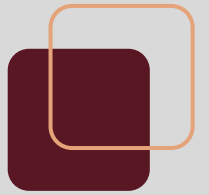
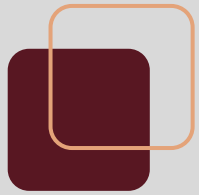


Aspectos avanzados de arquitectura de computadoras
Superescalares I

Facultad de Ingeniería - Universidad de la República

Curso 2017

Instruction Level Parallelism



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

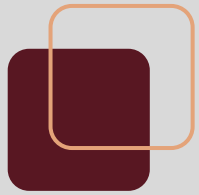
```
a = b + c;
d = e + f;
```

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

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

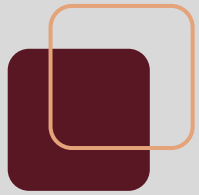
Estas dos instrucciones **no** pueden ser ejecutadas completamente en paralelo pues la segunda instrucción precisa el primer resultado para poder iniciar!

Machine Level Parallelism



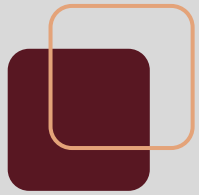
- Propiedad de un CPU. Indica qué tanto código puede ejecutar de forma paralela.
- Cuando un programa con ILP se ejecuta en un CPU con MLP, se acelera la ejecución.
- Hasta el momento se han estudiado procesadores que como máximo pueden ejecutar UNA instrucción por ciclo (CPI = 1)
- Los superescalares son procesadores que pueden lograr CPI < 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.
- Técnicas de Hardware (Dinámicas)
  - El propio hardware detecta el ILP en tiempo de ejecución y lo aprovecha.

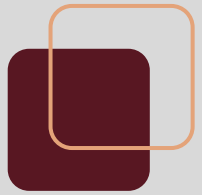
Técnicas Estáticas
Reordenar instrucciones (1/2)



- 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

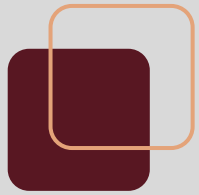
Técnicas Estáticas
Reordenar instrucciones (2/2)



- 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
Loop Unrolling (1/5)



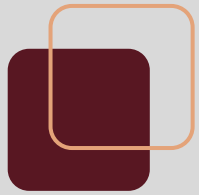
- En el mismo pipeline, considere el siguiente código MIPS:
  - loop: L.D F0, 0(R1) ;carga mem[r1]
    ADD.D F4, F0, F2 ;suma F2 (cte)
    S.D F4, 0(R1) ;guarda mem[r1]
    DADDUI R1, R1, #-08 ;avanza puntero
    BNE R1, R2, loop ;salta a loop
- ¿Cuánto demora la ejecución de cada loop?

Técnicas Estáticas
Loop Unrolling (2/5)

Instrucción\Ciclo	1	2	3	4	5	6	7	8	9	10
l.d F0, 0(R1)	IF	ID	EX	MEM	WB					
add.d F4, F0, F2			IF	ID	EX	MEM	WB			
s.d F4, 0(R1)				IF	ID	EX	MEM	WB		
daddui R1, R1, #-08					IF	ID	EX	MEM	WB	
bne R1, R2, loop						IF	ID	EX	MEM	WB
l.d F0, 0(R1)										IF

- Asumiendo forwarding, se demoran 10 ciclos en ejecutar el loop.

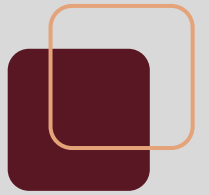
Técnicas Estáticas
Loop Unrolling (3/5)



- La técnica de Loop Unrolling consiste en reducir la cantidad de iteraciones ejecutadas, replicando el código dentro del loop:
  - loop:

```
L.D     F0, 0(R1)      ;carga mem[r1]
ADD.D   F4, F0, F2     ;suma F2 (cte)
S.D     F4, 0(R1)      ;guarda mem[r1]
DADDUI  R1, R1, #-08 ;avanza puntero
L.D     F0, 0(R1)      ;carga mem[r1]
ADD.D   F4, F0, F2     ;suma F2 (cte)
S.D     F4, 0(R1)      ;guarda mem[r1]
DADDUI  R1, R1, #-08 ;avanza puntero
BNE     R1, R2, loop  ;salta a loop
```

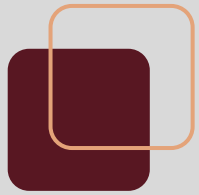
Técnicas Estáticas
Loop Unrolling (4/5)



Instrucción\Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14
l.d F0, 0(R1)	IF	ID	EX	MEM	WB									
add.d F4, F0, F2			IF	ID	EX	MEM	WB							
s.d F4, 0(R1)				IF	ID	EX	MEM	WB						
l.d F0, -8 (R1)						IF	ID	EX	MEM	WB				
add.d F4, F0, F2							IF	ID	EX	MEM	WB			
s.d F4, -8 (R1)								IF	ID	EX	MEM	WB		
daddui R1, R1, #-16									IF	ID	EX	MEM	WB	
bne R1, R2, loop										IF	ID	EX	MEM	WB
l.d F0, 0(R1)														IF

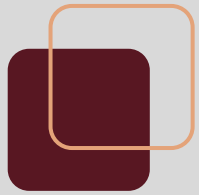
- En total se demora 14 ciclos en ejecutar DOS iteraciones.

Técnicas Estáticas
Loop Unrolling (5/5)



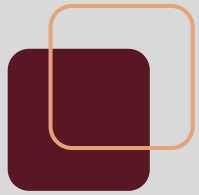
- Ventaja: Además de mejorar el código reduciendo el overhead de los saltos, loop unrolling permite reordenar las instrucciones de varias iteraciones.
- Desventaja: Aumenta el tamaño del código.
- Generalmente se debe además contemplar casos de borde.

Técnicas Estáticas: Conclusiones



- Permiten aumentar la performance sin tener que realizar mejoras en el procesador.
- Funcionan si se tiene conocimiento del hardware subyacente.
  - No es útil cuando se realizan distribuciones binarias.

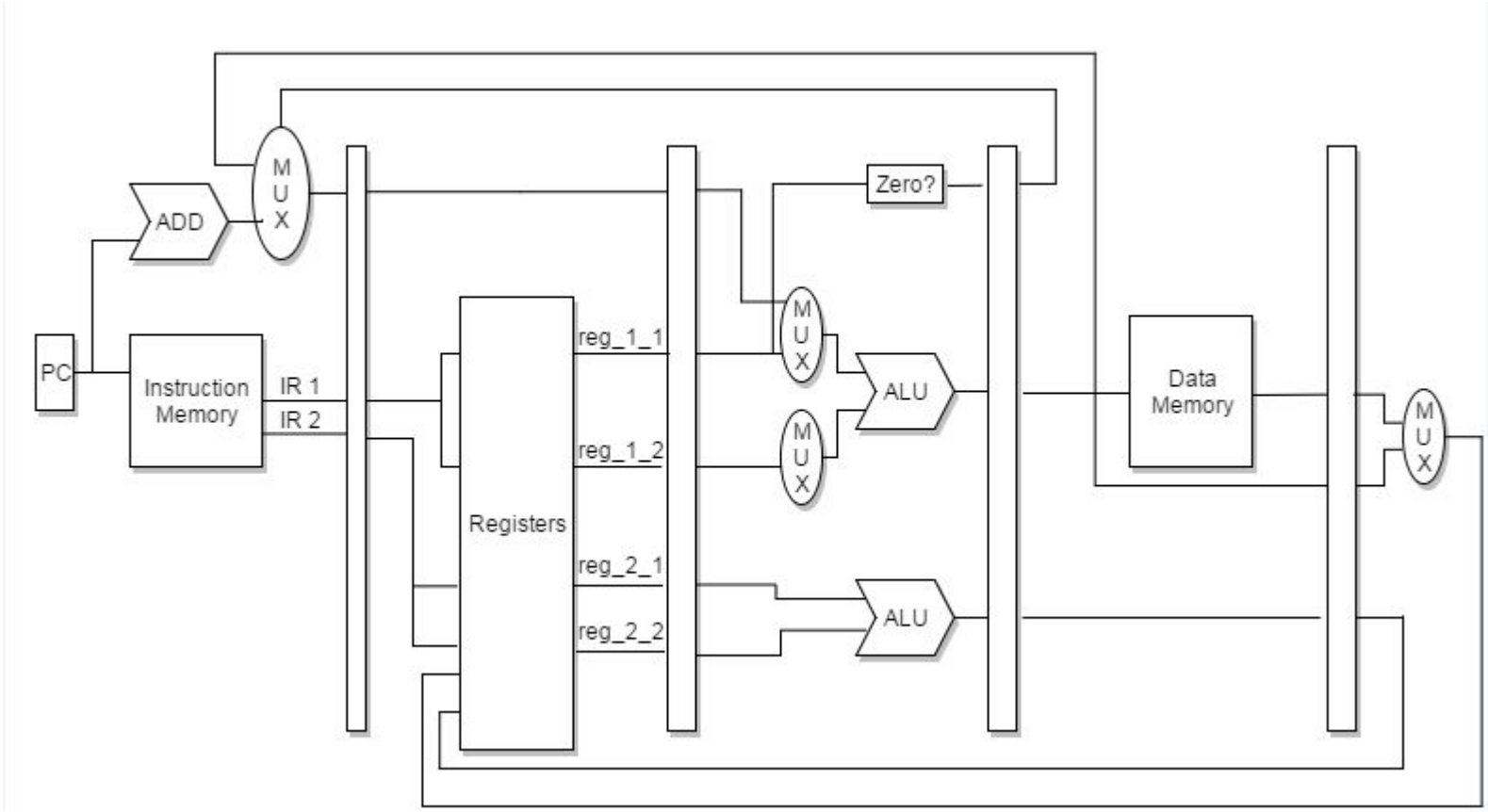
Organización Superescalar (1/2)



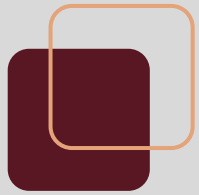
- La implementación superescalar implica como mínimo la capacidad de ejecutar dos instrucciones por ciclo. Un procesador con dichas características se denomina *two-wide superescalar* (superescalar de ancho 2)
- Un superescalar con posibilidad de ejecutar *N* instrucciones por ciclo es denominado *N-wide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

Organización Superescalar (2/2)

- Ejemplo BÁSICO 2-wide superscalar:

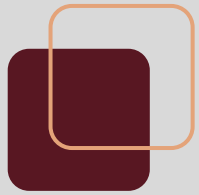


Recursos de Hardware (1/3)



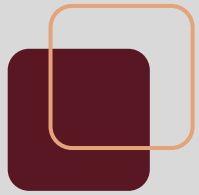
- ¿Qué recursos se requieren como mínimo para implementar un *2-wide superscalar?*
  - Fetch de (al menos) 2 instrucciones a la vez.
  - Cuatro puertos de lectura y dos de escritura en el banco de registros.
  - Lógica de despacho (issue logic)
  - Lógica de control duplicada
  - Al menos dos unidades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

Recursos de Hardware (2/3)
Fetch Logic



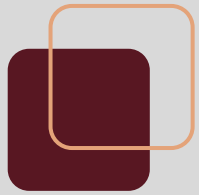
- Fetch de múltiples instrucciones en un mismo ciclo:
  - Bus más ancho entre CPU - Caché de Instrucciones L1
- Problemas:
  - Alineación de memoria
  - Fetch de múltiples bloques
  - Dependencias de control

Recursos de Hardware (3/3)
Fetch Logic



- Típicamente, se realiza el fetch de tamaños *fijos* del caché de instrucciones, y luego la lógica de *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, cuarto, bloque, etc.

Diagramas de Pipeline Superescalar



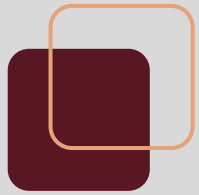
- Se levanta la restricción de que 'solo una instrucción puede estar en una etapa en el mismo ciclo'.
- Ejemplo *ideal* para el pipeline anterior:

	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,

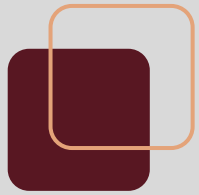
CPI = 1/2!

Forwarding (1/2)



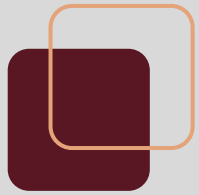
- ¿Cómo se implementa el forwarding?
  - De la misma manera que en el pipeline de 5 etapas, pero la cantidad de conexiones crece linealmente con el número de etapas del pipeline y el ancho del superescalar.
  - Multiplexor de 4 entradas para el pipeline MIPS de 5 etapas.
  - ¿Para el Intel Pentium 4,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

Forwarding (2/2)



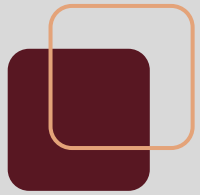
- ¿Solución?
  1. No implementar full-forwarding. Incluir solo conexiones más probables.
  2. Agregar otras restricciones (forwarding entre *algunos* pipelines).
  3. Agregar etapa dedicada y estructuras de datos especiales (ej: Scoreboard)
  4. Cambiar lógica distribuida:
     - Tomasulo (más adelante)

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.

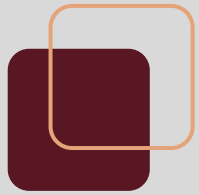
Scoreboard (1/8)



- Un scoreboard mantiene el control de qué registros están por ser escritos, a partir de las instrucciones que están en las unidades funcionales.

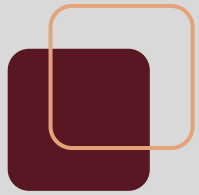
Registro	Pendiente	Unidad Funcional	Dato Calculado	Ciclo Disponible				
				N	N-1	...	1	0
R1	0	-	-	-	-	-	-	-
R2	1	EX	1	0	0	...	1	0
...								
R30	1	MUL	0	0	1	...	0	0
R31	0	-	-	-	-	-	-	-

Scoreboard (2/8)



- *Registro*: el registro del que se lleva registro en la fila
- *Pendiente*: indica si el registro está disponible para ser leído desde el banco de registros (0) o si su valor actual está siendo calculado por una instrucción(1).
- *Functional Unit (FU)*: Indica en qué unidad funcional se está calculando.
- *Dato Calculado*: Indica que el dato ya se terminó de calcular (se puede realizar forwarding)

Scoreboard (3/8)



- Ciclo disponible: Esta parte del scoreboard es un *shift register*. Se mantiene una columna por cada etapa del pipeline desde la etapa *issue* hasta la etapa writeback. Cuando una instrucción entra en la unidad funcional, se coloca un 1 en el ciclo correspondiente. Luego es desplazado hacia la derecha en cada ciclo, indicando en cada momento dónde se encuentra el último valor disponible del registro.

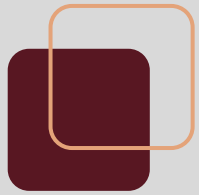
Scoreboard (4/8)

- Ejemplo para el pipeline de 5 etapas. Al entrar la instrucción en la unidad funcional, se marca como pendiente. Al calcularse el resultado, se indica que está calculado (columna DC). En el ejemplo, etapa '0' indica writeback, etapa '1' indica MEM, etc.

Scoreboard

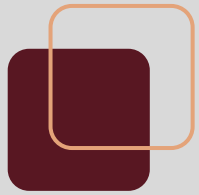
	1	2	3	4	5	Reg	Pend?	FU	DC	2	1	0
subu R1,R3, R5	IF	ID	EX	MEM	WB	R1	0	-	0	-	-	-
	1	2	3	4	5	Reg	Pend?	FU	DC	2	1	0
subu R1,R3, R5	IF	ID	EX	MEM	WB	R1	1	EX	0	1	0	0
	1	2	3	4	5	Reg	Pend?	FU	DC	2	1	0
subu R1,R3, R5	IF	ID	EX	MEM	WB	R1	1	EX	1	0	1	0
	1	2	3	4	5	Reg	Pend?	FU	DC	2	1	0
subu R1,R3, R5	IF	ID	EX	MEM	WB	R1	1	EX	1	0	0	1

Scoreboard (5/8)



- El uso de Scoreboard simplifica enormemente la lógica de forwarding, ya que se reutiliza la lógica de decodificación/issue.
- Una conexión de forwarding pasa a estar activa sí:
  - El registro de lectura y escritura coinciden.
  - El registro está marcado como pendiente Y disponible en el scoreboard.
  - La etapa desde donde se inicia la conexión de forwarding está marcada con 1 para el registro.

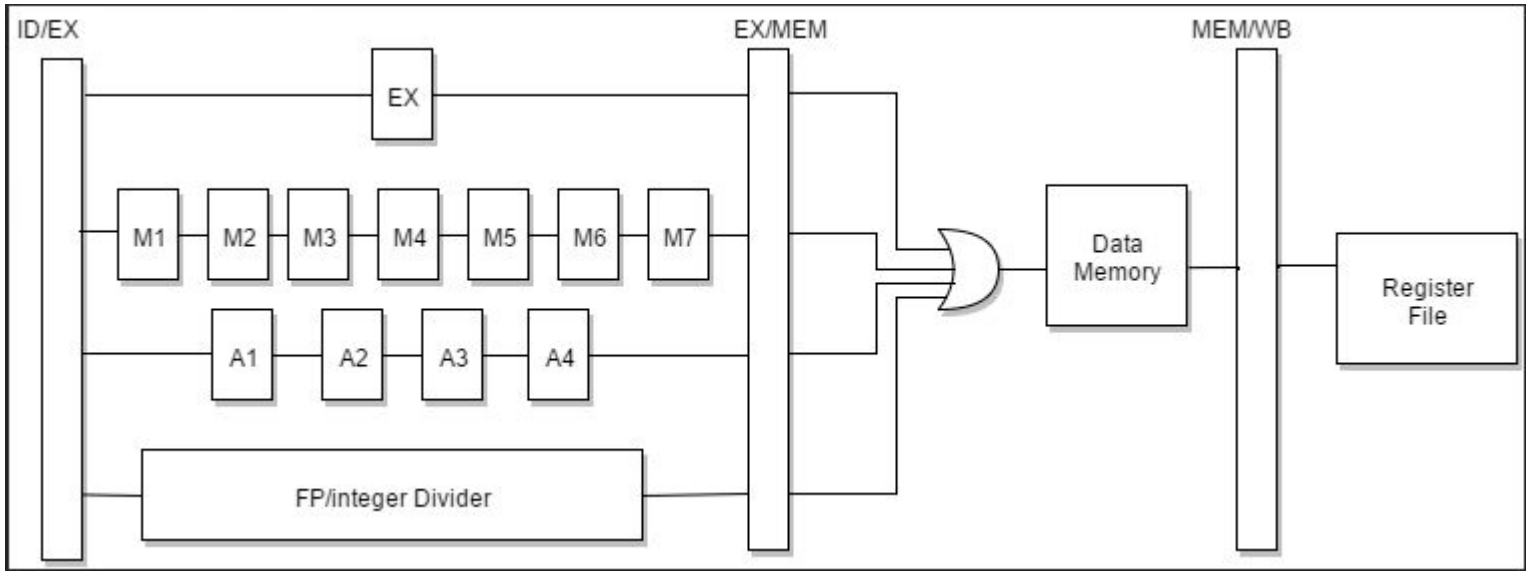
Scoreboard (6/8)



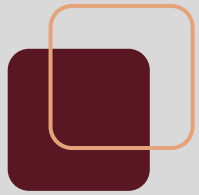
- Para evitar hazards RAW, se debe cumplir ALGUNA de las siguientes condiciones al momento de pasar la instrucción a la etapa *execute*.
  - Los registros a leer (origen) están disponibles (pendiente = 0)
  - Los registros a leer están pendientes pero calculados (dato calculado = 1)
  - Los registros a leer están pendientes y estarán disponibles en el próximo ciclo (se puede deducir de la unidad funcional y ciclo en el que se encuentren)
  - En cualquier otro caso se debe detener la instrucción en la etapa decode/issue.

Scoreboard (7/8)

- En pipelines más complejos, el Scoreboard puede ser utilizado para resolver otros problemas.
- Supongamos el pipeline con múltiples unidades funcionales y un solo puerto de escritura en el banco de registros:

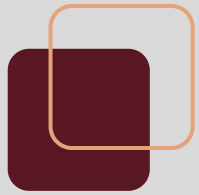


Scoreboard (8/8)



- Dos problemas:
  - Hazard estructural en etapa de memoria/writeback
  - Hazard de datos WAW
- ¿Cómo se puede utilizar la información del Scoreboard para solucionar estos problemas?
  - Ver práctico :)

Fin



¿Preguntas?