

# ILP y Pipeline

## Lección 8

Prof.Ing. Fabián Zamora Ramírez

CE-4301 Arquitectura de Computadores I

*Área de Ingeniería en Computadores*

*Instituto Tecnológico de Costa Rica*

# Agenda

- 1 ILP
- 2 Segmentación
- 3 Riesgos

# Paralelismo

Técnica de programación e implementación en la que se pretende realizar operaciones simultáneamente, con el fin de reducir tiempos de ejecución, en un procesador.

Existen diferentes tipos de paralelismo:

- Paralelismo a nivel de bit
- Paralelismo a nivel de instrucciones
- Paralelismo a nivel de datos
- Paralelismo a nivel de tareas
- Paralelismo a nivel de hilos

# Paralelismo a nivel de instrucción (ILP)

Técnica de **paralelismo** basada en la ejecución simultánea de instrucciones.

Posee dos enfoques:

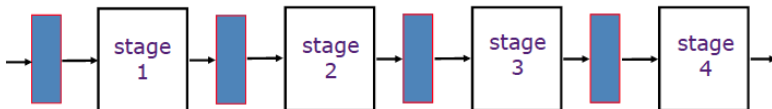
- Paralelismo por hardware (dinámico) - ejecución
- Paralelismo por software (estático) - compilación

Tipos de ILP:

- Segmentación Pipeline
- Ejecución fuera de orden (OoOE)
- VLIW
- CPU's superescalares

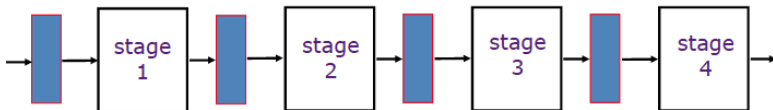
# Segmentación - Pipeline

Técnica utilizada en el diseño de CPUs para aumentar el rendimiento, mediante la separación de las etapas en el proceso de ejecución de una instrucción.



# Segmentación - Pipeline

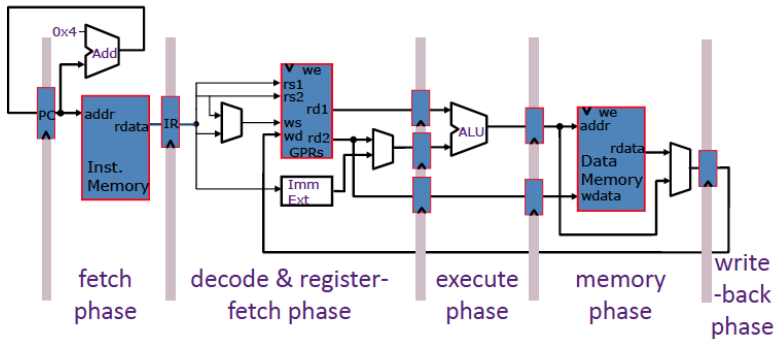
Técnica utilizada en el diseño de CPUs para aumentar el rendimiento, mediante la separación de las etapas en el proceso de ejecución de una instrucción.



# Supuestos Pipeline

- Todas las instrucciones pasan por todas las etapas.
- Las etapas no comparten recursos de hardware entre sí.
- El tiempo de propagación entre las etapas es el mismo.
- Las instrucciones son independientes entre sí.
- Las etapas se puede aislar temporalmente.

# Segmentación - MIPS





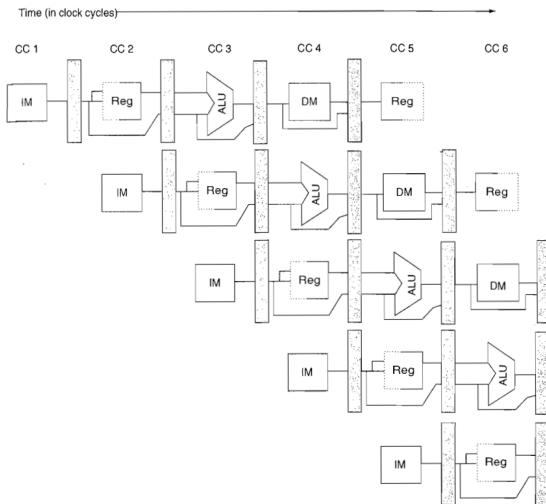
# Etapas básicas de un pipeline

- Búsqueda de instrucción (**IF**): Enviar el PC a memoria, traer nueva instrucción, actualizar el PC.
- Decodificación de instrucción (**ID**): "Traducción de instrucción", lectura de registros operandos.
- Ejecución /Dir efectiva (**Ex**): Operaciones en ALU: Memoria efectiva, R-R, R-I.
- Acceso a memoria (**MEM**): Instrucciones L/S.
- Escritura a registros (**WB**): Escritura de resultados R-R o instrucciones L/S a banco de registros.

# Pipeline Ideal

Instruction number	Clock 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

# Uso de hardware



# Ganancia en desempeño

La ganancia en el desempeño debido al pipeline está dada por.

$$\begin{aligned}\text{Speedup from pipelining} &= \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}} \\ &= \frac{\text{CPI unpipelined} \times \text{Clock cycle unpipelined}}{\text{CPI pipelined} \times \text{Clock cycle pipelined}} \\ &= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}\end{aligned}$$

# Diseño de un ISA pensando en segmentación

- **Tamaño de instrucciones.** Instrucciones del mismo tamaño hace mucho mas fácil la etapa de fetch y decode.
- **Formato de instrucciones.** Pocos formatos de instrucciones con los operandos en las mismas posiciones. Esto implica que en decode se pueden leer los operandos al mismo tiempo que se determina el tipo de instrucción.
- **Load/Store.** Esto permite que la etapa de ejecución se utilice para calcular la dirección efectiva en memoria.
- **Alineamiento de memoria.** Las instrucciones de acceso a memoria no requieren multiples accesos. Los datos se pueden obtener en una sola etapa.

# Ventajas y desventajas

## Ventajas

- Aumenta el rendimiento del CPU.
- Brinda determinismo en ejecución de instrucciones.

## Desventajas

- Etapas e instrucciones lentas afectan el rendimiento general
- Mayor complejidad, más hardware.
- Latencia es ligeramente mayor.
- **Riesgos**

# Riesgos en la segmentación

**Riesgo:** Situación que previene que la siguiente instrucción pueda ser ejecutada en el ciclo de reloj correspondiente.

- Riesgos estructurales: conflictos de hardware entre instrucciones.
- Riesgos de datos: Dependencias **reales** entre datos de instrucciones
- Riesgos de control: saltos y branches.

Los riesgos reducen el desempeño ideal ganado por la técnica de pipeline.

# Stalls

Los riesgos provocan que el pipeline se *detenga* (**stall**)

- Las instrucciones calendarizadas antes de la instrucción detenida deben terminar su ejecución.
- Las instrucciones calendarizadas después de la instrucción detenida deben ser detenidas igualmente.

Se debe tomar en cuenta el tiempo detención por instrucción:

$$\begin{aligned}\text{CPI pipelined} &= \text{Ideal CPI} + \text{Pipeline stall clock cycles per instruction} \\ &= 1 + \text{Pipeline stall clock cycles per instruction}\end{aligned}$$



# Unidades Funcionales - FU

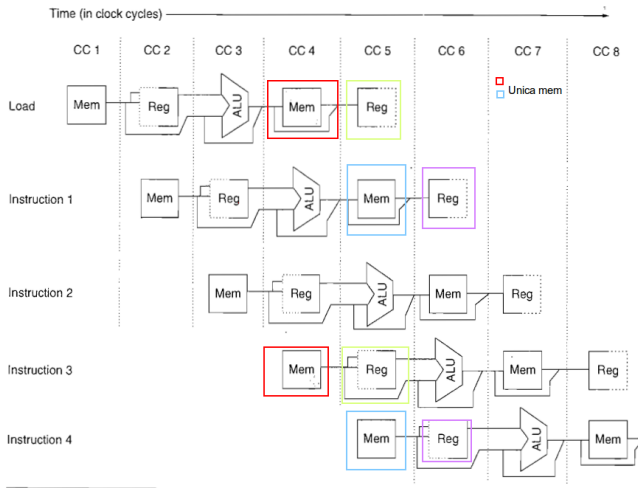
Una unidad funcional es un elemento, dentro del hardware de un procesador, que realiza una función específica: Ejemplos:

- ALU
- fpALU
- Multiplicadores
- Comparadores
- MAC
- etc.

# Riesgos Estructurales

En un procesador con pipeline se requieren unidades funcionales duplicadas para alojar recursos en todas las posibles combinaciones de instrucciones. Cuando no hay recursos necesarios para evitar conflictos en uso de hardware se tiene un **riesgo estructural**.

# Riesgos Estructurales



# Posibles Soluciones

# Posibles Soluciones

Uso de stalls.

- Solución simple
- Disminuye el rendimiento  $\rightarrow$  1 ciclo más

# Posibles Soluciones

Uso de stalls.

- Solución simple
- Disminuye el rendimiento  $\rightarrow$  1 ciclo más

Duplicar hardware

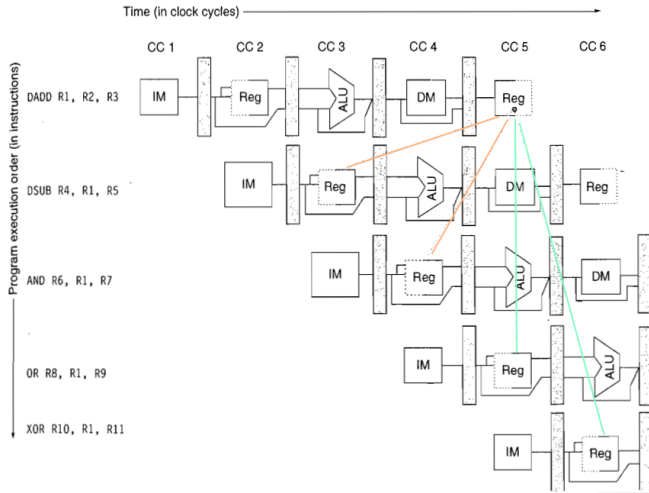
- Solución más compleja. Puede requerir lógica de control adicional
- Puede ser cara (más hardware, más potencia)
- No disminuye el desempeño

# Riesgos de datos

Los **riesgos de datos** ocurren cuando en el pipeline se cambia el orden de acceso a lectura/escritura de operandos, de forma que el orden difiere de la ejecución secuencial en un procesador sin pipeline.

DADD	R1,R2,R3
DSUB	R4,R1,R5
AND	R6,R1,R7
OR	R8,R1,R9
XOR	R10,R1,R11

# Riesgos de datos





# Posibles Soluciones

# Posibles Soluciones

Uso de stalls.

- Solución simple
- Disminuye el rendimiento -> requiere más de 1 ciclo adicional

# Posibles Soluciones

Uso de stalls.

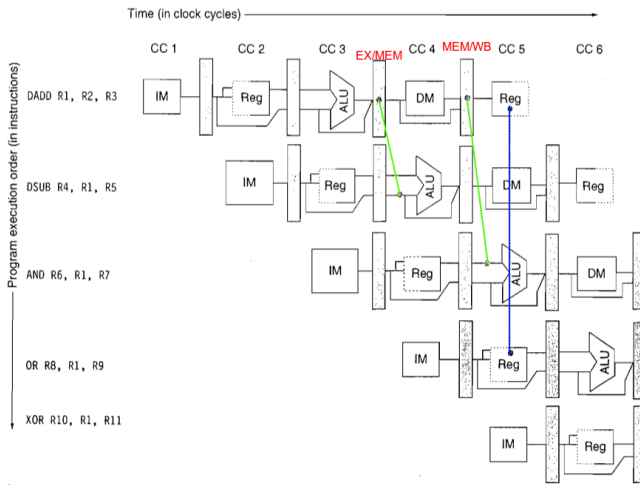
- Solución simple
- Disminuye el rendimiento -> requiere más de 1 ciclo adicional

**Adelantamiento:** Consiste en mover el resultado de un registro directamente hacia la siguiente etapa donde se necesita, si esperar al WB.

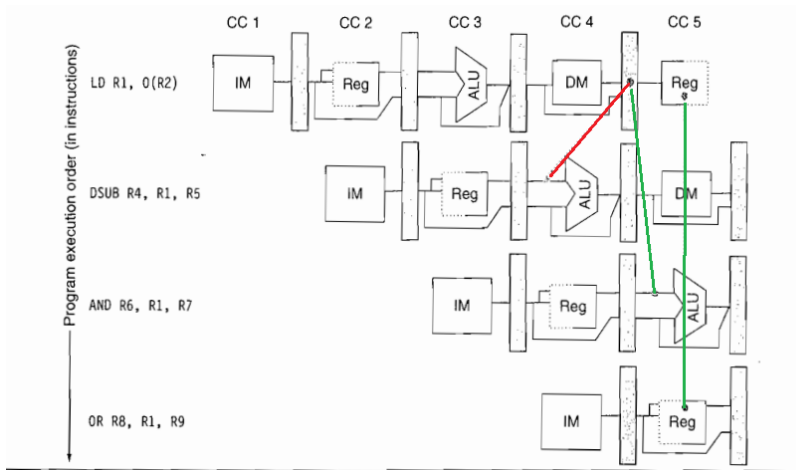
# Adelantamiento

- El resultado de la ALU de los registros EX/MEM y MEM/RB se realimenta a las entradas de la ALU.
- El control selecciona, en caso de detectar adelantamiento, las entradas correctas de la ALU.
- No es solución definitiva, en algunos casos no funciona. Esto es común en instrucciones LD, donde la carga se realiza al final del ciclo.

# Adelantamiento



# Limitación Adelantamiento



# Adelantamiento + Stall

LD	R1,0(R2)	IF	ID	EX	MEM	WB			
DSUB	R4,R1,R5		IF	ID	EX	MEM	WB		
AND	R6,R1,R7			IF	ID	EX	MEM	WB	
OR	R8,R1,R9				IF	ID	EX	MEM	WB
LD	R1,0(R2)	IF	ID	EX	MEM	WB			
DSUB	R4,R1,R5		IF	ID	stall	EX	MEM	WB	
AND	R6,R1,R7			IF	stall	ID	EX	MEM	WB
OR	R8,R1,R9				stall	IF	ID	EX	MEM WB

# Riesgos de control

Los **riesgos de control** ocurren cuando al tener la ejecución de una instrucción *branch* se puede modificar o no el valor del PC, alterando el flujo de ejecución del programa.

- Dependiendo de si el salto se toma o no (etapa ID), la siguiente instrucción será o no la correcta.



# Posibles Soluciones

# Posibles Soluciones

Stall de un ciclo.

- Después de cada salto se realiza dos IF

# Posibles Soluciones


Stall de un ciclo.

- Después de cada salto se realiza dos IF

**Predicción de salto:** Estrategia basada en métodos estadísticos o probabilistas para tratar de predecir el si salto es tomado o no  
En caso de fallar la predicción, se debe vaciar (flush) el pipeline.

# Riesgos de control

## Stall

Branch instruction	IF	ID 	EX	MEM	WB		
Branch successor		IF	IF	ID	EX	MEM	WB
Branch successor + 1				IF	ID	EX	MEM
Branch successor + 2					IF	ID	EX

## Predicción

Untaken branch instruction	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
Instruction $i + 4$					IF	ID	EX

Taken branch instruction	IF	ID	EX	MEM	WB		
Instruction $i + 1$		IF	idle	idle	idle	idle	flush
Branch target			IF	ID	EX	MEM	WB
Branch target + 1				IF	ID	EX	MEM
Branch target + 2					IF	ID	EX

# Referencias



J Hennesy and David Patterson (2005)

Computer Organization and Design. The hardware/software interface. 4th Edition. [Cap 4]



J Hennesy and David Patterson (2012)

Computer Architecture: A Quantitative Approach. 5th Edition. Elsevier – Morgan Kaufmann. [Cap 3]



Jeferson González G. (2017)

Material de clase: Arquitectura de computadores I.