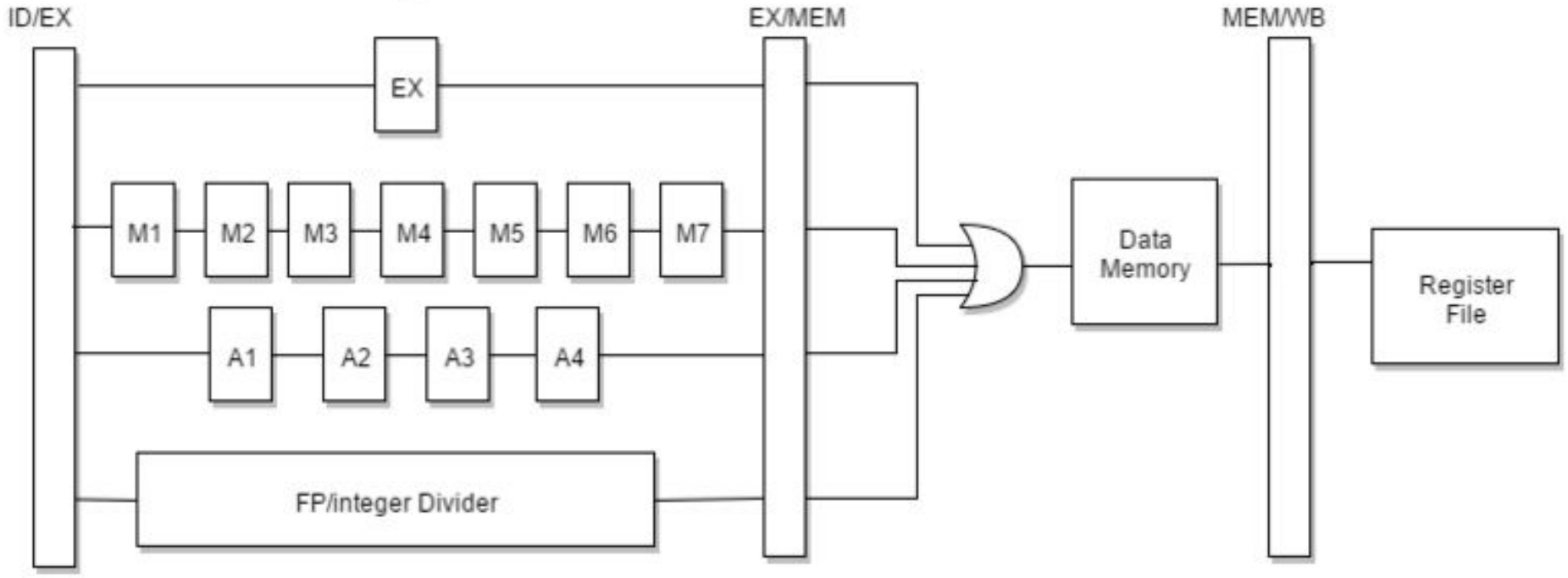
Operaciones Multiciclo (7/11)



- Dado que múltiples instrucciones pueden llegar a la etapa MEM a la misma vez, se debe decidir si a partir de ahí se multiplicarán las conexiones (permitiendo la finalización de múltiples instrucciones a la misma vez), o si se obligará a finalizar una instrucción por vez, obteniendo un *hazard estructural* en el caso de que varias instrucciones completen la ejecución a la vez.

Operaciones Multiciclo (8/11)

- Sin múltiples caminos:



- Se debe verificar en etapa ID que no puedan ocurrir hazards estructurales

Operaciones Multiciclo (9/11)

- Ejemplo:

  ADD.D F2,F0,F8

  XOR R1, R2, R3

  OR R4, R5, R6

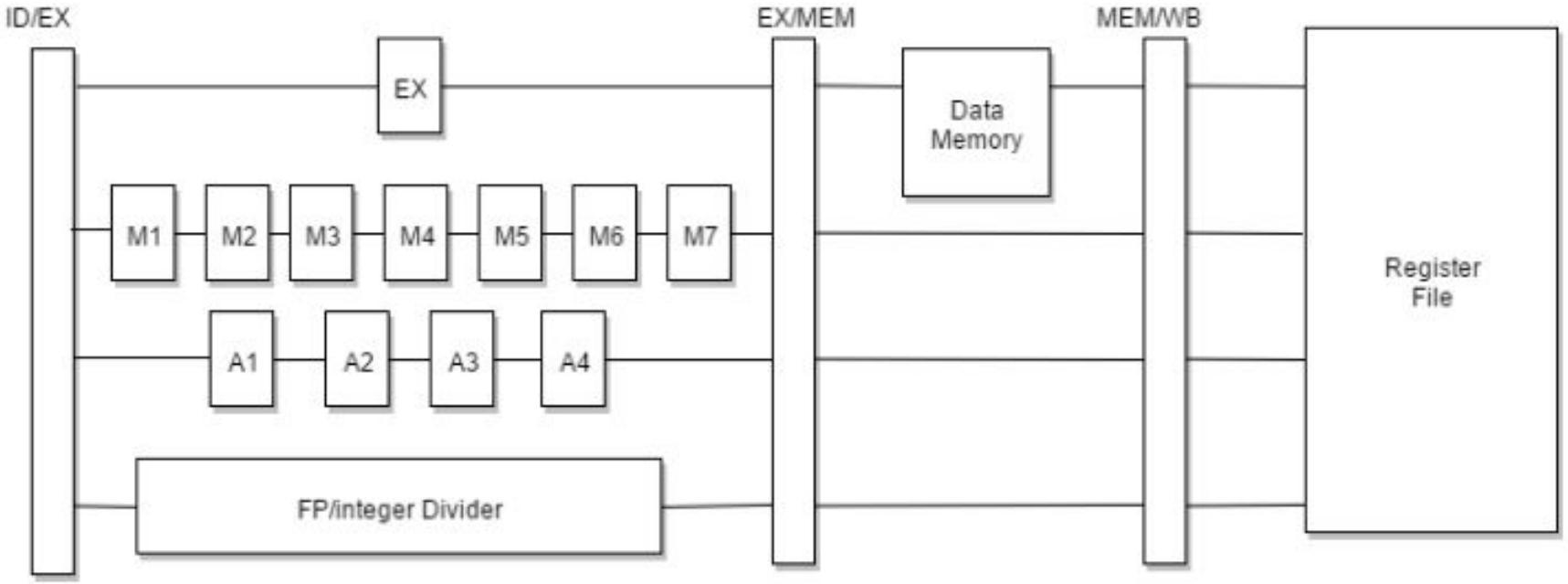
  ADD R7, R8, R9

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9
ADD.D F2, F0, F8	IF	ID	A1	A2	A3	A4	MEM	WB	
XOR R1, R2, R3		IF	ID	EX	MEM	WB			
OR R4, R5, R6			IF	ID	EX	MEM	WB		
ADD R7, R8, R9				IF	ID	ID	EX	MEM	WB

Hazard Estructural

Operaciones Multiciclo (10/11)

- Con múltiples caminos:



- Requiere múltiples puertos de escritura en banco de registros pero evita hazards estructural en etapa MEM.

Operaciones Multiciclo (11/11)

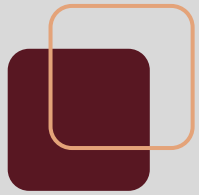
- Ejemplo:

```
ADD.D   F2, F0, F8
XOR     R1, R2, R3
OR      R4, R5, R6
ADD     R7, R8, R9
```

Instrucción\Ciclo	1	2	3	4	5	6	7	8
ADD.D F2, F0, F8	IF	ID	A1	A2	A3	A4	MEM	WB
XOR R1, R2, R3		IF	ID	EX	MEM	WB		
OR R4, R5, R6			IF	ID	EX	MEM	WB	
ADD R7, R8, R9				IF	ID	EX	MEM	WB

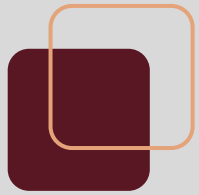
- Notar que hay múltiples instrucciones en etapas MEM/WB en el mismo ciclo!

Instrucciones Complejas (1/2)



- El caso extremo de la situación anterior se da con instrucciones *realmente* complejas de la arquitectura, como copia de strings o movimientos memoria-memoria.
- Adaptar el modelo de *pipeline* a estas instrucciones exige incluir unidades funcionales de decenas (o cientos!) de ciclos.

Instrucciones Complejas (2/2)



- Unidades funcionales largas impactan por varios lados:
  - Mayor cantidad de conexiones de *register forwarding*.
  - Mayor densidad de instrucciones finalizadas fuera de orden
- Para evitarlo, una estrategia implementada en CPUs de la arquitectura IA-32 es poner en el pipeline las *microinstrucciones* generadas por la UC en lugar de las propias instrucciones.

Hazards WAW (1/2)

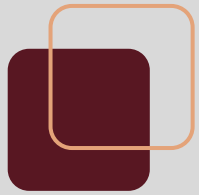
- La finalización fuera de orden introduce la posibilidad de que ocurran hazards WAW.
- Ejemplo:

```
MUL.D   F2,F0,F8
ADD.D   F2,F4,F6
```

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9	10	11
mul.d F2, F0, F8	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
add.d F2, F4, F6		IF	ID	A1	A2	A3	A4	MEM	WB		

- Hazard WAW!

Hazards WAW (2/2)

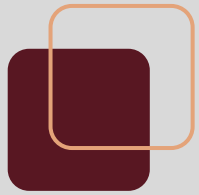


- Para evitar el hazard, la dependencia puede detectarse en decodificación y detener el pipeline.

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9	10	11	12
mul.d F2, F0, F8	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB	
add.d F2, F4, F6		IF	ID	ID	ID	ID	A1	A2	A3	A4	MEM	WB

- Otra solución es implementar *register renaming* (presentado más adelante)

Interrupciones (1/3)



- El manejo de las interrupciones se vuelve más complejo en los pipelines con finalización fuera de orden.
- Ejemplo:

```
DIV.D    F0, F2, F4
ADD.D    F10,F10,F8
SUB.D    F12,F12,F14
```

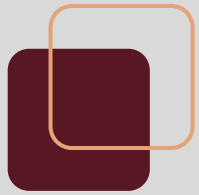
Interrupciones (2/3)

- El manejo de las interrupciones se vuelve más complejo en los pipelines con finalización fuera de orden.

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13
div.d F0,F2,F4	IF	ID	DIV	DIV	DIV	DIV	DIV	DIV	DIV	DIV	DIV	DIV	...
add.d F10,F10,F8		IF	ID	A1	A2	A3	A4	MEM	WB				
sub.d F12, F12, F14			IF	ID	A1	A2	A3	A4	MEM	WB			

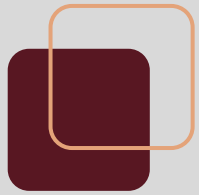
- Si ocurre una interrupción en este momento, no se puede atender hasta no terminar la división!

Interrupciones (3/3)



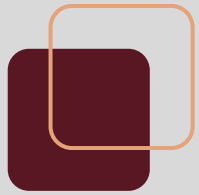
- ¿Soluciones?
  - Mantener un *buffer* con las operaciones finalizadas, impactando sus resultados una vez que todas las instrucciones que se despacharon antes finalizaron, obligando a las instrucciones a finalizar en orden.

Commit Stage (1/3)



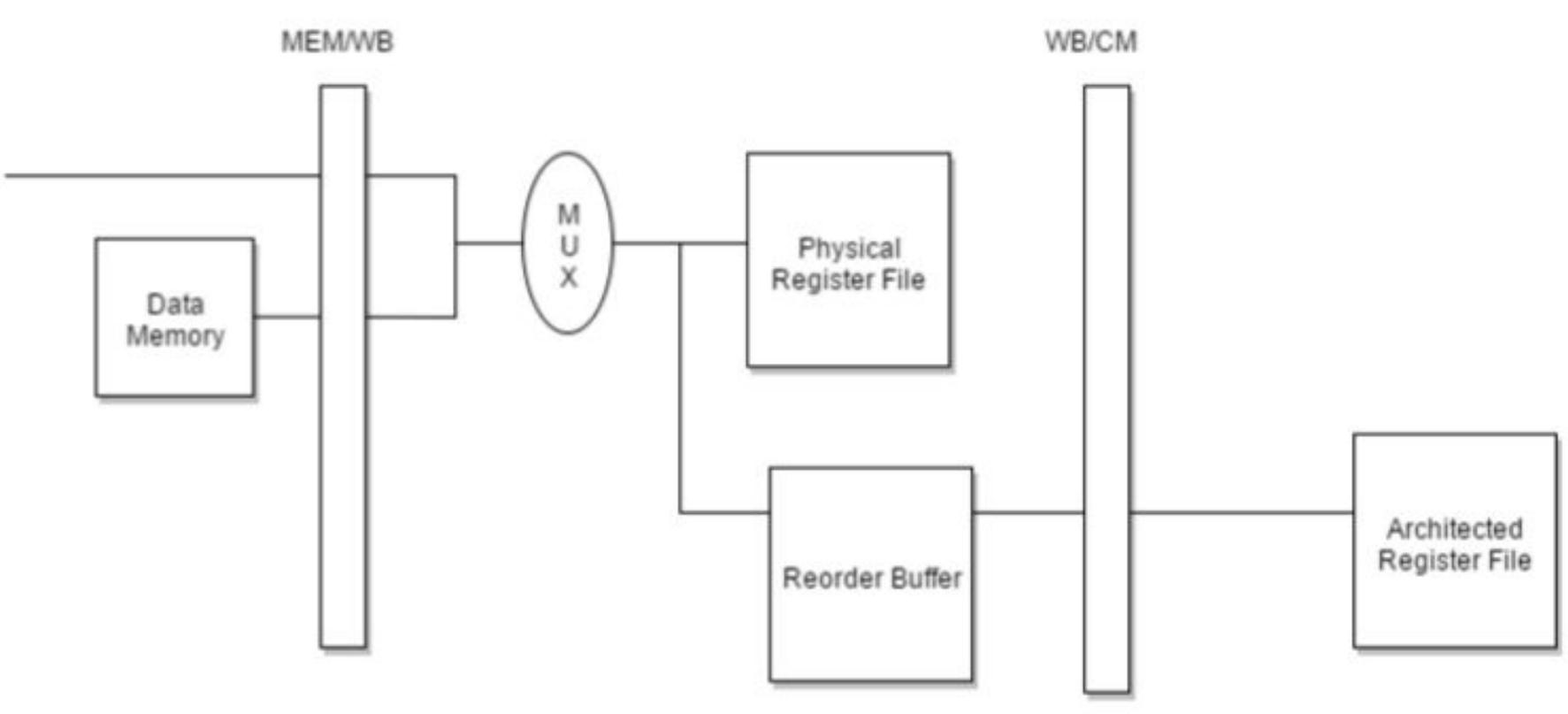
- Para poder atender interrupciones sin importar en qué etapa se produzcan, se debe agregar una etapa adicional, denominada *commit*. Al llegar a dicha etapa una instrucción se considerará *finalizada*.

Commit Stage (2/3)



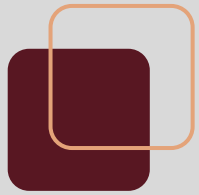
- Se agrega un nuevo banco de registros, el cual será actualizado en la etapa commit, y que tendrá únicamente el propósito de servir para restaurar el estado en caso de una interrupción.
- De este modo, se diferencian el *banco de registros físico* (el usado hasta el momento), del *banco de registros de la arquitectura* (el de la etapa commit).

Commit Stage (3/3)



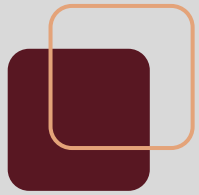
- El pipeline presentado tiene *ejecución fuera de orden* y *commit en orden.*

Reorder Buffer (1/2)



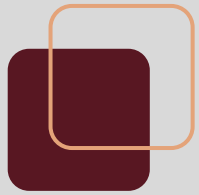
- Para asegurar el acceso en orden a la etapa commit, se puede utilizar una estructura de datos llamada *Reorder Buffer (ROB - Buffer de Reorden)*.
- El ROB es el encargado de almacenar instrucciones que finalicen fuera de orden, permitiendo que finalicen solo cuando no existan instrucciones anteriores pendientes (según el orden lógico del programa).

Reorder Buffer (2/2)



- Un ROB es esencialmente una cola circular:
  - Al emitir una instrucción, se agrega al ROB con estado pendiente (*queue*)
  - Al llegar a la etapa Write Back, se marca como finalizada.
  - Al final del ciclo, si la última instrucción del ROB está finalizada, se realiza el commit y se quita de la estructura (*dequeue*).

Instruction Level Parallelism



- Propiedad de un *programa*. Indica *qué tanto se puede paralelizar*.

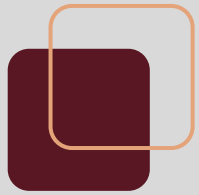
```
a = b + c;
d = e + f;
```

Estas dos instrucciones pueden ejecutarse en paralelo pues no hay dependencias de datos entre sí.

```
a = b + c;
d = a + f;
```

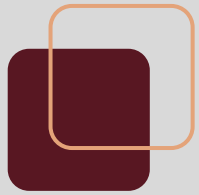
Estas dos instrucciones **no** pueden ser ejecutadas completamente en paralelo pues la segunda instrucción precisa el primer resultado para poder iniciar!

Machine Level Parallelism



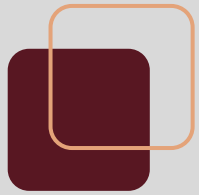
- Propiedad de un CPU. Indica qué tanto código puede ejecutar de forma paralela.
- Cuando un programa con ILP se ejecuta en un CPU con MLP, se acelera la ejecución.
- Hasta el momento se han estudiado procesadores que como máximo pueden ejecutar UNA instrucción por ciclo (IPC = 1)
- Los superescalares son procesadores que pueden lograr IPC > 1.

Técnicas de explotación de ILP



- Técnicas del Compilador (Estáticas)
  - El compilador detecta el ILP en tiempo de compilación y reordena o modifica el código, de forma que se aproveche mejor el paralelismo a nivel de máquina (ej: salto retardado).
- Técnicas de Hardware (Dinámicas)
  - El propio hardware detecta el ILP en tiempo de ejecución y lo aprovecha (ej: forwarding).

Técnicas Estáticas
Reordenar instrucciones (1/2)



- Supongamos el pipeline MIPS de 5 etapas sin forwarding y el código:
  - addi $3, $5, 4000
  - xor $4, $3, $2
  - sub $1, $5, $6

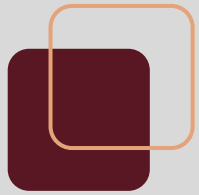
Instrucción\Ciclo	1	2	3	4	5	6	7	8	9
addi $3, $5, 4000	IF	ID	EX	MEM	WB				
xor $4, $3, $2		IF	ID	ID	ID	EX	MEM	WB	
sub $1, $5, $6			IF	IF	IF	ID	EX	MEM	WB

Técnicas Estáticas
Reordenar instrucciones (2/2)

- El compilador, previendo el stall, puede reordenar las instrucciones y reducir la penalización sin alterar el resultado del programa:
  - addi $3, $5, 4000
  - sub $1, $5, $6
  - xor $4, $3, $2

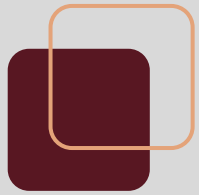
Instrucción\Ciclo	1	2	3	4	5	6	7	8
addi $3, $5, 4000	IF	ID	EX	MEM	WB			
sub $1, $5, $6		IF	ID	EX	MEM	WB		
xor $4, $3, $2			IF	ID	ID	EX	MEM	WB

Técnicas Estáticas: Conclusiones



- Permiten aumentar la performance sin tener que realizar mejoras en el procesador.
- Funcionan si se tiene conocimiento del hardware subyacente.

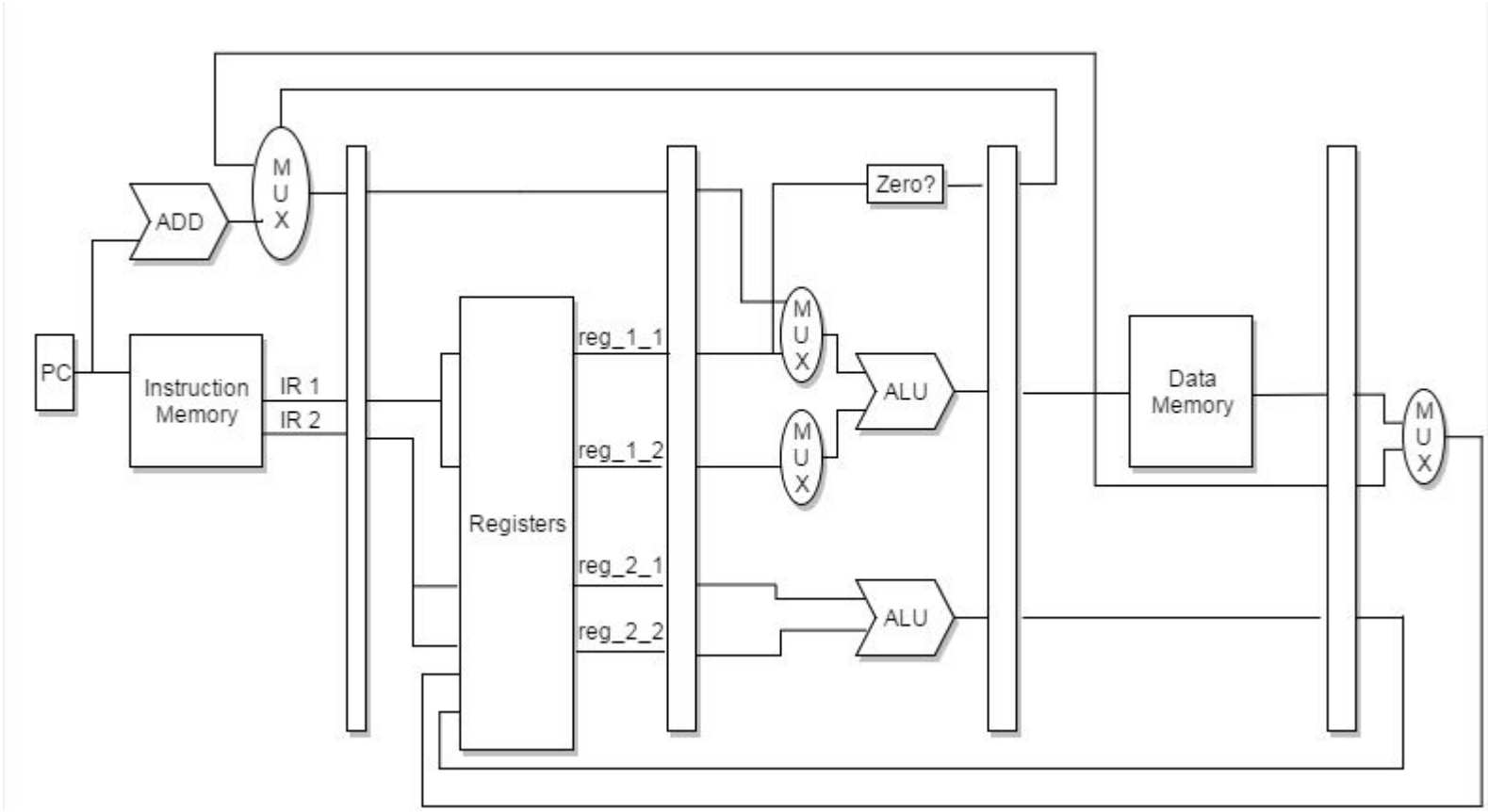
Organización Superescalar (1/2)



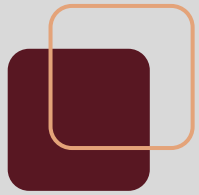
- La implementación superescalar implica como mínimo la capacidad de ejecutar dos instrucciones por ciclo. Un procesador con dichas características se denomina *two-wide superescalar* (superescalar de ancho 2)
- Un superescalar con posibilidad de ejecutar *N* instrucciones por ciclo es denominado *N-wide superescalar* (superescalar de ancho N)

Organización Superescalar (2/2)

- Ejemplo BÁSICO 2-wide superscalar:

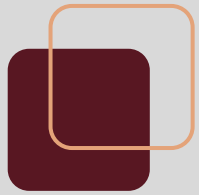


Recursos de Hardware (1/3)



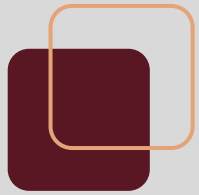
- ¿Qué recursos se requieren como mínimo para implementar un *2-wide superscalar?*
  - Fetch de (al menos) 2 instrucciones a la vez.
  - Cuatro puertos de lectura y dos de escritura en el banco de registros.
  - Lógica de despacho (issue logic)
  - Lógica de control duplicada
  - Al menos dos unidades funcionales (alus, multiplicadores, divisores, etc)

Recursos de Hardware (2/3)
Fetch Logic



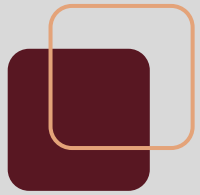
- Fetch de múltiples instrucciones en un mismo ciclo:
  - Bus más ancho entre CPU - Caché de Instrucciones L1
- Problemas:
  - Qué pasa si se realiza fetch desde múltiples bloques de caché al mismo tiempo?
  - Dependencias de control

Recursos de Hardware (3/3)
Fetch Logic



- Típicamente, se realiza el fetch de tamaños *fijos* del caché de instrucciones, y luego la etapa *fetch* toma las instrucciones que precisa/puede.
  - Ocasionalmente con algunas restricciones, como cargar de a bloques, o solo ciertos segmentos de bloques, alineados a medio bloque, etc.

Diagramas de Pipeline Superescalar



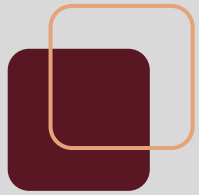
- Se levanta la restricción de que 'solo una instrucción puede estar en una etapa en el mismo ciclo'.
- Ejemplo *ideal* para el pipeline anterior:

	1	2	3	4	5	6	7	8
alu inst 1 R1, R2, R3	IF	ID	EX	MEM	WB			
alu inst 2 R4, R5, R6	IF	ID	EX	MEM	WB			
alu inst 3 R7, R8, R9		IF	ID	EX	MEM	WB		
alu inst 4 R10, R11, R12		IF	ID	EX	MEM	WB		
alu inst 5 R13, R14, R15			IF	ID	EX	MEM	WB	
alu inst 6 R16, R17, R18			IF	ID	EX	MEM	WB	
alu inst 7 R19, R20, R21				IF	ID	EX	MEM	WB
alu inst 8 R22, R23, R24				IF	ID	EX	MEM	WB

A partir del ciclo 5,

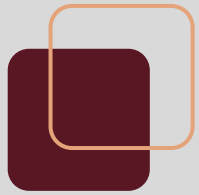
IPC = 2

Forwarding (1/2)



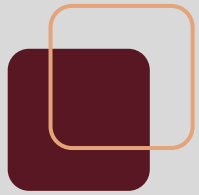
- ¿Cómo se implementa el forwarding?
  - De la misma manera que en el pipeline de 5 etapas, pero la cantidad de conexiones crece con el número de etapas del pipeline y el ancho del superescalar.
  - En el pipeline MIPS de 5 etapas, basta con un multiplexor de 4 entradas.
  - ¿Para el Intel Pentium 4, con más de 40 etapas y varios pipelines?

Forwarding (2/2)



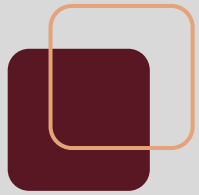
- ¿Solución?
  1. No implementar todas las conexiones de forwarding. Incluir solo conexiones más probables.
  2. Agregar etapa dedicada y estructuras de datos especiales.
  3. Cambiar a lógica distribuida:
     - Tomasulo (más adelante)

Issue Stage



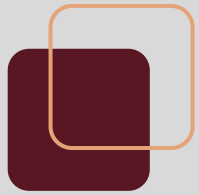
- A medida que el hardware de control aumenta, no es extraño dedicar una etapa completa para la generación de las señales. Típicamente se denomina *'Issue Stage'*
- En esta etapa se realiza generalmente la actualización de las estructuras de datos para control del pipeline.

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 (1/2)

- Motivación:

  mul R2, R3, R4

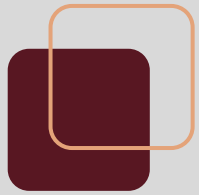
  add R4, R2, R6

  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

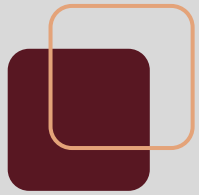
La tercer instrucción no depende de ninguna anterior. La razón por la que no comienza antes es que la emisión de instrucciones se realiza en orden!

Out of Order Issue (2/2)



- 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)*

Instruction Window (1/2)



- ¿Cuándo una instrucción está lista para ser ejecutada?
  - Todos los registros necesarios están disponibles (no están siendo utilizados por otra instrucción).
  - La emisión de la instrucción no provoca hazards estructurales.

Instruction Window (2/2)

- ¿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

Hazards WAR (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!

Hazards WAR (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

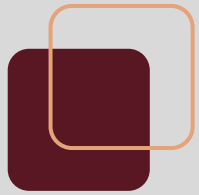
Register Renaming (1/5)

- 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
```

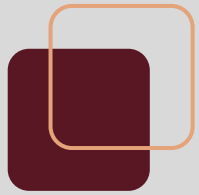
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
mul R2, R3, R4	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C		
add R3, R4, R2		IF	ID							I	EX	MEM	WB	C	
add R11, R6, R7			IF	ID	I	EX	MEM	WB							C

Register Renaming (2/5)



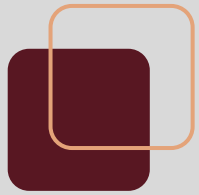
- El ejemplo anterior soluciona el problema, pero requiere trabajo del compilador.
- Alternativamente, la técnica de *register renaming* consiste en generar el mismo efecto pero dinámicamente, a través del hardware.
- Al entrar en la *ventana de instrucciones*, se mapea el registro destino a uno de los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

Register Renaming (3/5)



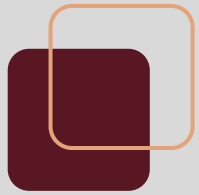
- Dado que el renombrado se realiza en hardware, los registros del banco de registros físico no son visibles para el programador.
- Para lograr buena performance, el banco de registros físico debe ser más grande que el de la arquitectura.

Register Renaming (4/5)
Estructuras de hardware



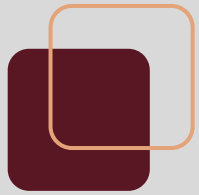
- Se debe mantener el mapeo de qué registro físico contiene el valor de cada registro de la arquitectura (*Rename Table*)
- Adicionalmente a la *Rename Table*, se mantiene una arreglo de bits para indicar cuáles registros físicos están libres para realizar el renombrado.

Register Renaming (5/5)
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.

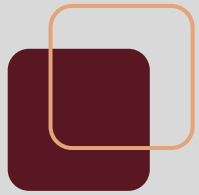
```
    mul   R3, R5, R6
    beqz R3, target    ; salta a destino si R3 = 0
    addu R2, R4, R5
    addu R3, R6, R8
    addu R5, R8, R8
    addu R8, R4, R2
destino:
    subu R2, R4, R5
```

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)



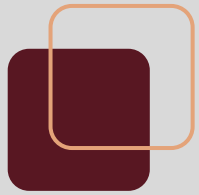
- Cuando un salto condicional (branch) entra en la ventana de instrucciones, las siguientes instrucciones son marcadas como *especulativas*, pues hasta que el resultado del salto no es calculado, es posible que las mismas no deban ser ejecutadas.
- Al conocerse el resultado del salto, se eliminan las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

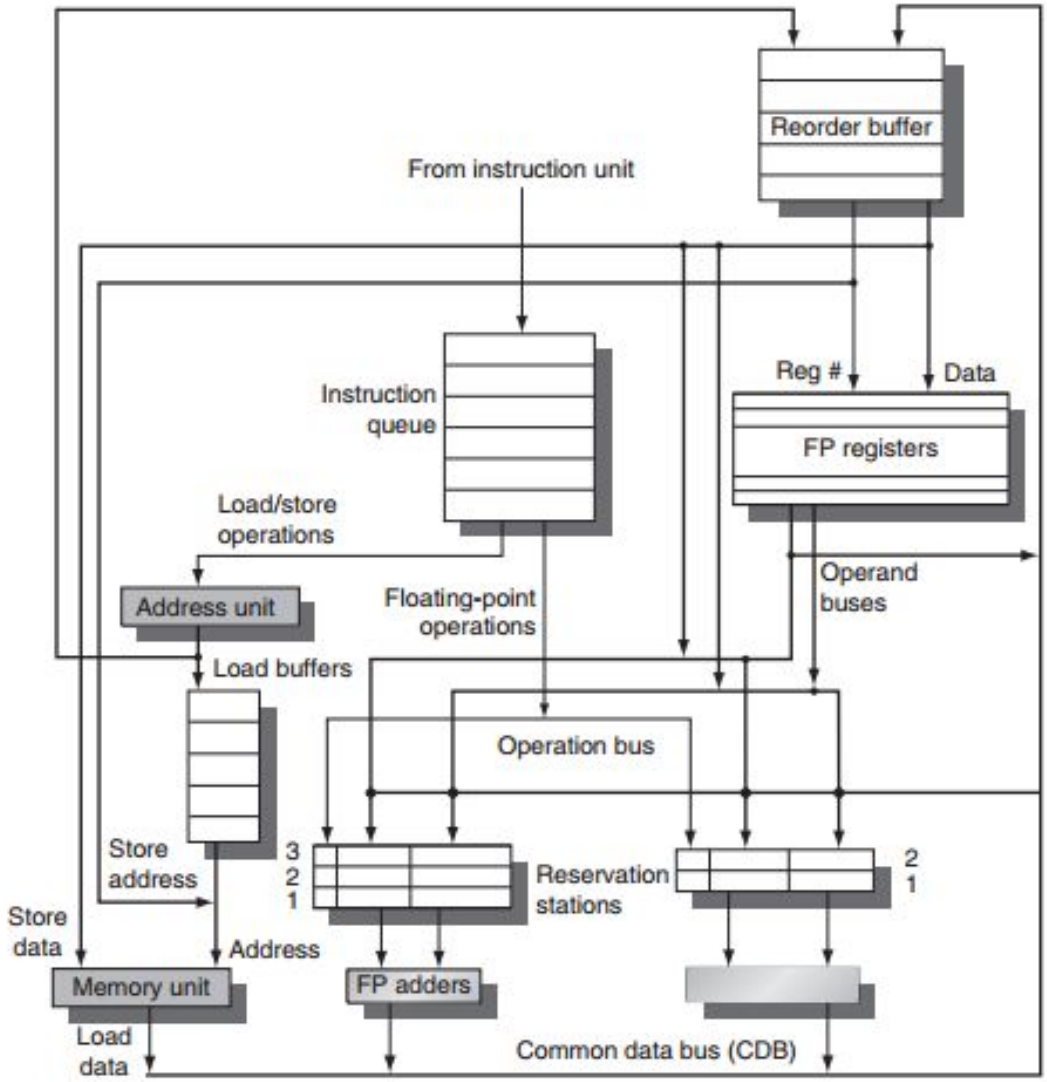
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
mul R3, R5, R6	IF	ID	I	M1	M2	M3	M4	M5	M6	M7	MEM	WB	C				
beqz R3, target		IF	ID							I	EX	MEM	WB	C			
addu R2, R4, R5			IF	ID	I	EX	MEM	WB									
addu R3, R6, R8				IF	ID	I	EX	MEM	WB								
addu R5, R8, R8					IF	ID	I	EX	MEM	WB							
addu R8, R4, R2					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

Ejecución Especulativa (5/5)

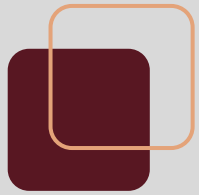


- Si bien una instrucción puede modificar el *banco de registros físico* en la etapa de write back siendo especulativa, no podrá impactar en el banco de registros de la arquitectura pues esperará en el reorder buffer hasta que se conozca si el salto se toma o no, pues este es anterior en el orden lógico 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 de la arquitectura.

Algoritmo de Tomasulo (1/4)

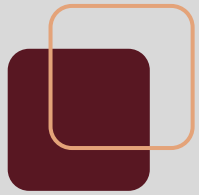


Algoritmo de Tomasulo (2/4)



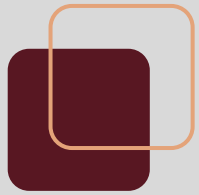
- La IBM 360/91 (1967) fue la primera en implementar el algoritmo de Tomasulo. Novedoso por incluir Register Renaming y *Reservation Stations*.
- Las Reservation Stations incluyen la lógica para detectar y mitigar hazards, y parte del de la ventana de instrucciones ya visto. Adicionalmente proveen la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

Algoritmo de Tomasulo
Reservation Stations (3/4)



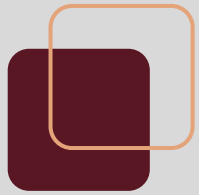
- Una reservation station contiene la información de una instrucción pendiente de ejecución, y guarda sus operandos origen hasta que todos estén disponibles.
- Una vez que todos los operandos son obtenidos, la instrucción puede comenzar a ejecutar.
- El renombrado de registros se provee renombrando el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