

TALLER DE SISTEMAS DIGITALES

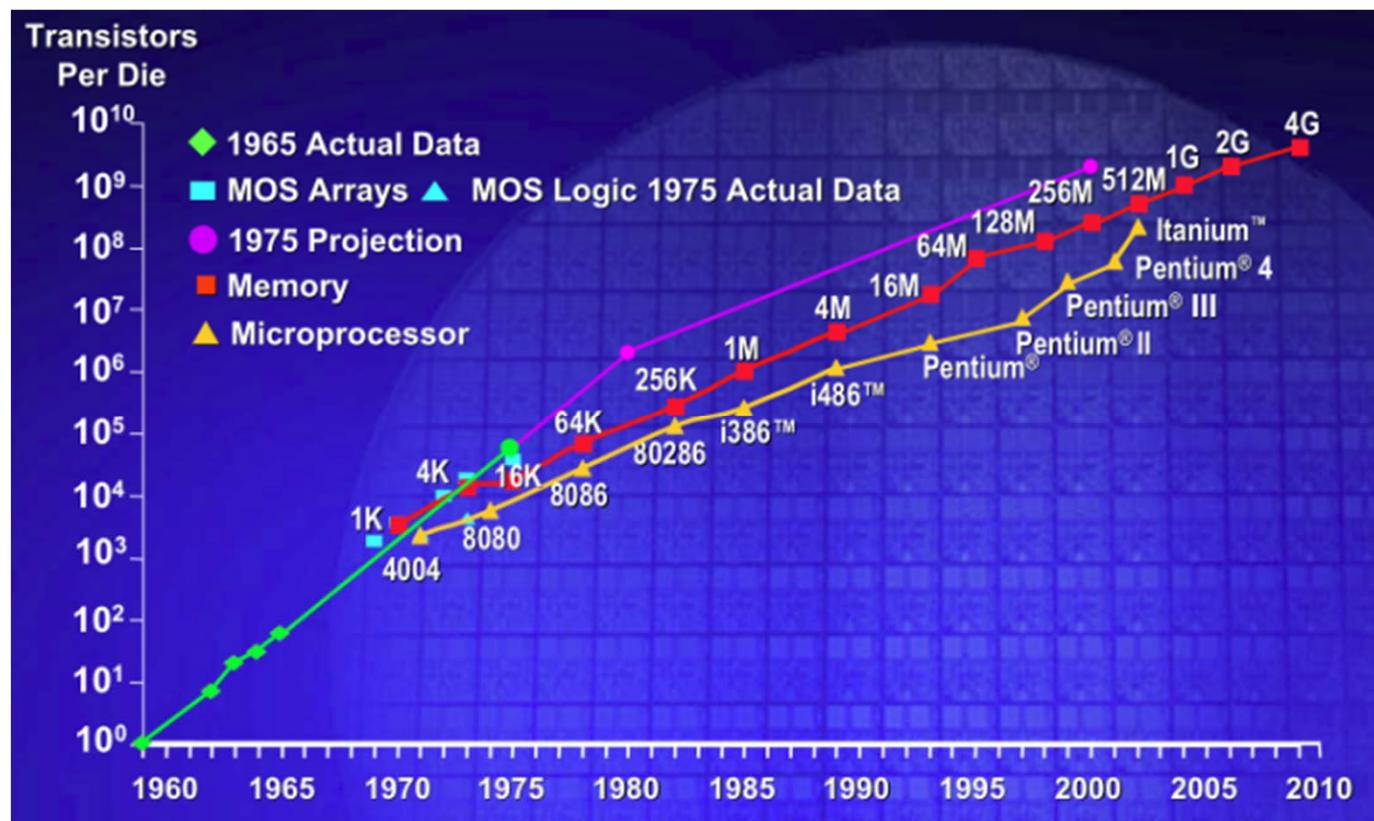
ARQUITECTURA DE COMPUTADORAS I

- Ley de Amdahl
- Pipelines



MEJORA EN EL DESEMPEÑO DE MICROPROCESADORES

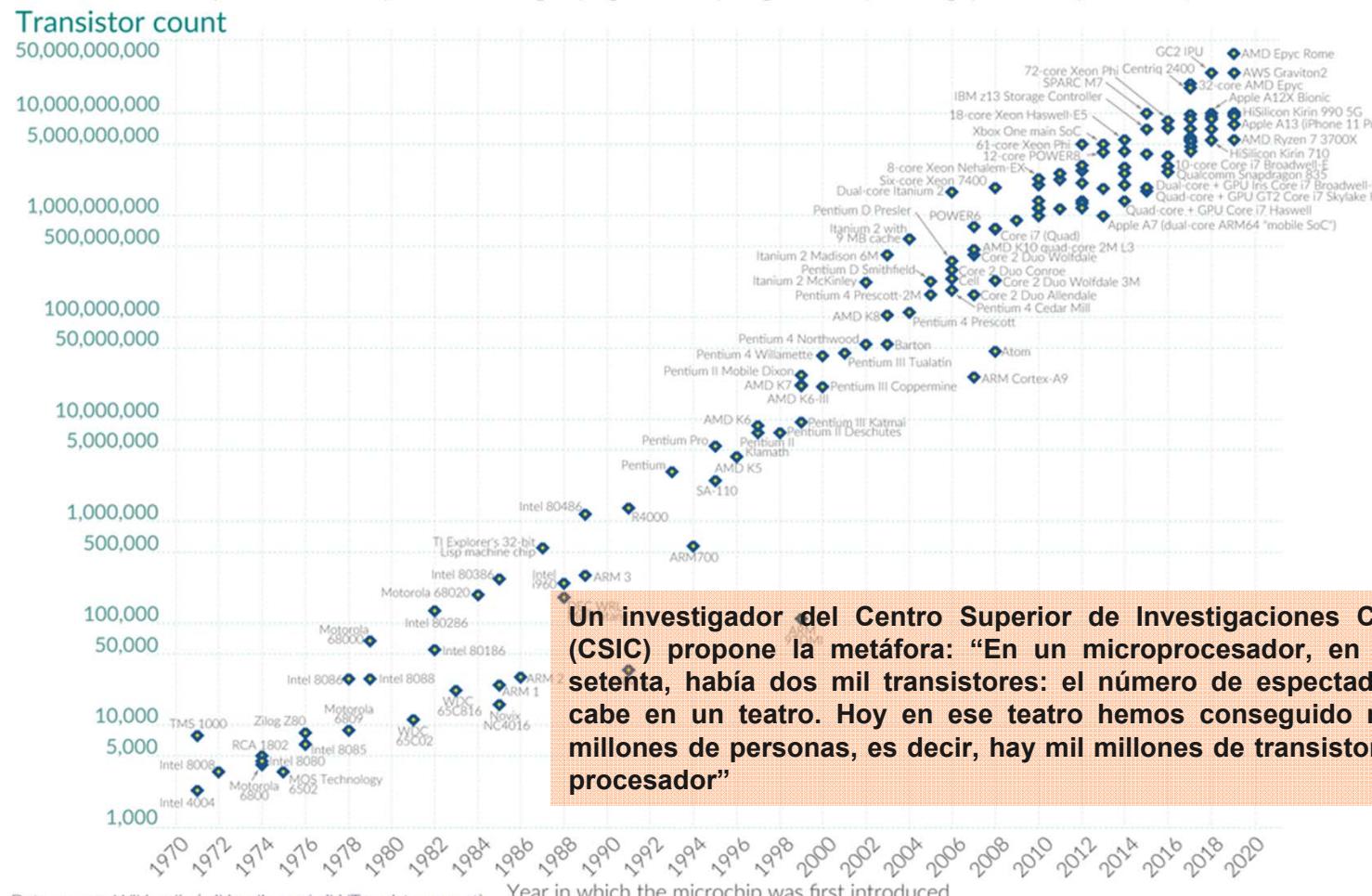
Ley de Moore



MEJORA EN EL DESEMPEÑO DE MICROPROCESADORES

Moore's Law: The number of transistors on microchips doubles every two years

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Un investigador del Centro Superior de Investigaciones Científicas (CSIC) propone la metáfora: “En un microprocesador, en los años setenta, había dos mil transistores: el número de espectadores que cabe en un teatro. Hoy en ese teatro hemos conseguido meter mil millones de personas, es decir, hay mil millones de transistores en un procesador”

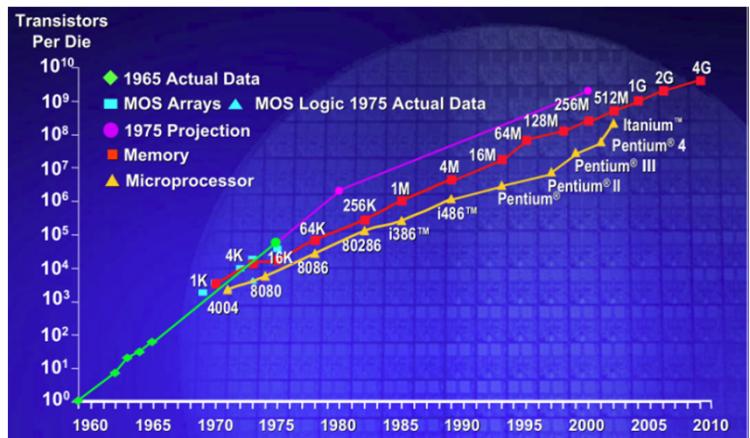
Data source: Wikipedia ([wikipedia.org/wiki/Transistor_count](https://en.wikipedia.org/w/index.php?title=Transistor_count&oldid=1000000000))

OurWorldInData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

MEJORA EN EL DESEMPEÑO DE MICROPROCESADORES

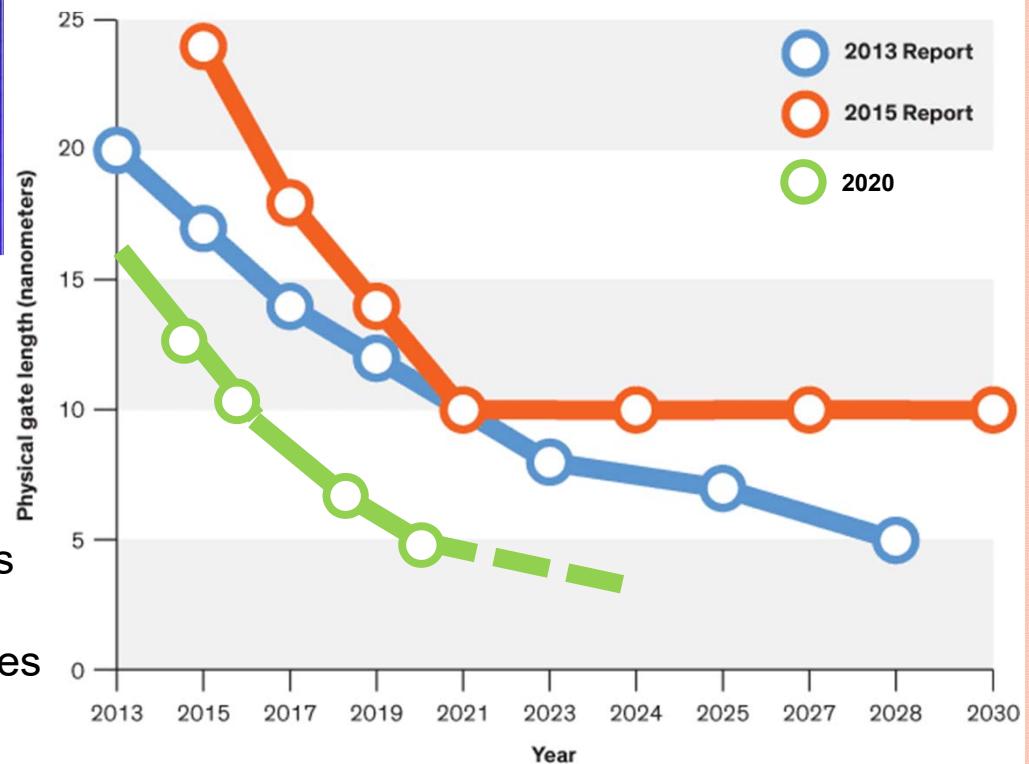
Ley de Moore



Pentium → 3.10^6 Transistores
Pentium 4 → 125.10^6 Transistores
Nehalem Core i7 → 700.10^6 Transistores

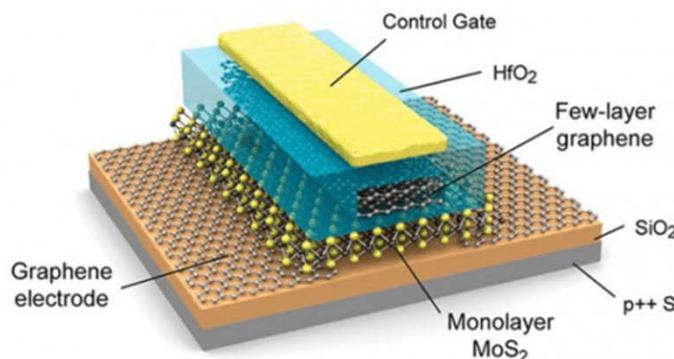
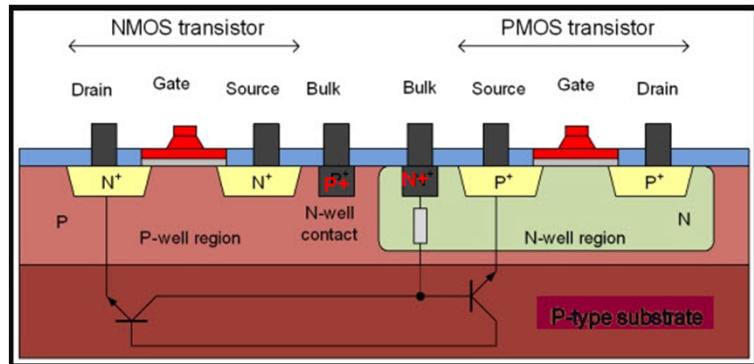
Futuro Intel → 30 Billones de Transistores en 2030

Especulaciones recientes en la miniaturización

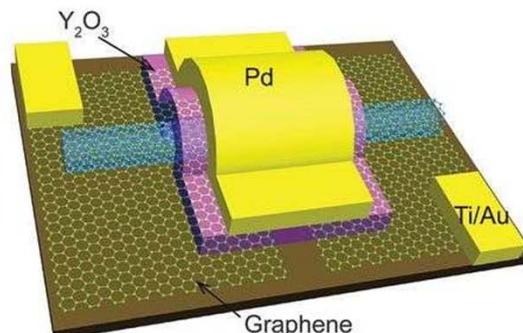


MEJORA EN EL DESEMPEÑO DE MICROPROCESADORES

Transistores Tecnología CMOS



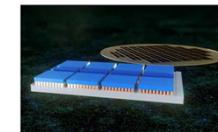
Nuevas tecnologías, materiales y formas



Relentlessly Pursuing
Moore's Law
Advancing and Accelerating Computing

Components Research, the research group of Intel Technology Development, is responsible for delivering revolutionary process and packaging technologies that extend Moore's Law and enable Intel products and services.

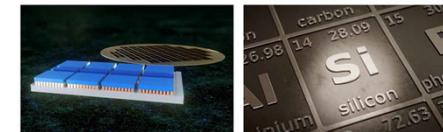
At the 67th Annual IEEE International Electron Devices Meeting, Components Research is presenting key breakthroughs in three areas of research for delivering the fundamental building blocks for more powerful computing well into the future:



Delivering More Transistors

Essential scaling technologies for delivering more transistors include making them smaller and smaller, so we can deliver millions more per square area and deliver them as tiles, or chipslets, via advanced packaging.

- Interconnect density improvement in packaging
- Area improvement with 3D transistor stacking
- Super thin materials for future scaling



Bringing New Capabilities to Silicon

As we enable more powerful computing through scaling, we need to stretch the limits of silicon and integrate new materials – we can deliver power more efficiently and meet greater demands for memory.

- World's first integration of GaN-based power switches
- Record FeRAM speed and endurance



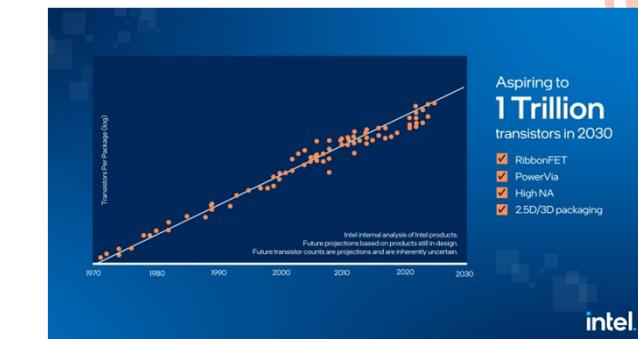
Embracing the Quantum Realm

We are exploring entirely new concepts in physics that may one day revolutionize computing by potentially replacing classic transistors, enabling even greater performance and power efficiencies.

- Metamaterials
- Spin-torque devices
- 300nm qubit process flows

Learn more at <https://intel.ly/31PxF9p>

© Intel Corporation. Intel, the Intel logo, and other Intel trademarks and service marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.



MEJORA EN EL DESEMPEÑO DE MICROPROCESADORES

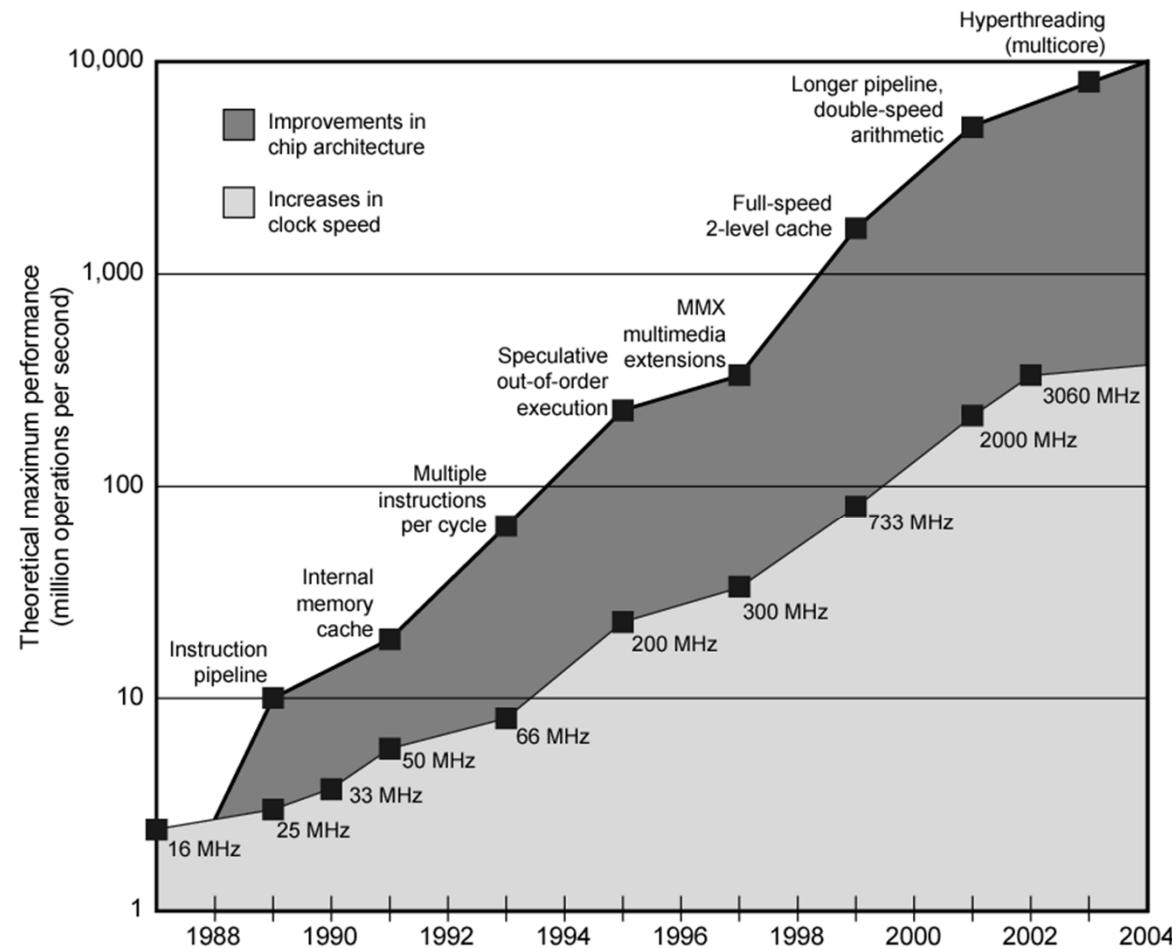


Figure 2.12 Intel Microprocessor Performance [GIBB04]

Benchmarks

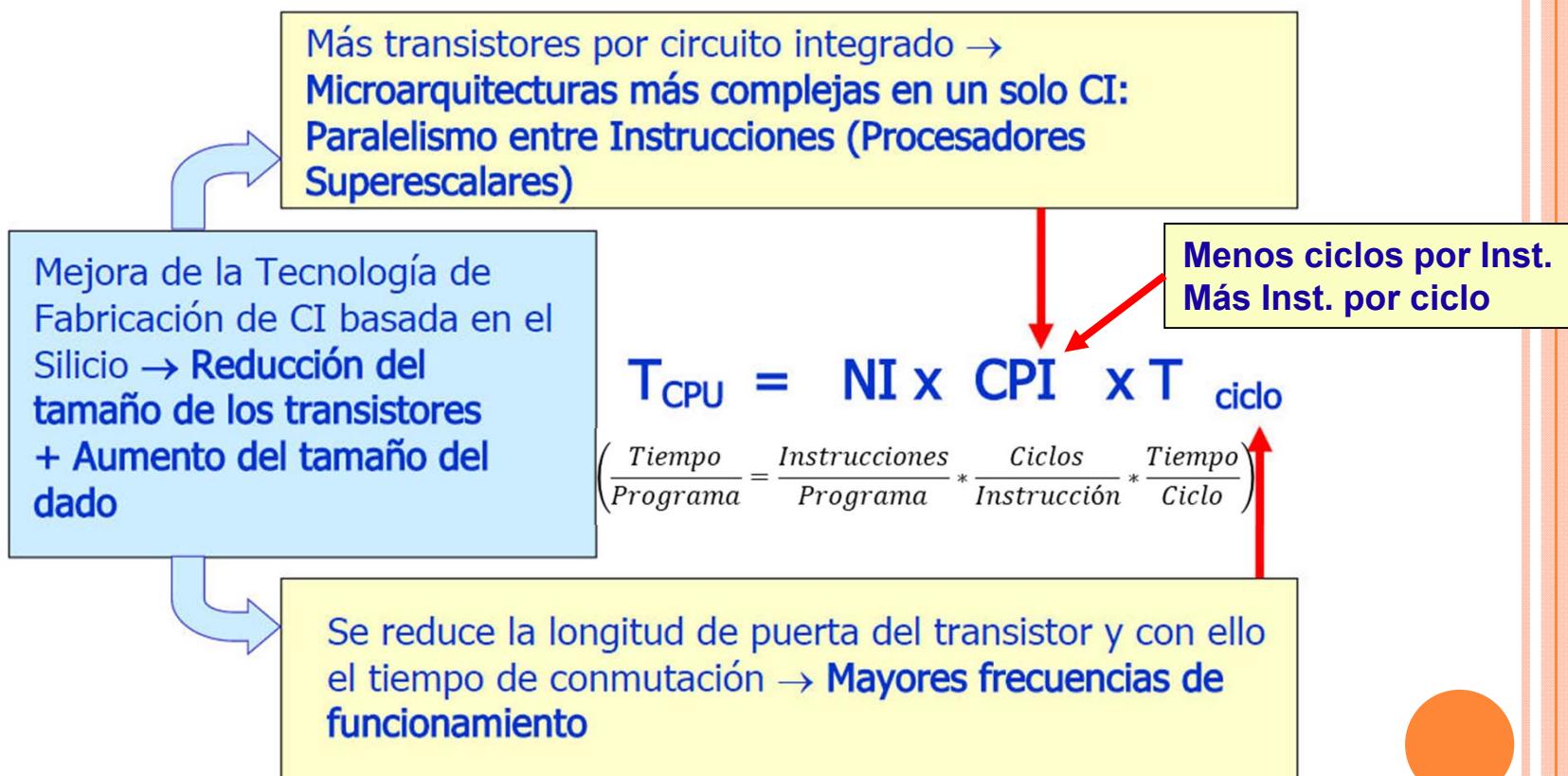
TASA DE EJECUCIÓN DE INSTRUCCIONES

- Clock que funciona a frecuencia constante f
- Se define la **cantidad de instrucciones I_c**, para un programa como el número de instrucciones de máquina **ejecutadas** por ese programa hasta que se termine o para algún intervalo de tiempo definido.
- Tener en cuenta que este es el número de instrucciones ejecutadas, no el número de instrucciones en el código objeto del programa.
- Un parámetro importante es el **promedio de ciclos por instrucción de un programa CPI**.
- Si todas las instrucciones requirieran el mismo número de ciclos de reloj, entonces CPI sería un valor constante para un procesador. (analizar RISC vs CISC y el desafío de los programadores)

CALCULO DE TASA DE MIPS

(MILLONES DE INSTRUCCIONES POR SEGUNDO)

Influencia de la Tecnología



CALCULO DE TASA DE MIPS

(MILLONES DE INSTRUCCIONES POR SEGUNDO)

$\tau = 1/f \rightarrow$ ciclo de máquina o clock

I_c instrucciones se ejecutan en un tiempo $T = I_c \cdot CPI \cdot \tau$ (segundos)

$T = (\text{total de instrucciones} \times \text{promedio de ciclos por instrucción} \times \text{tiempo de un ciclo})$

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

Otro Benchmarks (repasar):

MFLOPS (millones de operaciones de punto flotante por segundo),
Dhrystone, DMIPS, Whetstone, Speedup, Eficiencia, Productividad, etc.

Probar SiSoft SANDRA (system analyser diagnostic and reporting assistant)

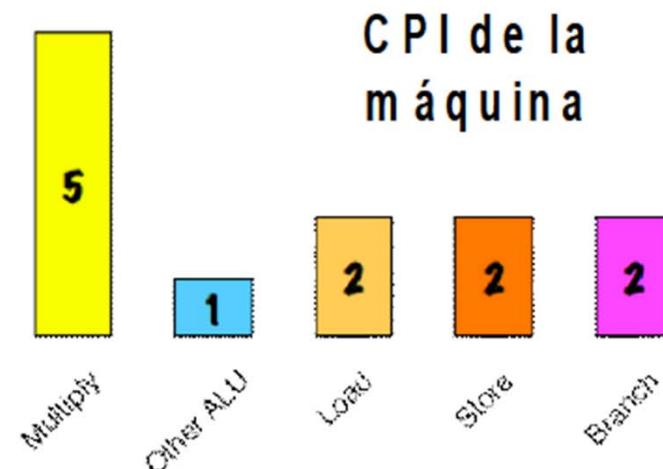
CPI

- Si CPI_i es el número de ciclos necesarios para la instrucción de tipo i.
- I_i es el número de instrucciones ejecutadas de tipo i para un programa dado.
- Se puede calcular el CPI promedio como se muestra a continuación

$$CPI = \frac{\sum_{i=1}^n (CPI_i \times I_i)}{I_c}$$

- Ver Pagina 51 del libro de William Stallings

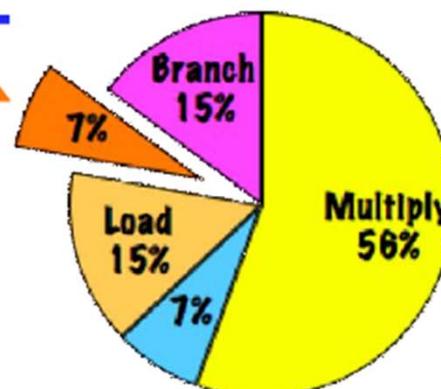
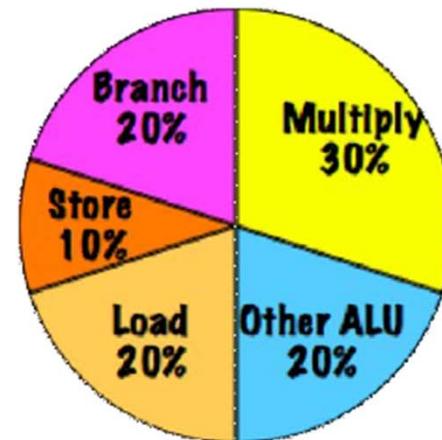
EJEMPLO CALCULO DE CPI



$$5 \times 30 + 1 \times 20 + 2 \times 20 + 2 \times 10 + 2 \times 20$$

$$\frac{100}{100} = 2.7 \text{ ciclos/instrucción}$$

Mezcla de Instrucciones



Donde se gasta el tiempo?



LEY DE AMDAHL

- Gene Amdahl 1967
- Potencial velocidad de ejecución de un programa usando multiples procesadores
- Se concluyó que:
 - El Código tiene que ser paralelizable
 - La velocidad está relacionada, dando rendimientos decrecientes para más procesadores.
- Dependencia de Tareas aplicadas a servidores
 - Los servidores ganan manteniendo multiples conexiones con varios procesadores.
 - Las bases de datos pueden separarse en tareas paralelas



LEY DE AMDAHL

- Para un programa corriendo en un procesador simple, se define:
 - f: Fracción de código infinitamente paralelizable sin sobrecarga de programación
 - $(1-f)$: fracción de código no paralelizable (**CUIDADO:** algunos autores asumen en forma inversa - siempre $f+(1-f)=1$)
 - T: tiempo total de ejecución del programa en un simple procesador
 - N: es el número de procesadores que aprovechan al máximo las porciones de código paralelizables

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

○ Conclusiones

- Para f pequeña, los procesadores paralelos tienen poco efecto
- Si $f \rightarrow 1$ la mejora o Speedup tiende a N
- $N \rightarrow \infty$, la velocidad tiende a $1/(1-f)$
- Rendimiento decreciente con el uso de muchos procesadores (curva como ejercicio)



ACLARACIONES DE AMDAHL

- Se afirmó en un congreso que estas conclusiones eran demasiado pesimistas, dando como ejemplo, que un servidor puede mantener varios subprocessos o tareas múltiples para manejar múltiples clientes y ejecutar los hilos o tareas en paralelo hasta el límite del número de procesadores. Muchas aplicaciones de base de datos implican cálculos con cantidades masivas de datos que se pueden dividir en múltiples tareas en paralelo. (punto de vista de Gustafson)

$$S_s(p) = \frac{ft_s + (1-f)t_s}{(1-f)t_s + \frac{ft_s}{p}} = \frac{p + (1-p)(1-f)}{1} = p + (1-p)(1-f)$$



ACLARACIONES DE AMDAHL

Tiempo de ejecución serie constante = Tamaño de problema constante

Ley de Amdahl :

$$\text{Speedup} = \frac{\text{time to execute program on a single processor}}{\text{time to execute program on } N \text{ parallel processors}} = \frac{T(1-f) + Tf}{T(1-f) + \frac{Tf}{N}} = \frac{1}{(1-f) + \frac{f}{N}}$$

Ley de Gustafson-Barsis :

$$S_s(p) = \frac{ft_s + (1-f)t_s}{(1-f)t_s + \frac{ft_s}{p}} = \frac{p + (1-p)(1-f)}{1} = p + (1-p)(1-f)$$

Speedup escalado

Tiempo de ejecución paralelo constante = Tamaño de problema crece con el número de procesadores

Con parte en serie = 5% y 20 procesadores → S (Amdahl) = 10.25
S (Gustafson) = 19.05

GENERALIZACIÓN DE LA LEY DE AMDAHL

- Sin embargo la ley de Amdahl ilustra los problemas que enfrenta la industria en el desarrollo de procesadores multi-núcleo, en máquinas con un número cada vez mayor de núcleos, El software que se ejecuta en máquinas de este tipo debe adaptarse a un entorno de ejecución altamente paralela para aprovechar la potencia del procesamiento paralelo.
- La ley de Amdahl puede ser generalizada para evaluar cualquier diseño o mejora técnica en un sistema informático. Considere cualquier mejora de la característica de un sistema que resulta en un aumento de velocidad. La aceleración se puede expresar como:

$$\begin{aligned}\text{Speedup} &= \frac{\text{Performance después de la mejora}}{\text{Performance antes de la mejora}} = \\ &= \frac{\text{Tiempo de ejecución antes de la mejora}}{\text{Tiempo de ejecución después de la mejora}}\end{aligned}$$



FORMULA DE LA LEY DE AMDAHL

Supongamos que una característica del sistema se utiliza durante la ejecución de una fracción del tiempo f , antes de la mejora, y que el aumento de velocidad de esa característica, después de la mejora K , es:

$$S = \frac{T_s}{(1-f)T_s + \frac{fT_s}{K}} = \frac{1}{(1-f) + \frac{f}{K}}$$

$$S = \frac{T_s}{fT_s + \frac{(1-f)T_s}{K}} = \frac{1}{f + \frac{(1-f)}{K}} \quad f + (1-f) = 1$$

EJEMPLO LEY AMDAHL

- Por ejemplo, supongamos que una tarea hace un amplio uso de operaciones de punto flotante, hasta un 40% del tiempo total consumido .
- Con un nuevo diseño de hardware (o puede ser Software), el módulo de punto flotante se acelera en un factor K. entonces, el aumento de velocidad en general será:

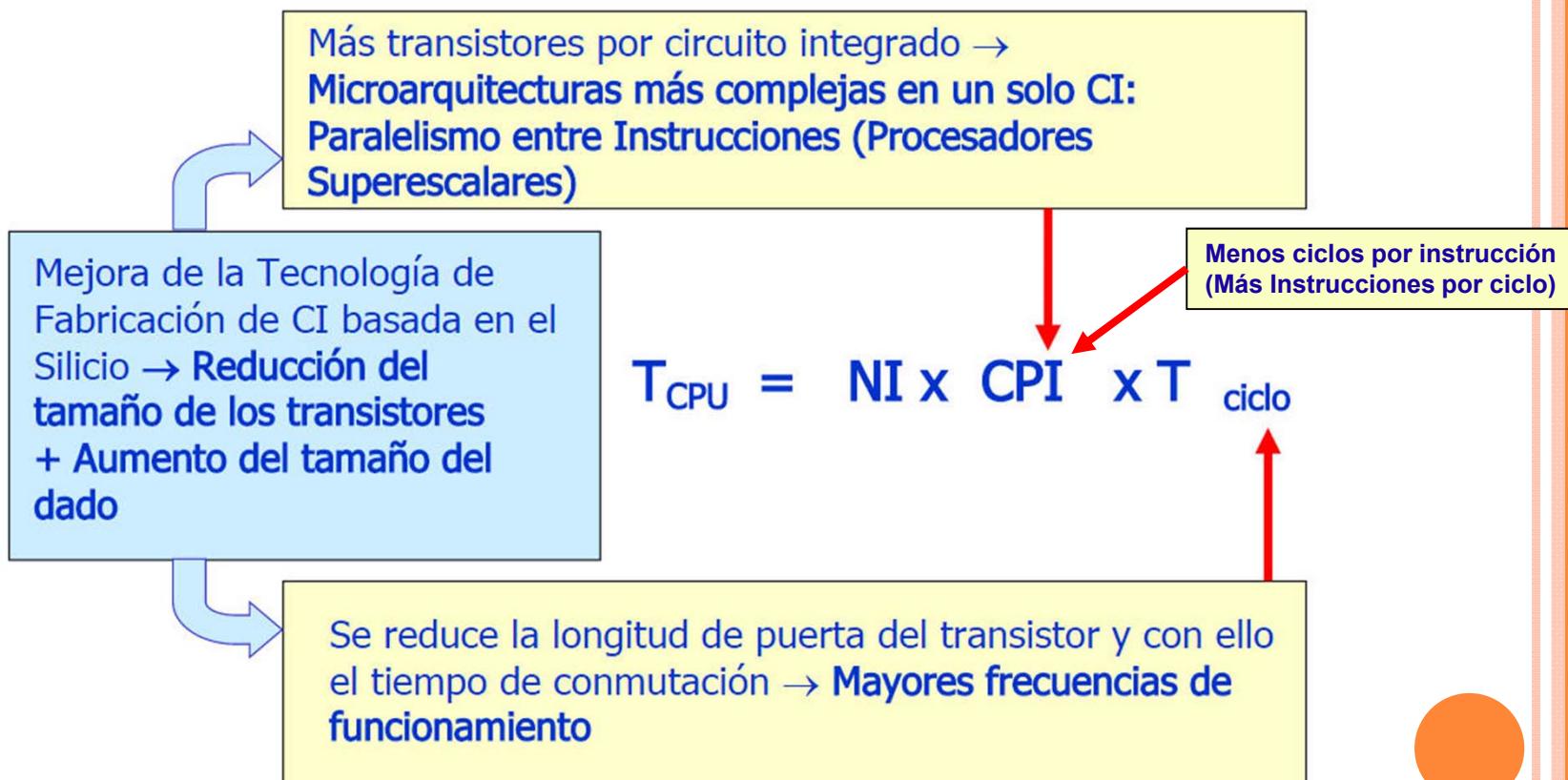
$$\text{Speedup} = \frac{1}{0.6 + \frac{0.4}{K}}$$

Si K = 2 → Speedup = 1,25
Si K = 10 → Speedup = 1,56
Si K = 100 → Speedup = 1,66



MEJORAS DE DESEMPEÑO

Influencia de la Tecnología



PIPELINE

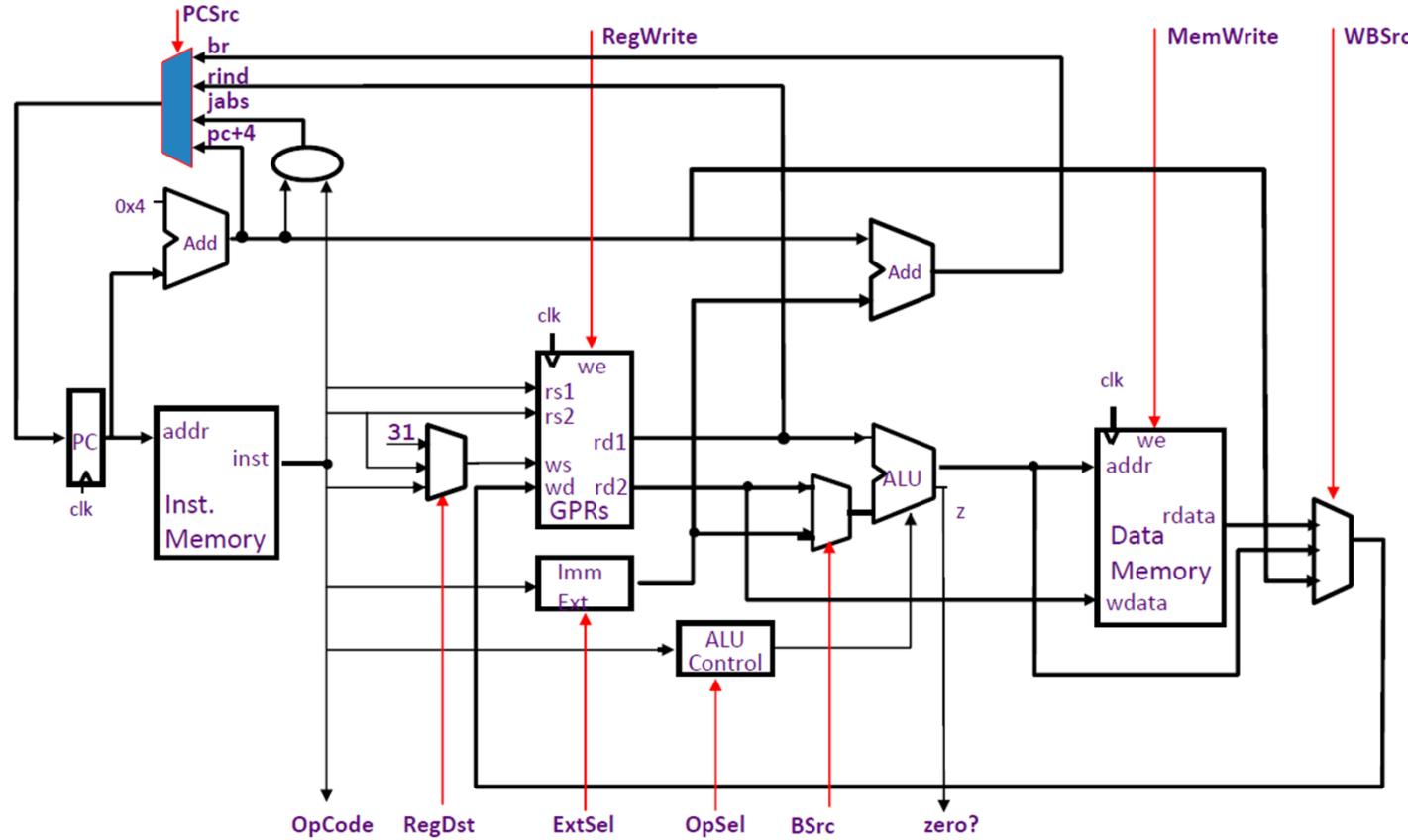


Diagrama en bloques para ejecutar una instrucción típica

PIPELINE

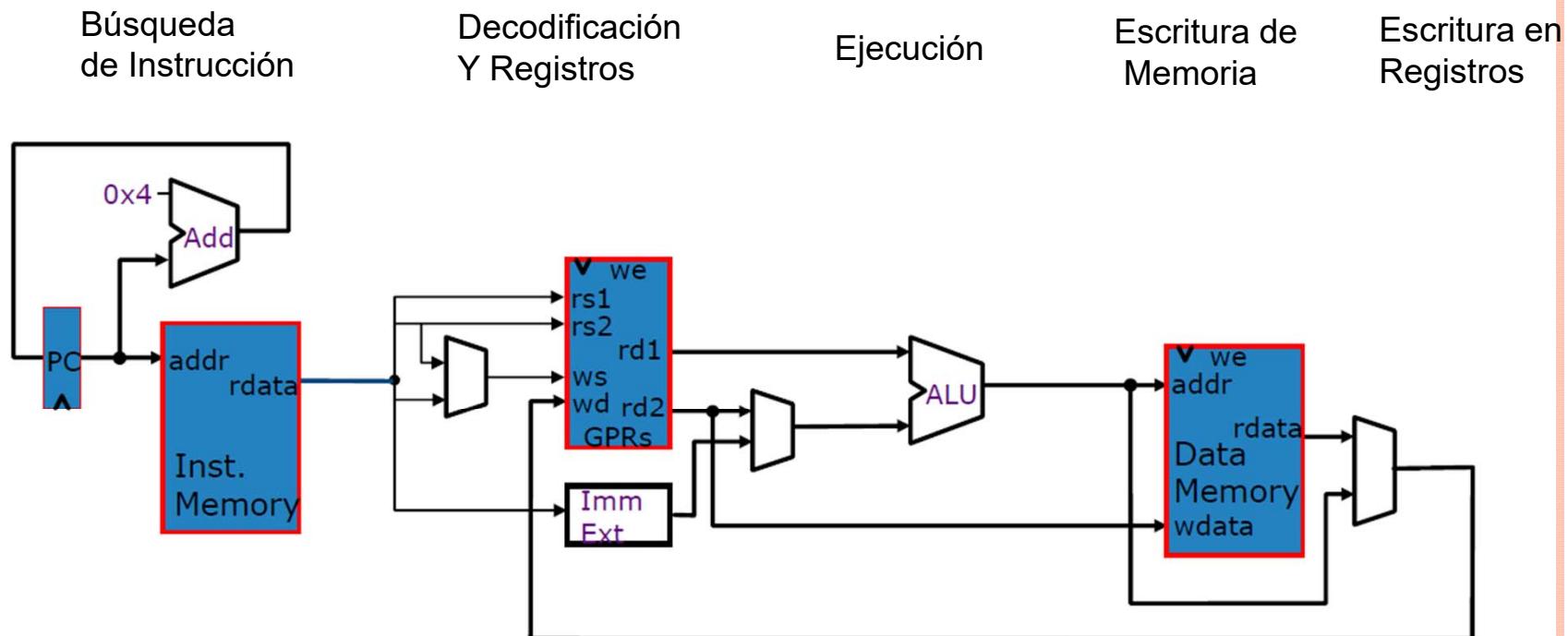
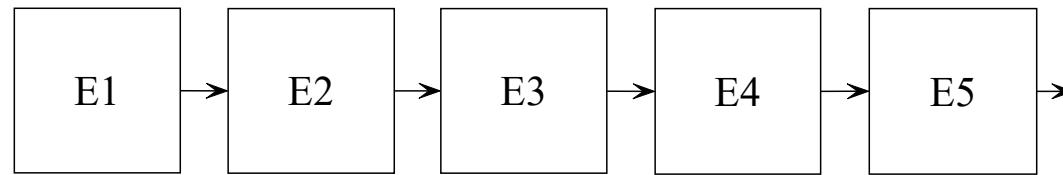
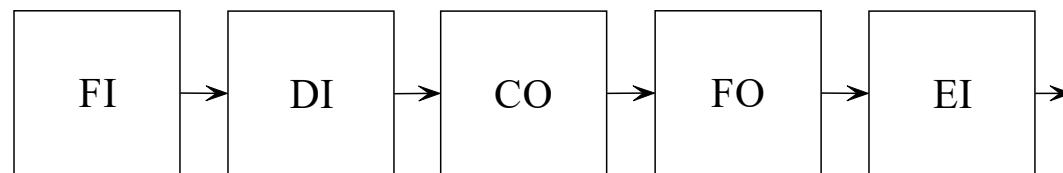


Diagrama en bloques para ejecutar una instrucción típica

PIPELINE



Pasos de una instrucción típica



FI = Fetch Instruction

DI = Decode Instruction

CO = Calculate Operand Address

FO = Fetch Operand

EI = Execute Instruction

FI = Fetch Instruction

DI = Decode Instruction

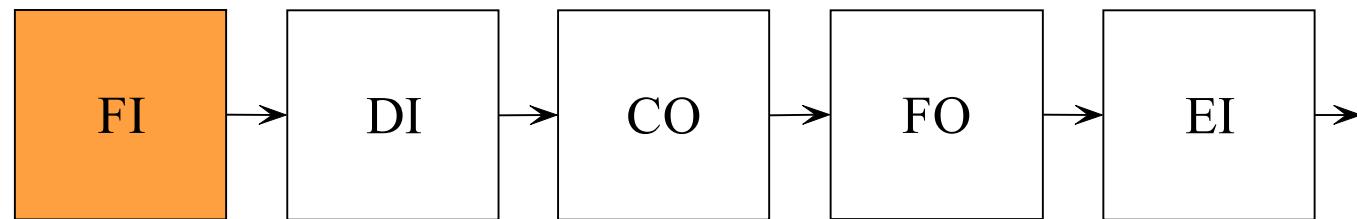
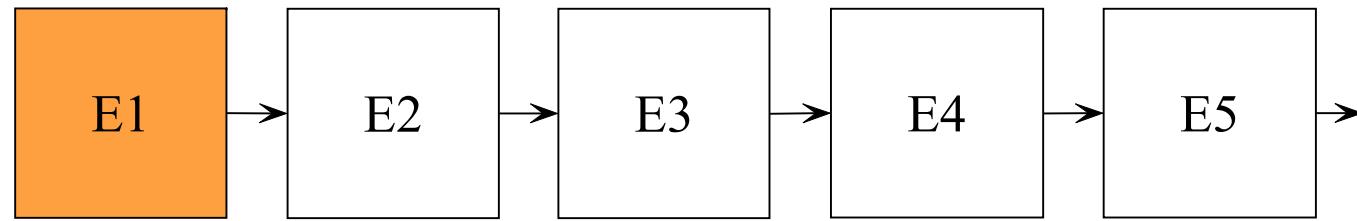
EX = Execute

MEM = Memory

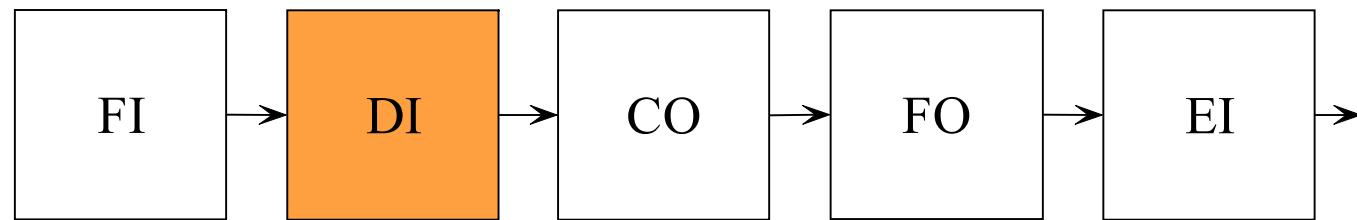
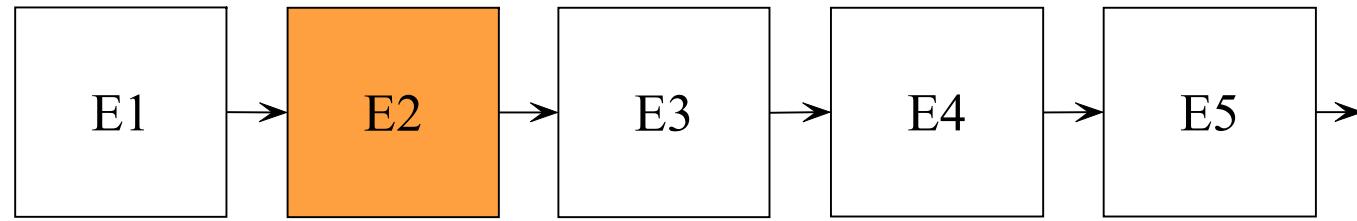
WB = Write Back

ó

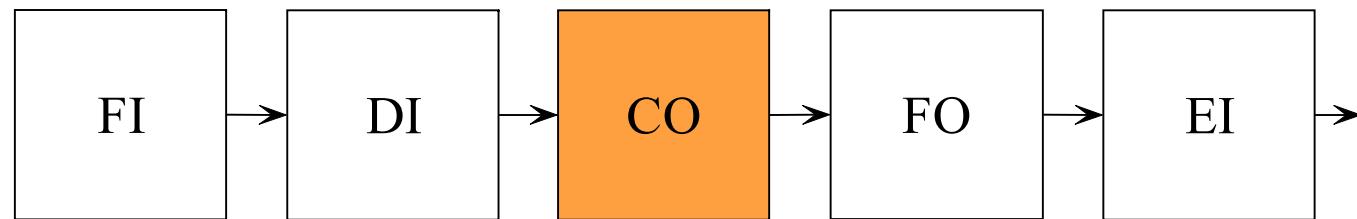
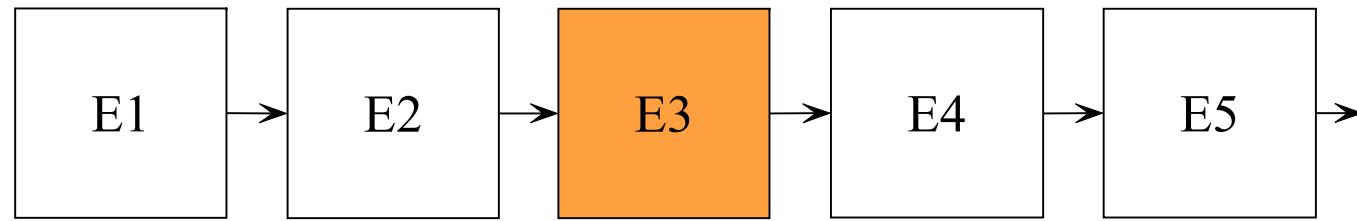
PIPELINE



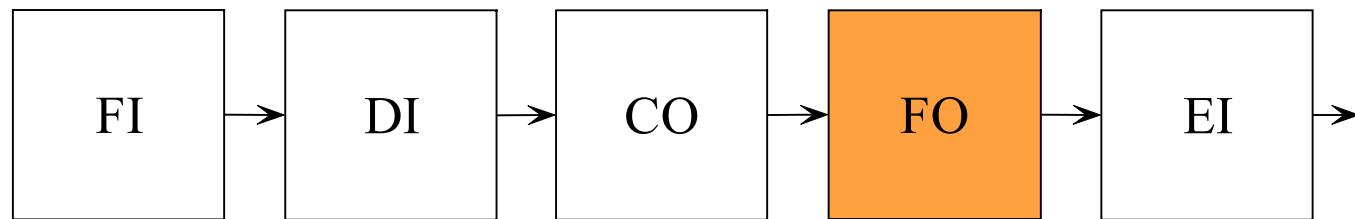
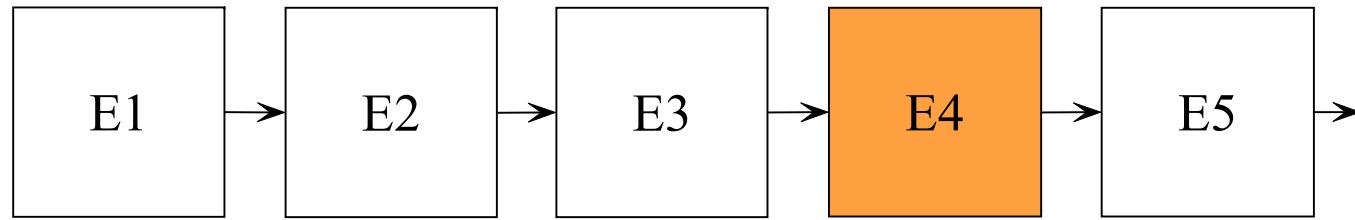
PIPELINE



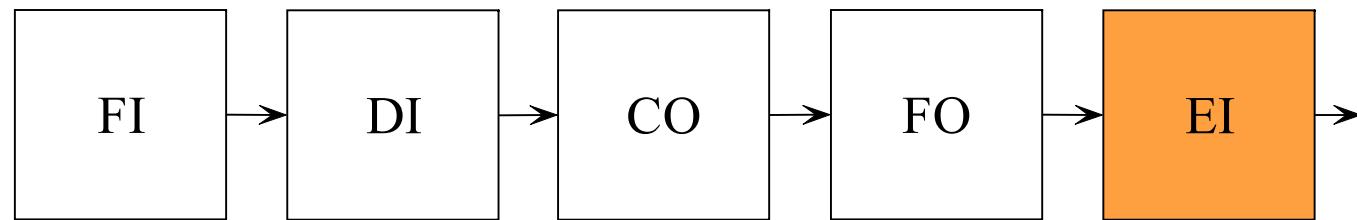
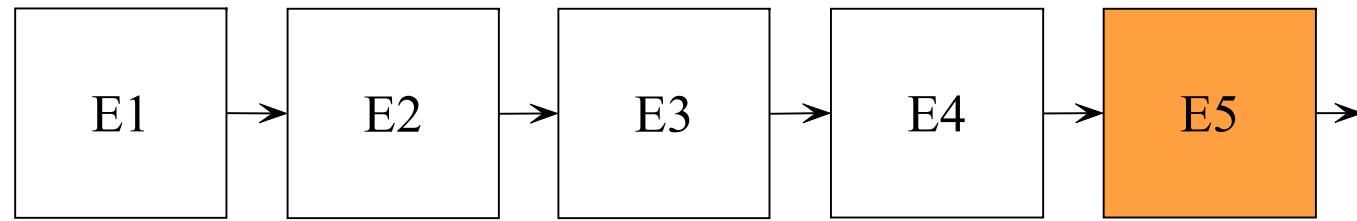
PIPELINE



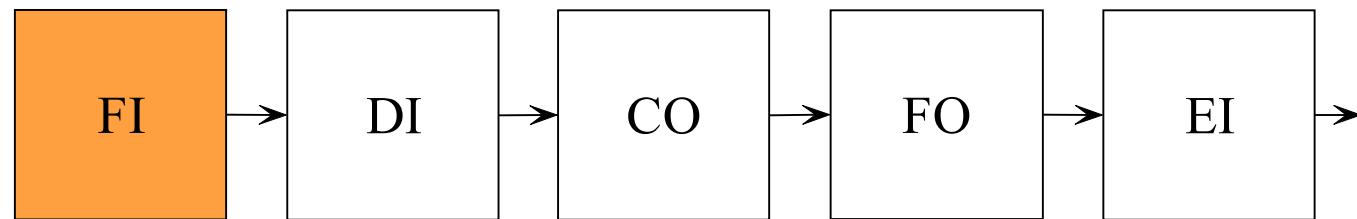
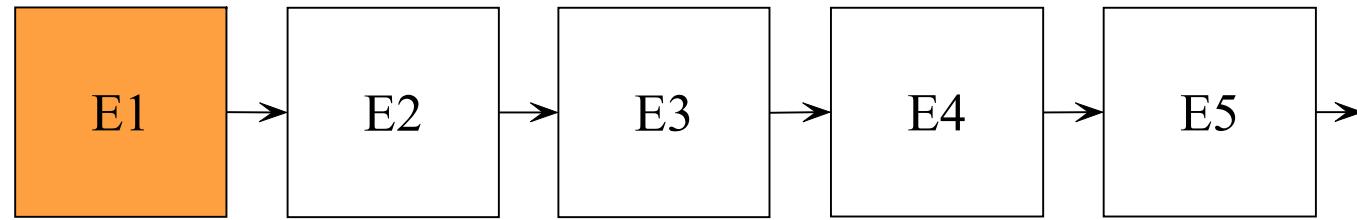
PIPELINE



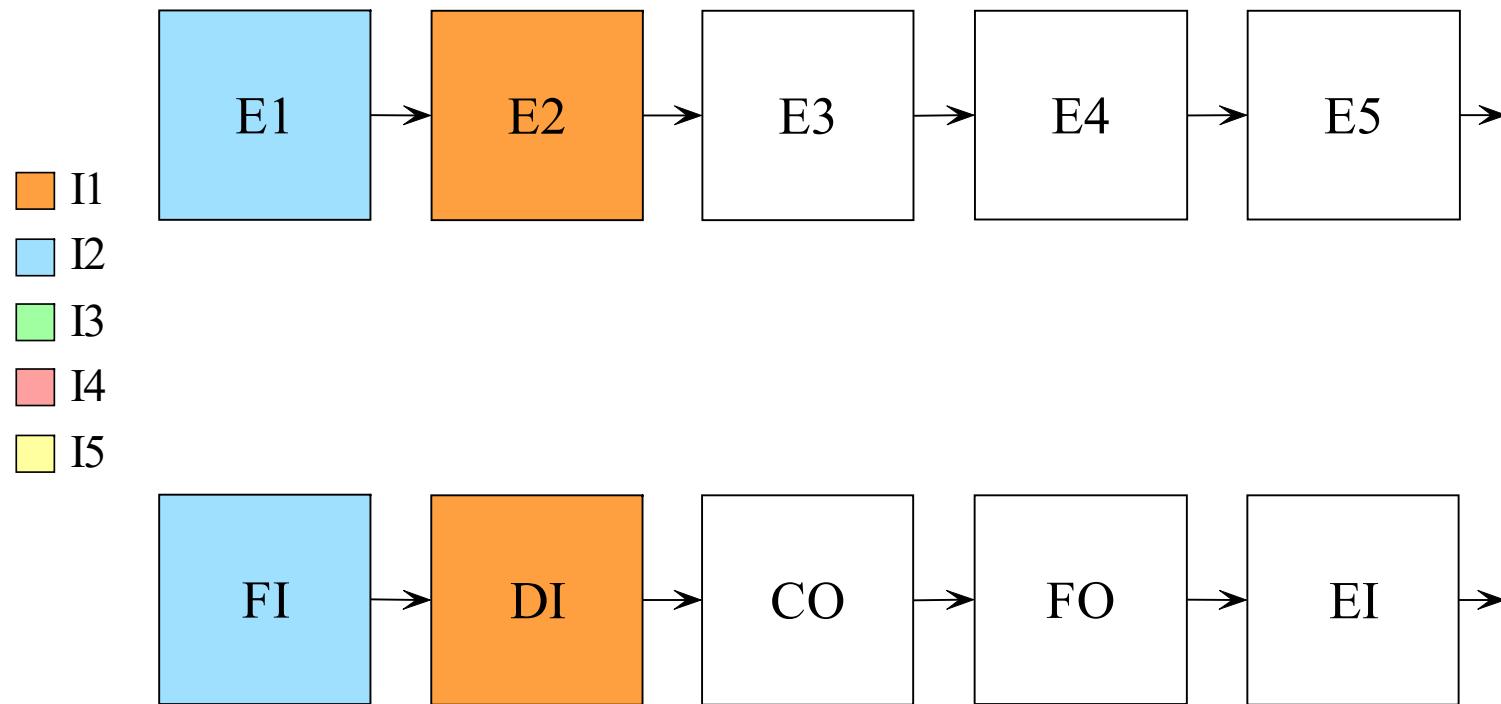
PIPELINE



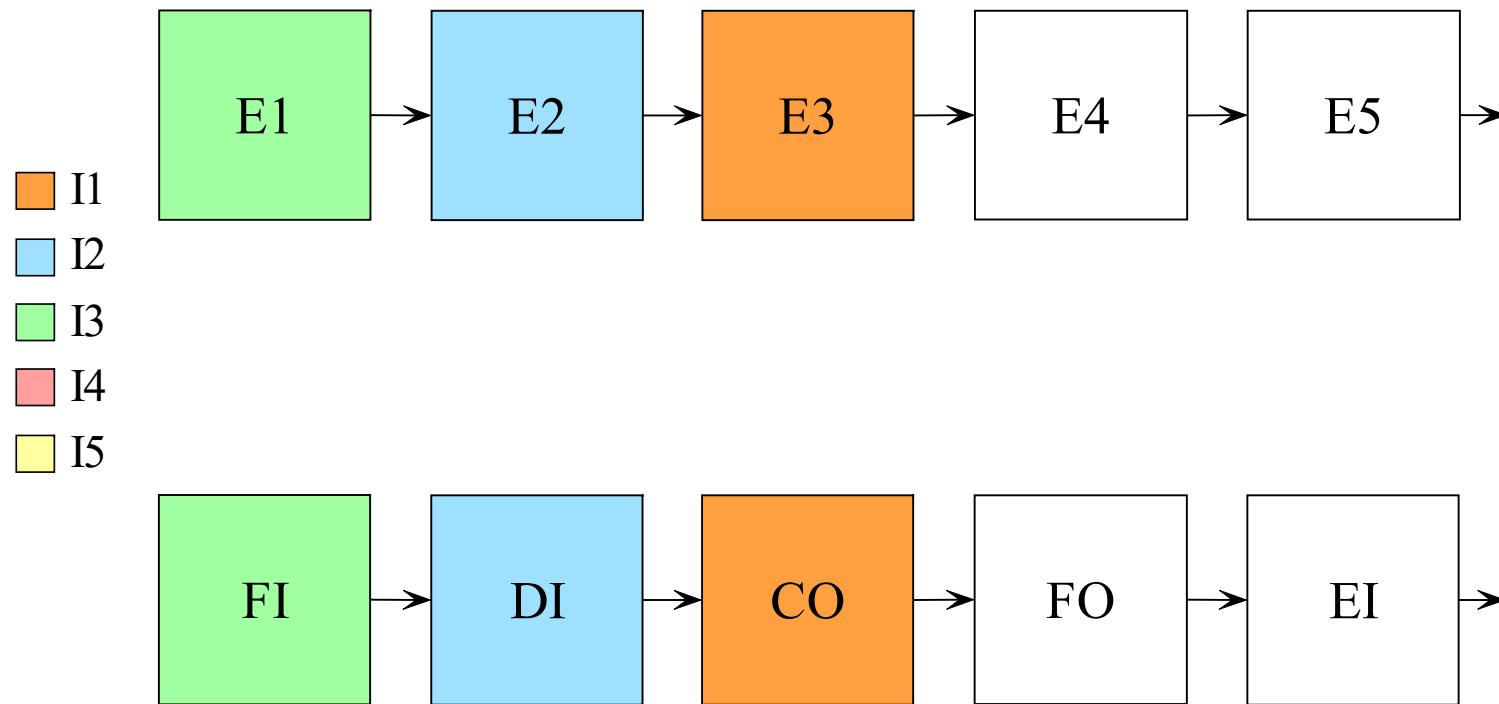
PIPELINE



PIPELINE

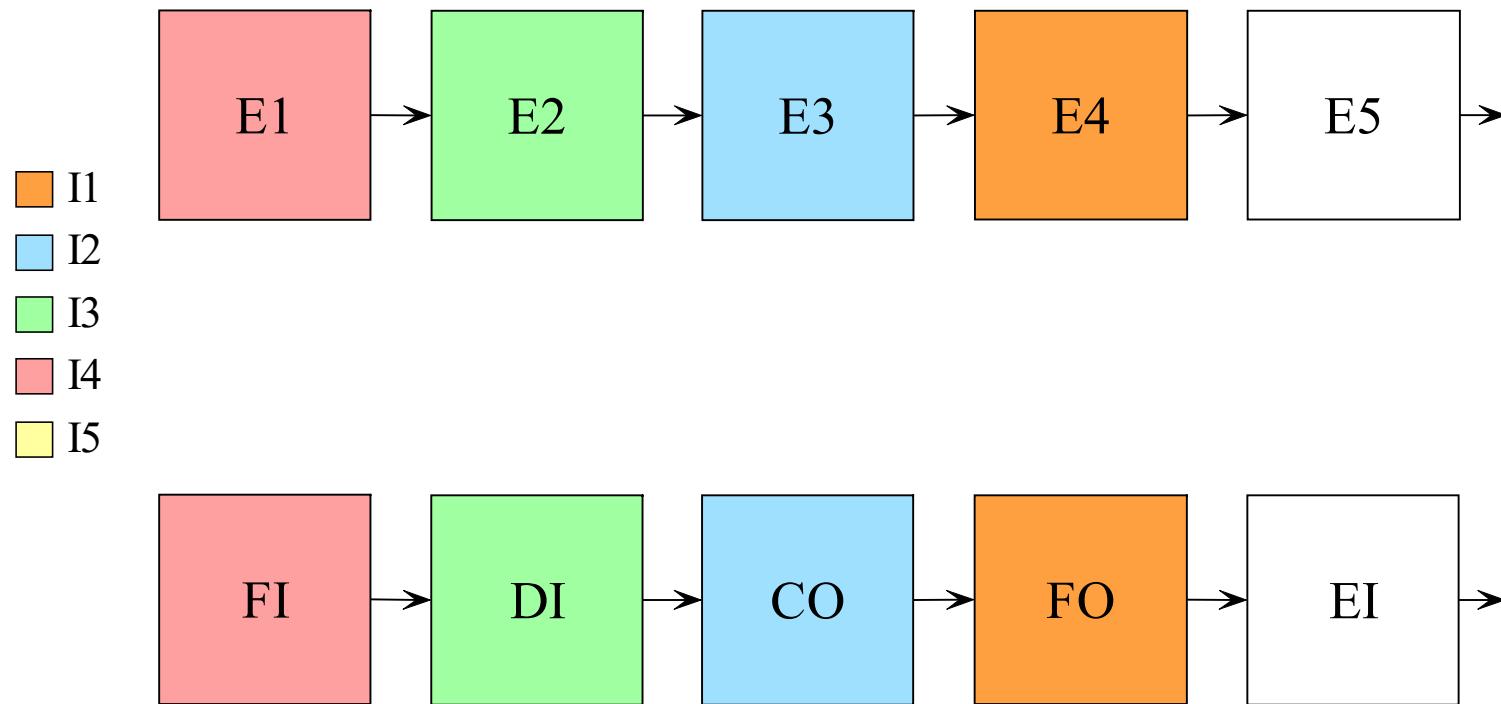


PIPELINE

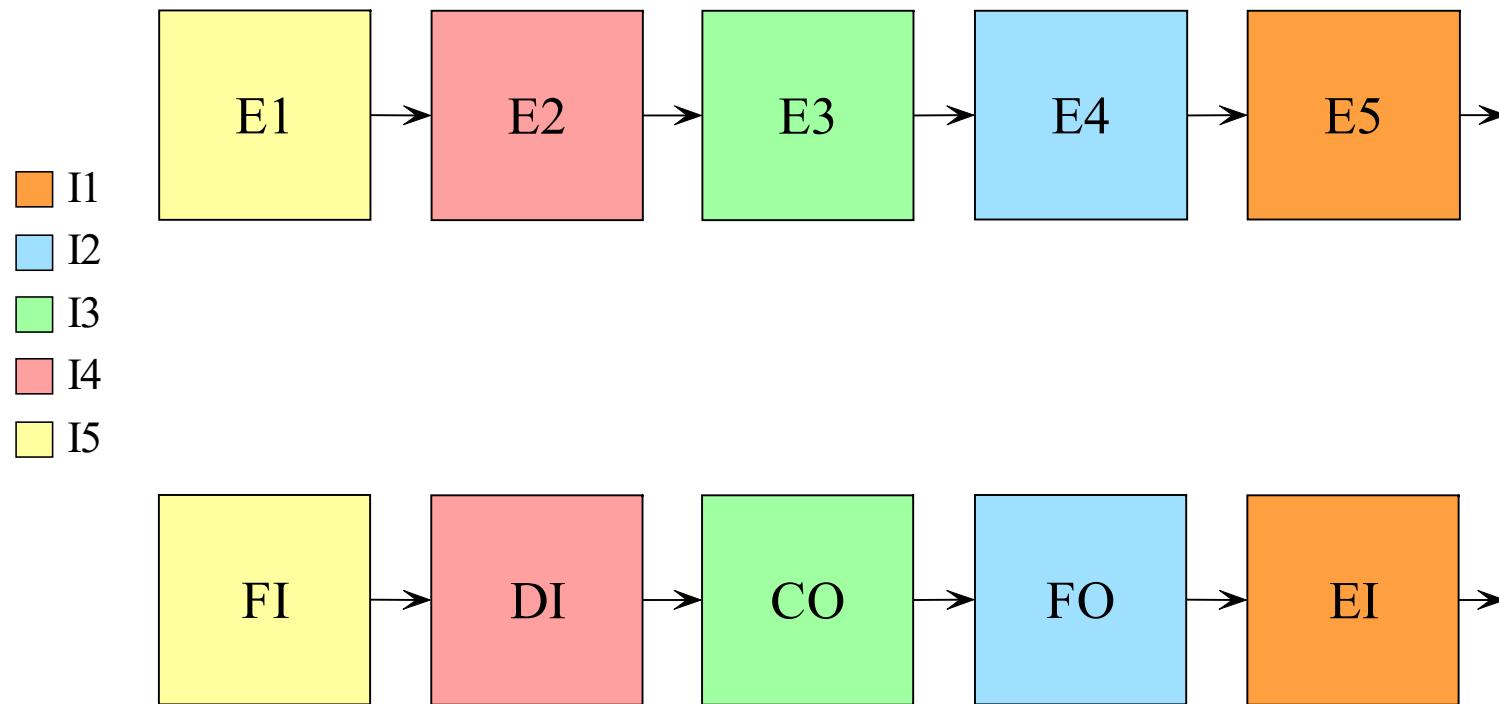


30

PIPELINE



PIPELINE



32

PIPELINE

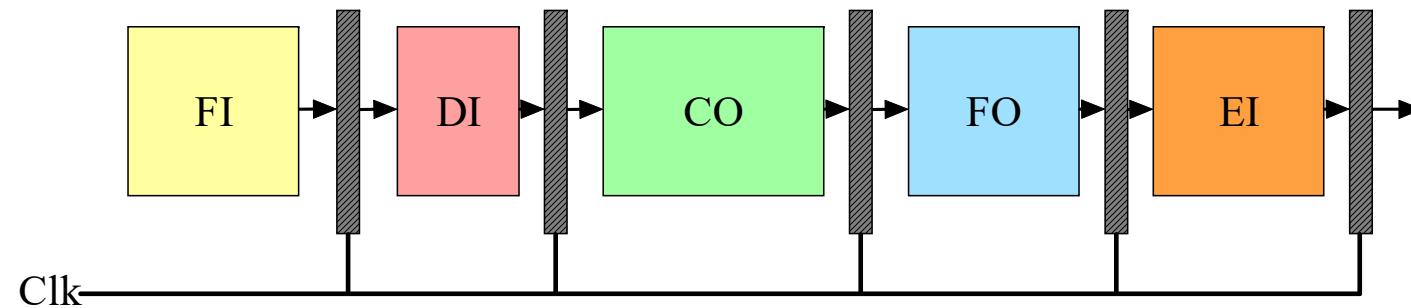
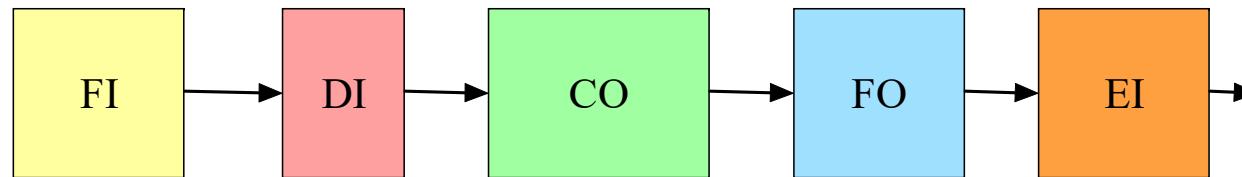
I1

I2

I3

I4

I5



33

PIPELINE

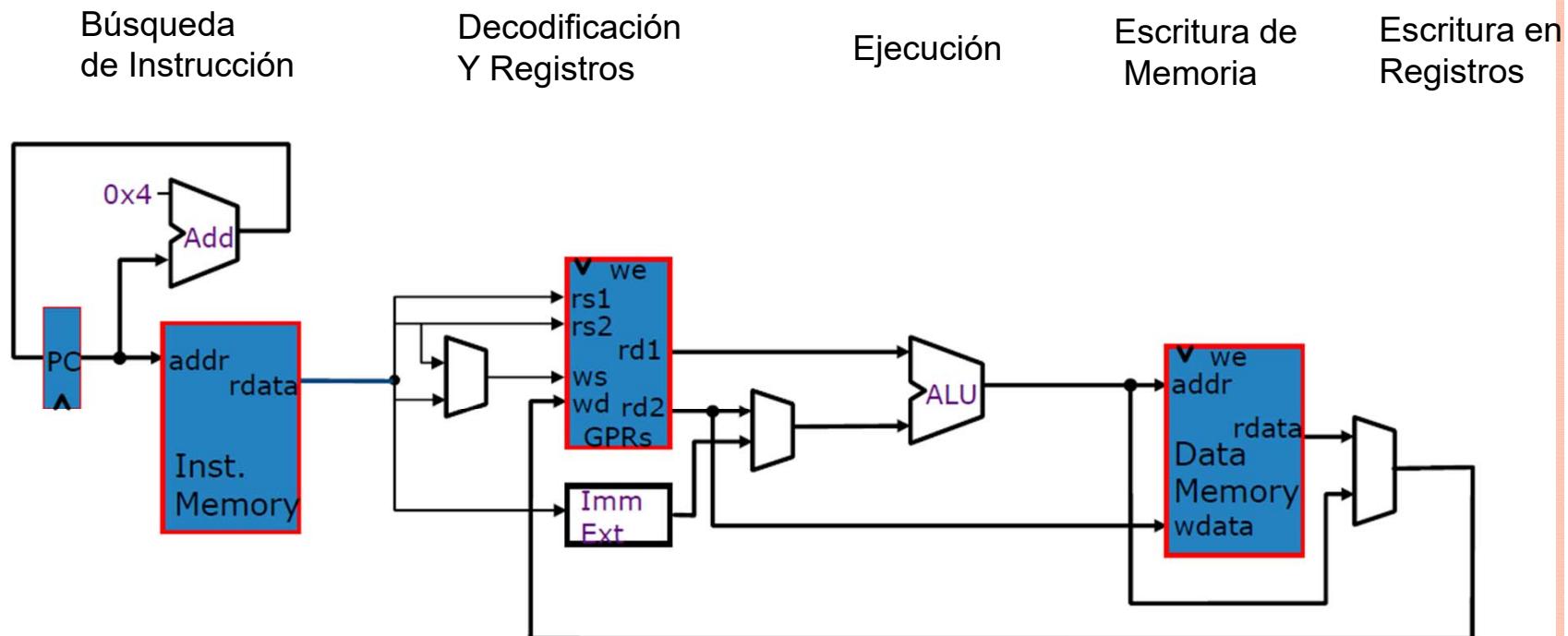


Diagrama en bloques para ejecutar una instrucción típica

PIPELINE

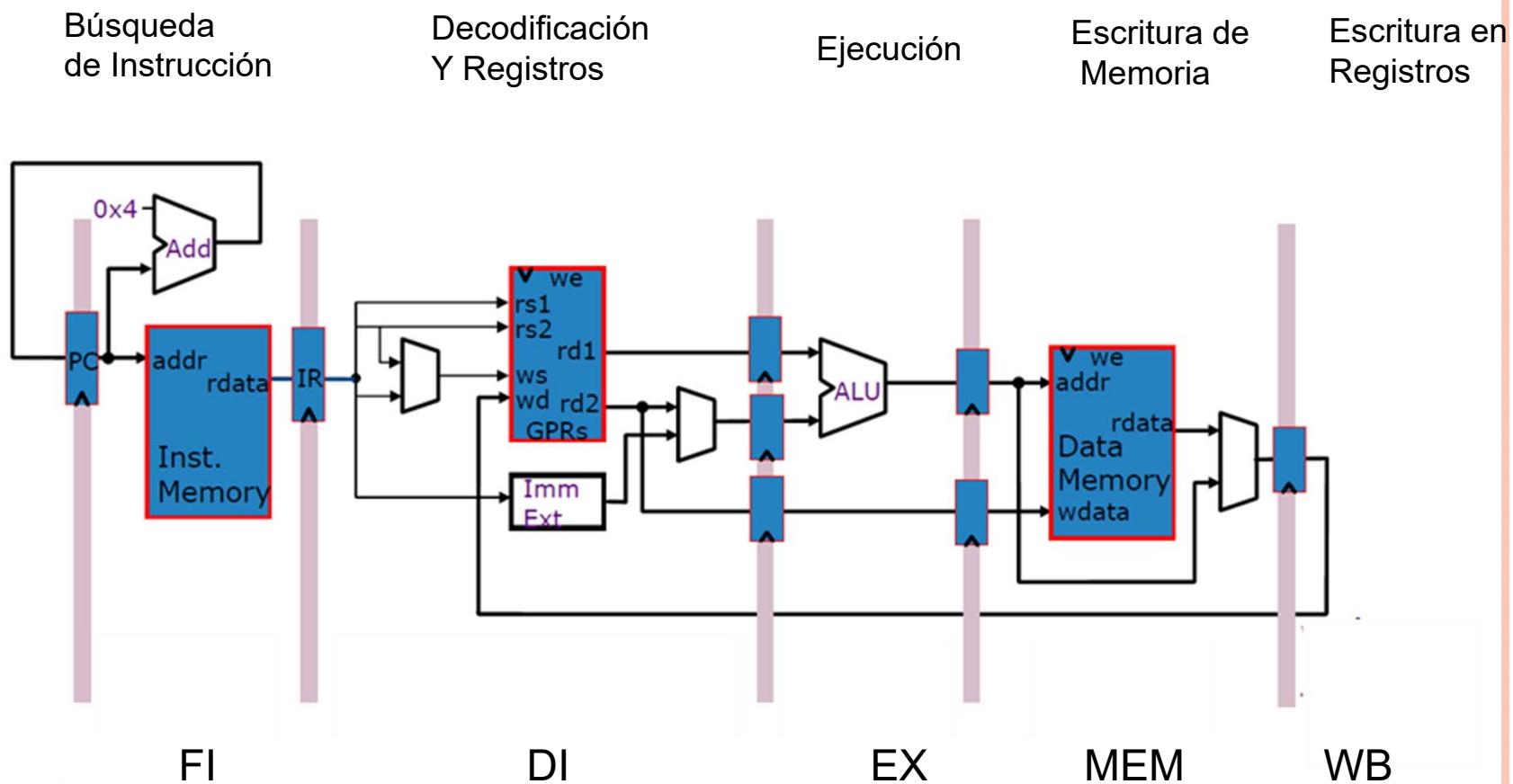


Diagrama en bloques para ejecutar una instrucción típica

PIPELINE

Pipeline AVR328

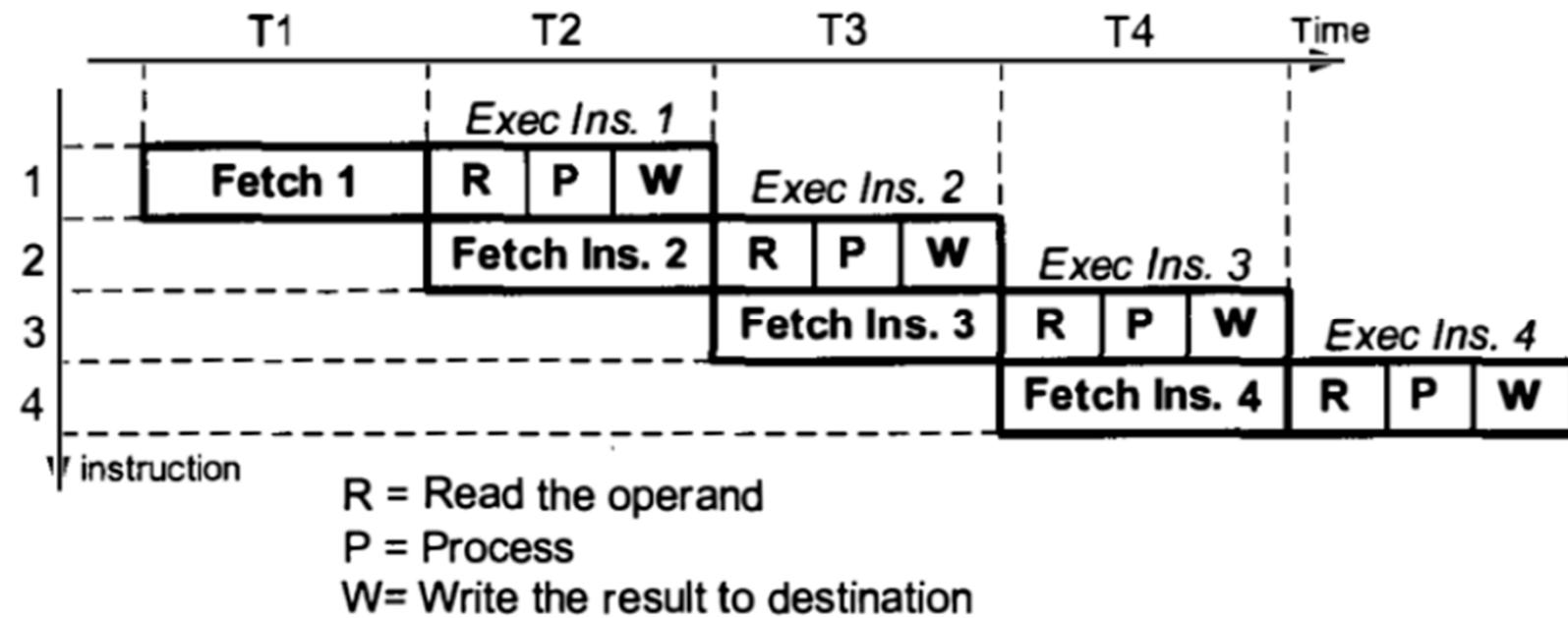
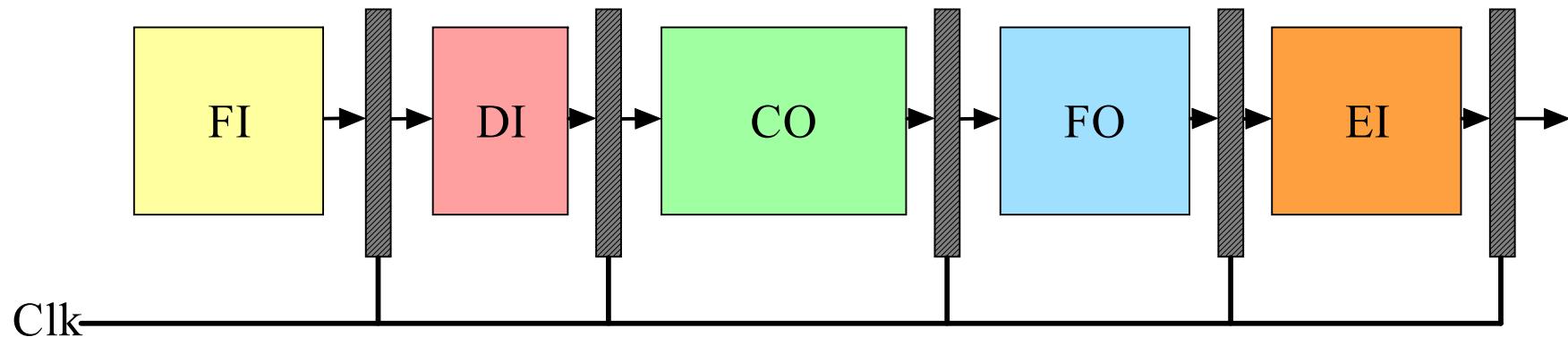


Diagrama en bloques para ejecutar una instrucción típica

PIPELINE



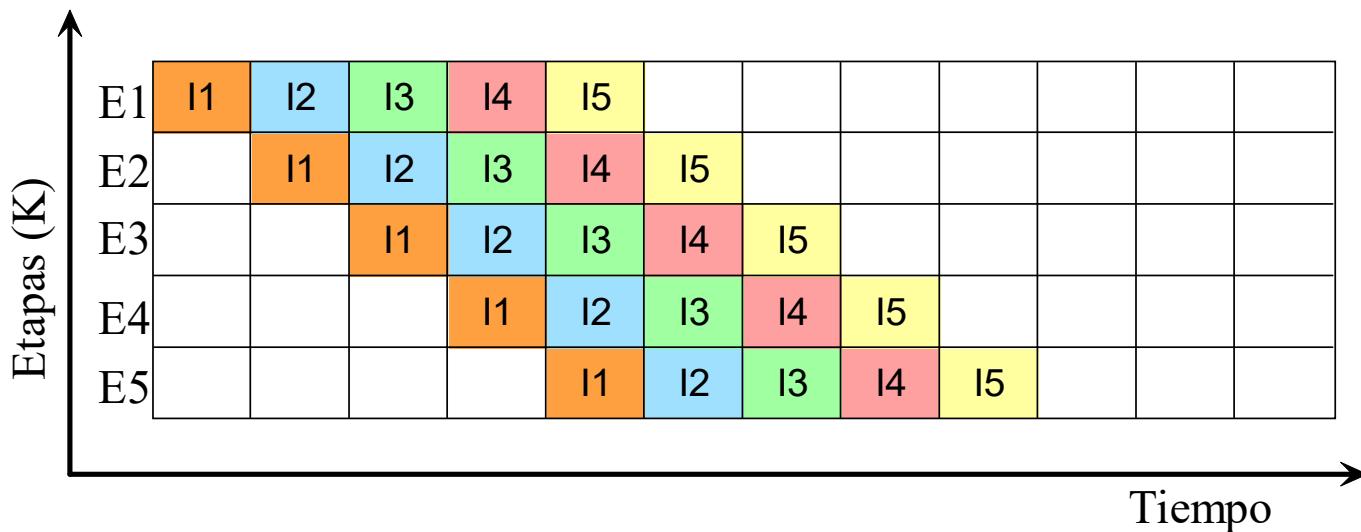
$E_1 = 50\text{ns}$
 $E_2 = 60\text{ns}$
 $E_3 = 80\text{ns}$
 $E_4 = 90\text{ns}$
 $E_5 = 60\text{ns}$
 Latch = 10ns

$$T_{clk\min} = \max T_i^K + T_{latch} \quad f_{\max} = \frac{1}{T_{clk\min}}$$

$$f_{\max} = \frac{1}{(90\text{ns} + 10\text{ns})} = 10\text{MHz}$$

Productividad = f_{\max} (inst/seg)
 Latencia = $\sum(T_i) + \sum(T_{latch})$

PIPELINE



Speedup: K etapas, n instrucciones

$$S = \frac{T_{sp}}{T_p} = \frac{nKt}{(K+n-1)t} = \frac{nK}{K+n-1} \quad S = K \quad n \rightarrow \infty$$

PIPELINE

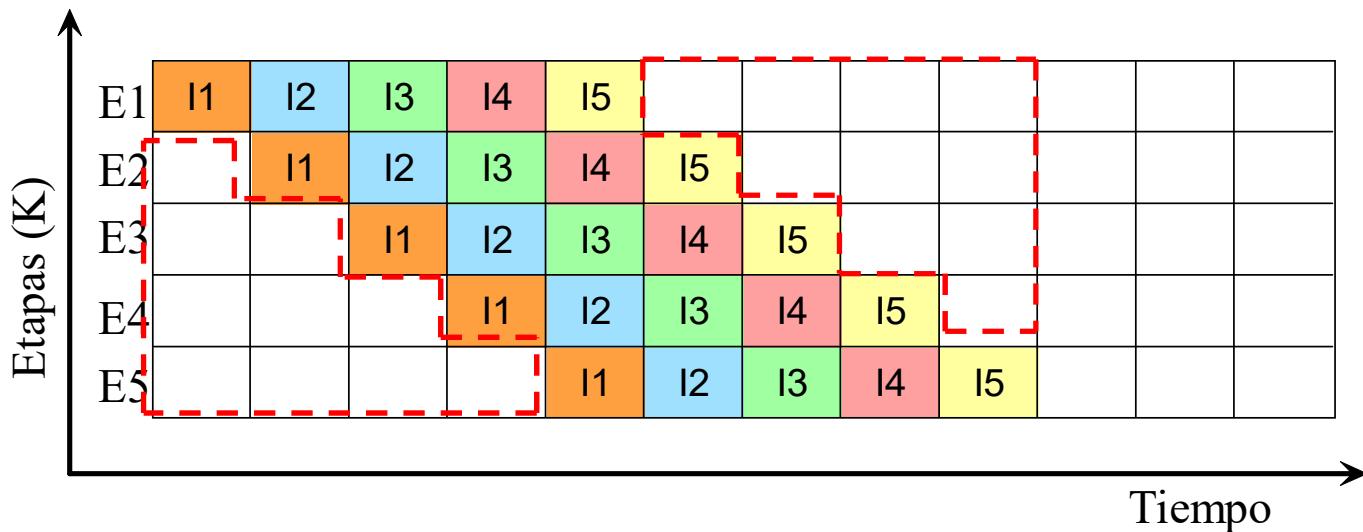
$$S = \frac{nK}{K+n-1}$$

$$n = 10 \rightarrow S = \frac{10 \times 5}{5 + 10 - 1} = 3,6$$

$$n = 50 \rightarrow S = \frac{50 \times 5}{5 + 50 - 1} = 4,6$$

$$n = 100 \rightarrow S = \frac{100 \times 5}{5 + 100 - 1} = 4,8$$

PIPELINE



Rendimiento:

$$\eta = \frac{S}{K} = \frac{nK}{K(K + n - 1)} = \frac{n}{K + n - 1}$$

$$\eta = 1$$

$$n \rightarrow \infty$$

PIPELINE

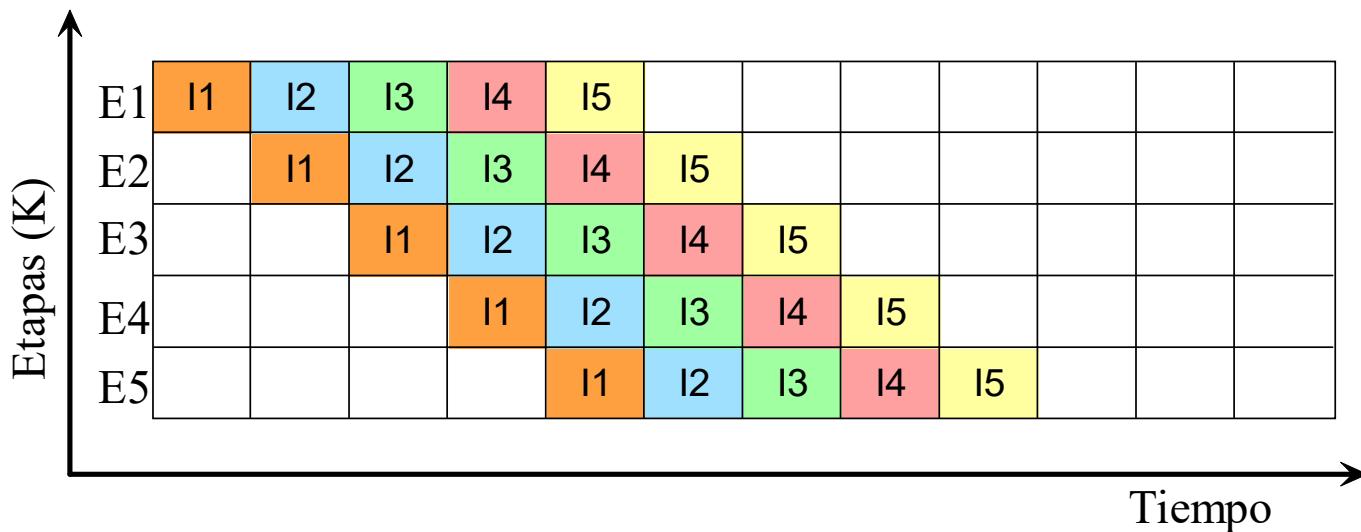
$$\eta = \frac{n}{K+n-1}$$

$$n = 10 \rightarrow \eta = \frac{10}{5+10-1} = 0,71$$

$$n = 50 \rightarrow S = \frac{50}{5+50-1} = 0,92$$

$$n = 100 \rightarrow S = \frac{100}{5+100-1} = 0,96$$

PIPELINE



Ancho de Banda:

$$F_H = \frac{n}{(K+n-1)T} = \frac{nf}{K+n-1}$$

$$\lim_{n \rightarrow \infty} F_H = f$$

PIPELINE

$$F_H = \frac{nf}{K+n-1}$$

$$n = 10 \rightarrow F_H = \frac{10 \times 10 \text{MHz}}{(5+10-1)} = 7,14 \text{MHz}$$

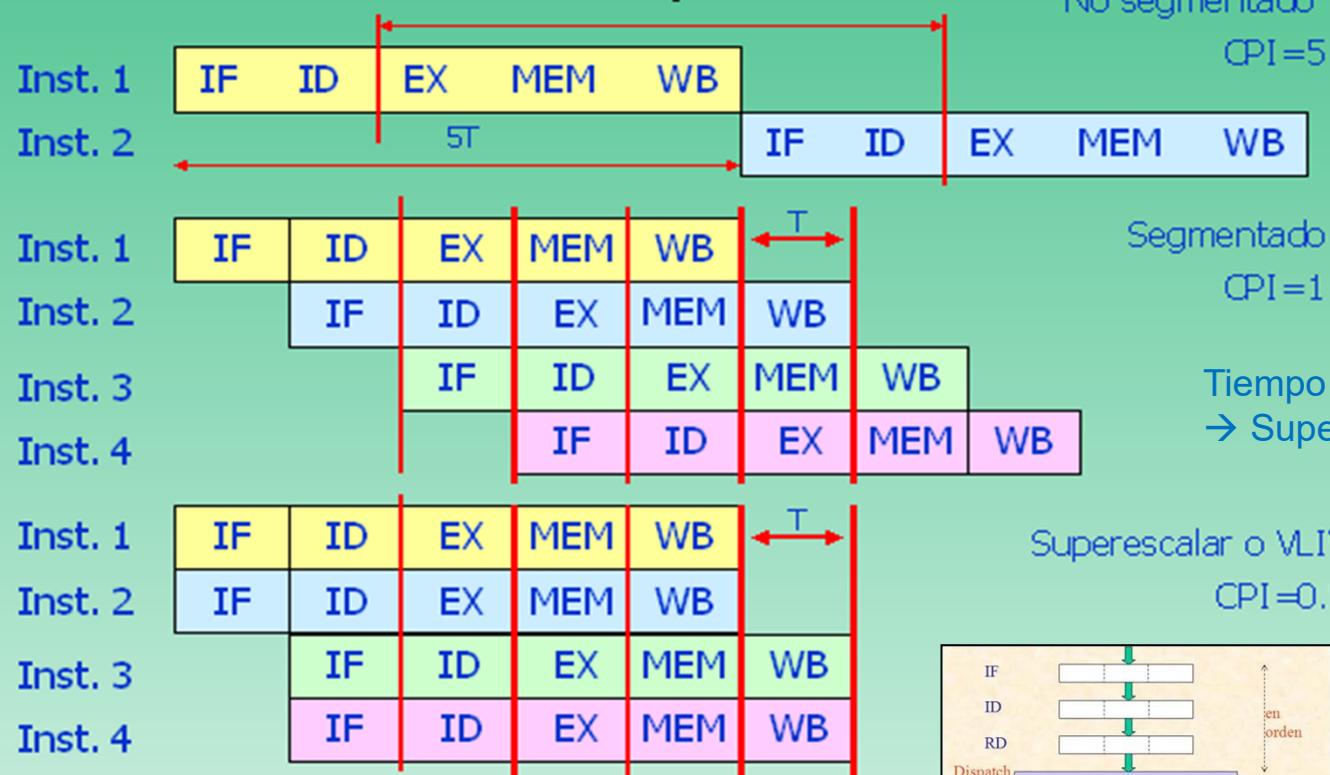
$$n = 50 \rightarrow F_H = \frac{50 \times 10 \text{MHz}}{5 + 50 - 1} = 9,25 \text{MHz}$$

$$n = 100 \rightarrow F_H = \frac{100 \times 10 \text{MHz}}{5 + 100 - 1} = 9,62 \text{MHz}$$

ESCALARES - SUPERESCALARES

$$t_{ejecución} = N_{inst} \times CPI \times T$$

Reducción de ciclos por instrucción

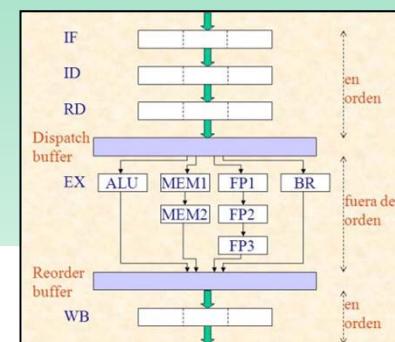


Ejecución Secuencial

Escalar,
1Inst/Ciclo

Tiempo de etapa < 1C
→ Supersegmentación

Superescalar o VLIW
CPI=0.5



MEJORA EN EL DESEMPEÑO DE MICROPROCESADORES

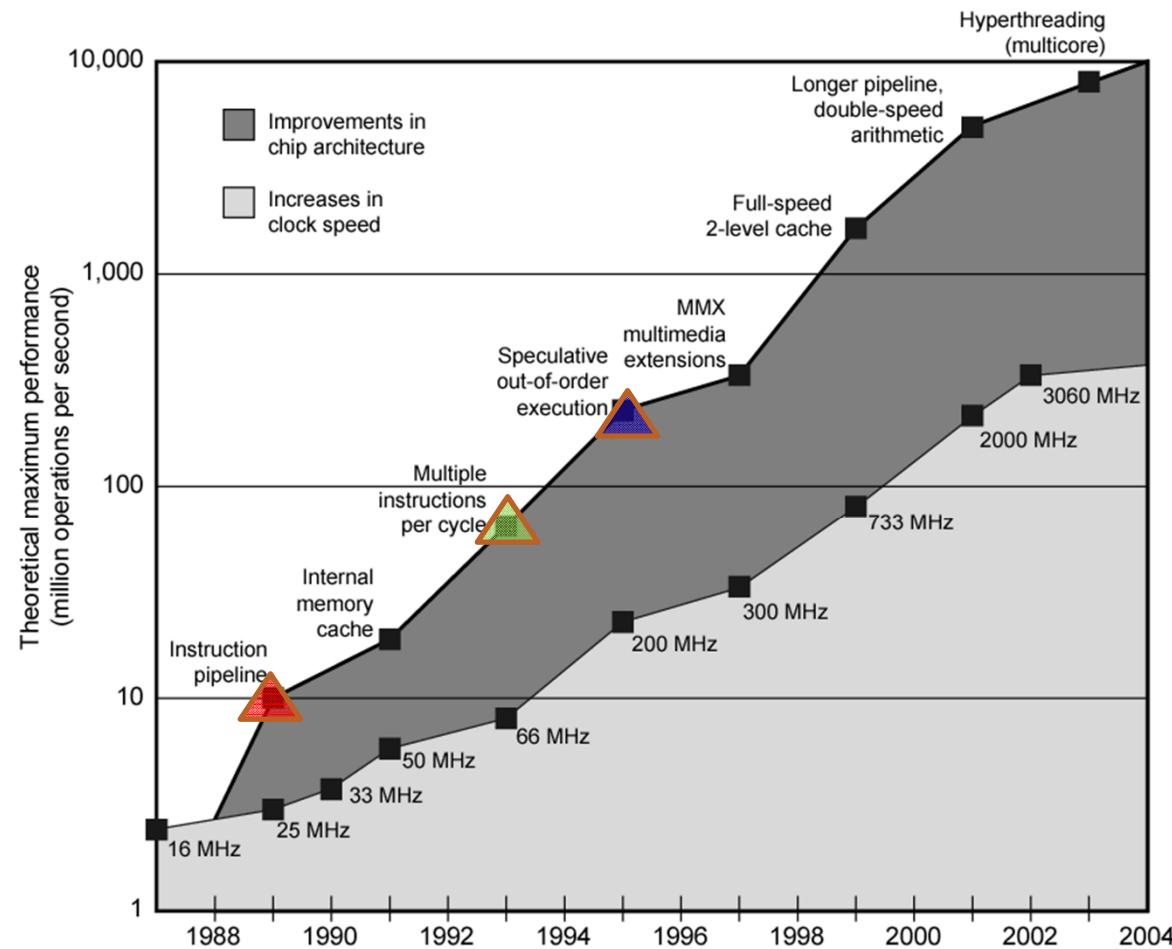
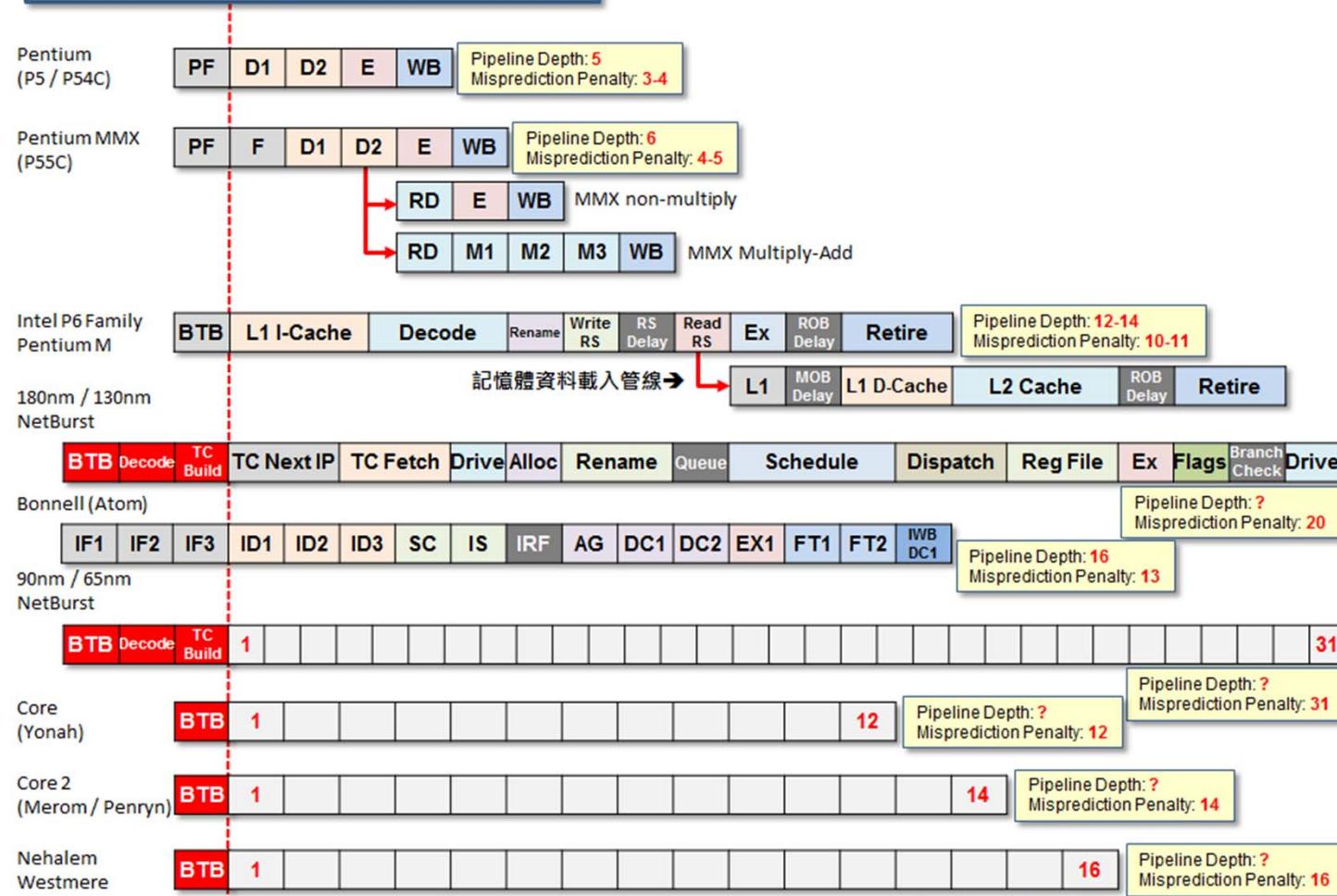


Figure 2.12 Intel Microprocessor Performance [GIBB04]

PIPELINE

Intel x86 Pipeline History

<http://molesterwaterball.blogspot.com>
痴漢水球的部落格



PIPELINES: RIESGOS

RIESGOS ESTRUCTURALES

(conflicto de recursos)

RIESGOS POR DEPENDENCIA DE DATOS

(RAW-WAR-WAW)

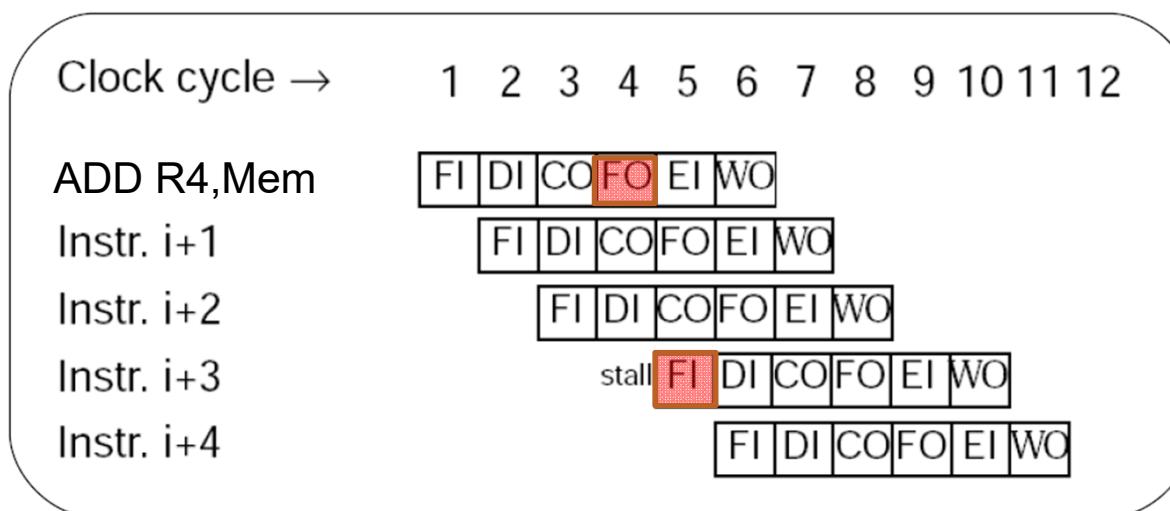
RIESGOS DE CONTROL

(saltos - interrupciones)

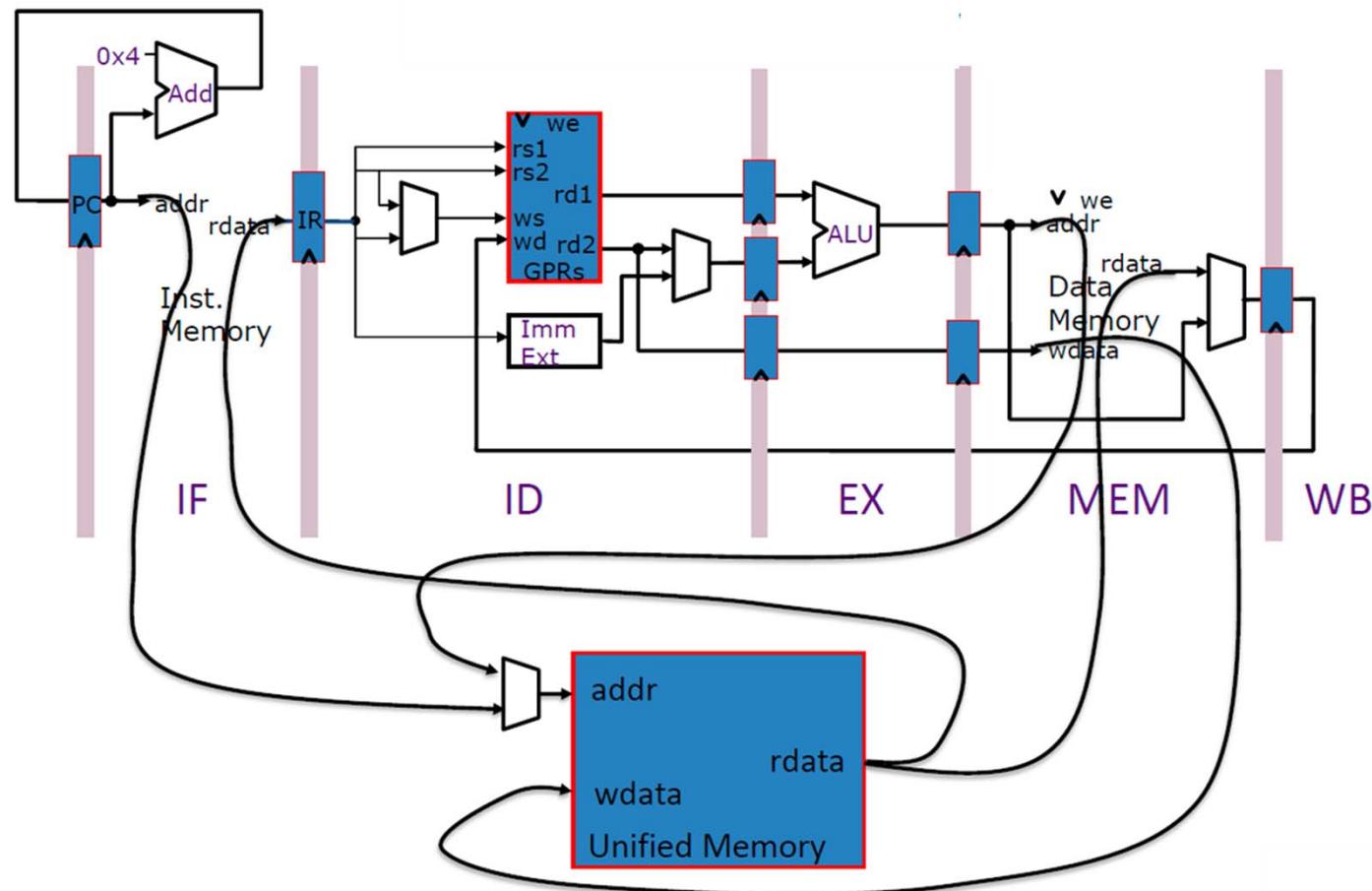
RIESGOS ESTRUCTURALES

Sucede cuando un recurso (memoria, alu) debe ser utilizado por varias instrucciones simultáneamente.

Ejemplo: una instrucción de carga desde memoria impide el acceso durante un ciclo. El fetch de la instrucción $i+3$ debe ser suspendido.



RIESGOS ESTRUCTURALES



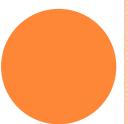
REDUCCIÓN DE LOS EFECTOS

- Duplicación de recursos, por ejemplo en el caso de la ALU
- Cache de datos e instrucciones independientes, para evitar los conflictos de memoria.
- Un solo acceso a memoria de datos por instrucción (RISC).
- Lectura y escritura simultánea del banco de registros (RISC).
- Las unidades funcionales de PF pueden ser a la vez segmentadas para soportar varias instrucciones simultáneas



TIPOS DE RIESGOS DE DATOS

- Para dos instrucciones consecutivas los riesgos pueden clasificarse en tres categorías, siendo la primera la más usual.
- - **RAW (read after write)**: la segunda instrucción lee un dato antes que la primera lo genere. Lee el dato antiguo.
- - **WAR (write after read)**: la segunda escribe un destino antes que sea leído por la primera. La primera toma el valor incorrecto (nuevo).
- - **WAW (write after write)**: la segunda escribe un operando antes de que sea escrito por la primera. Escrituras en orden incorrecto. Queda lo escrito por la primera.

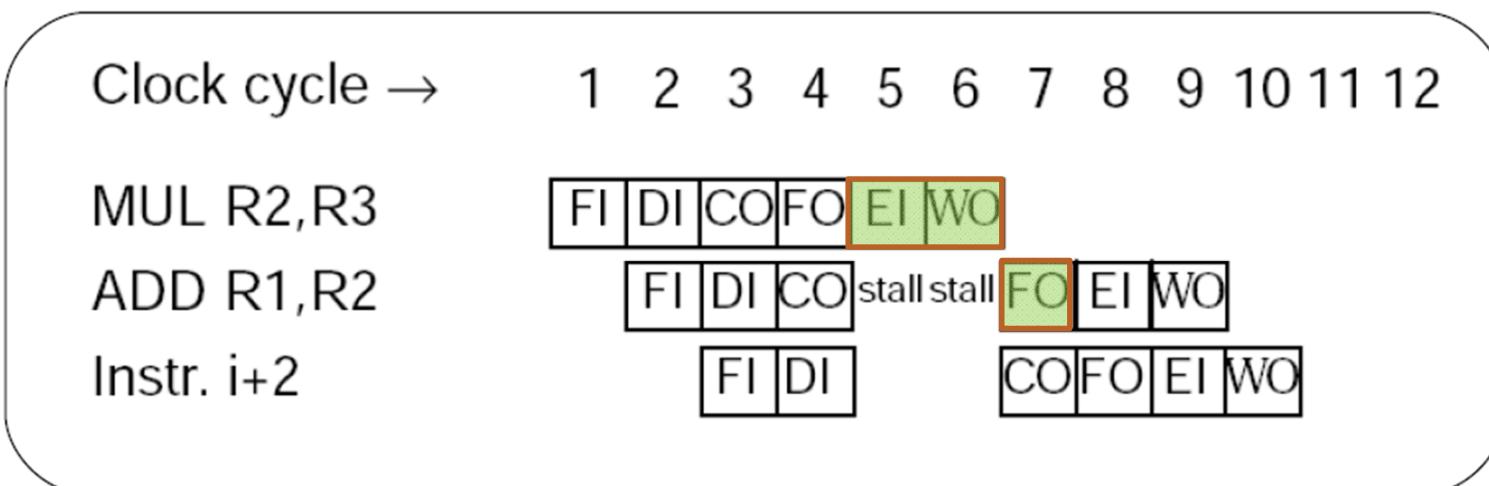


RIESGOS POR DEP. DE DATOS

Sucede principalmente cuando una instrucción requiere un dato generado por la ejecución de una instrucción anterior que aún no ha finalizado.

Ejemplo:

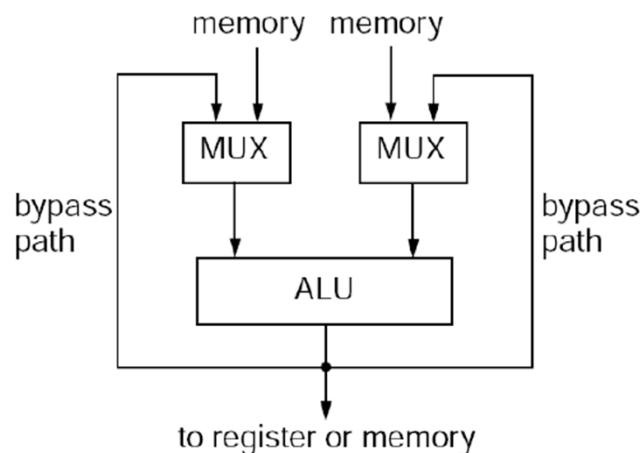
I1:	MUL	R2,R3	R2 \leftarrow R2 * R3
I2:	ADD	R1,R2	R1 \leftarrow R1 + R2



Penalidad: 2 ciclos



REDUCCIÓN DE LOS EFECTOS



Adelantamiento (Forwarding)

Camino adicional de hardware:

El resultado de la ALU es realimentado a su entrada, evitando el ciclo de escritura WO.
Búsqueda del operando (FO) y Escritura del operando por la instrucción anterior se hacen en el mismo ciclo.

4 5 6 7 8 9 10 11 12

MUL R2,R3
ADD R1,R2

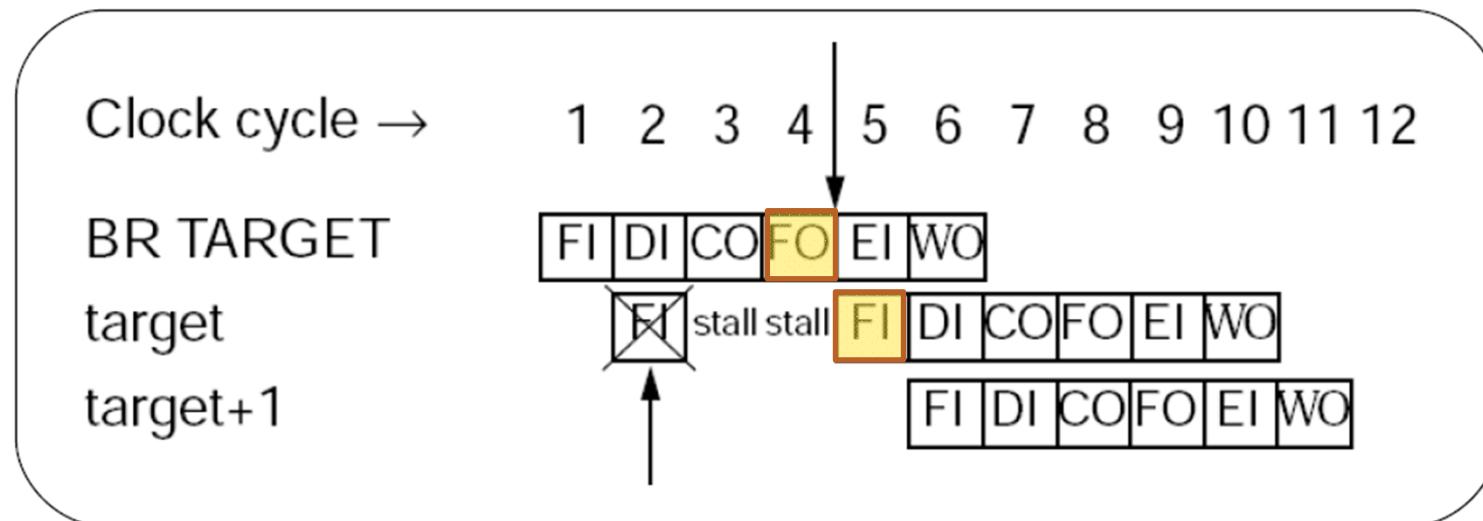


Penalidad: 1 ciclos

RIESGOS DE CONTROL

Producidos por las instrucciones de salto. Interrupciones.

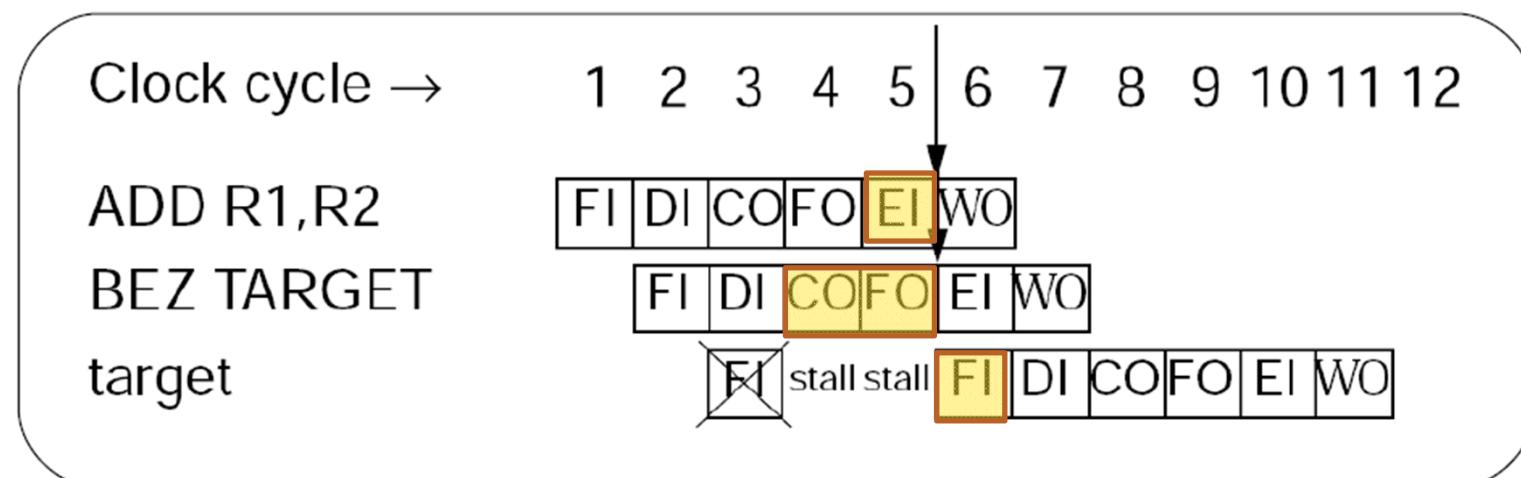
Ejemplo salto incondicional: no se conoce la dirección de la próxima instrucción hasta después del FO. Se realiza el fetch de la instrucción siguiente y luego se descarta.



Penalidad: 3 ciclos

RIESGOS DE CONTROL

Salto condicional que SALTA: no se conoce la dirección de la próxima instrucción hasta después del EI.

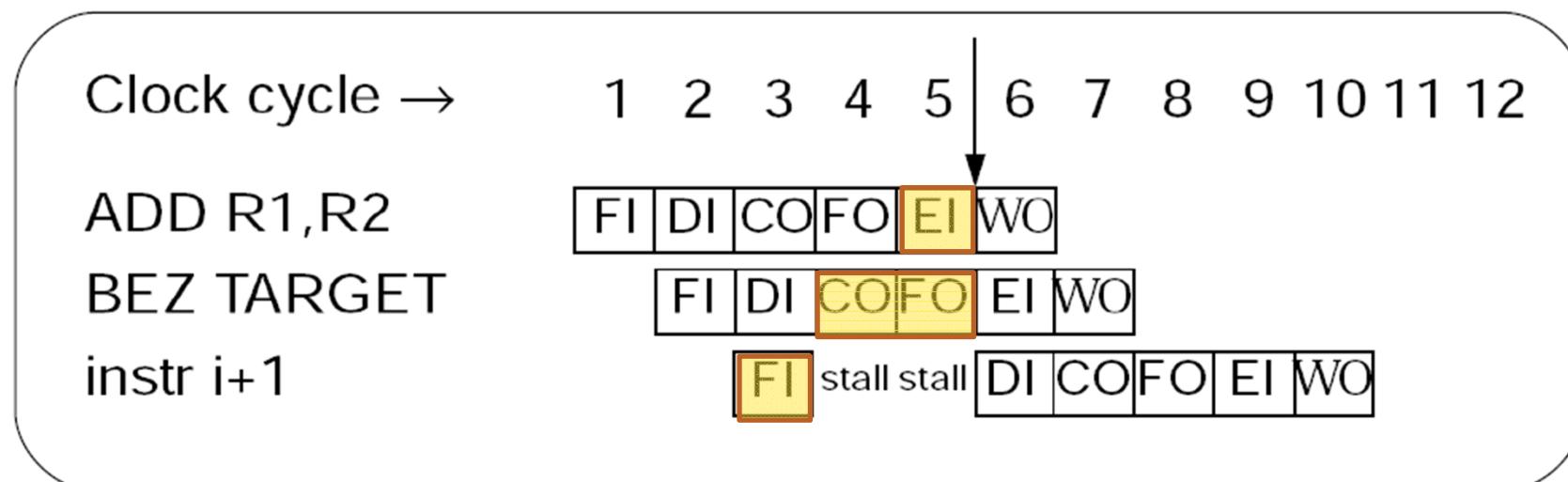


Penalidad: 3 ciclos



RIESGOS DE CONTROL

Ejemplo salto condicional que NO SALTA: no se conoce la condición hasta después del EI, cuando la siguiente instrucción puede continuar.



RIESGOS DE CONTROL

Las operaciones de control (condicionales o incondicionales) son muy frecuentes en los programas reales, por lo que pueden reducir drásticamente la performance del pipeline.

Estadísticas:

- 20-35% de las operaciones son saltos.
- ~65% de dichas operaciones toman el salto.
- Hay casi el doble de saltos condicionales que de saltos incondicionales.



REDUCCIÓN DE LOS EFECTOS

Instruction fetch units y colas de instrucciones

(hardware adicional)

Buffer de bucles (pequeña cache de instrucciones consecutivas)

Útiles solo en el caso de saltos incondicionales.

Flujos múltiples:

Se siguen los dos caminos posibles, duplicando las partes iniciales del hardware.

Puede entrar en el cauce una nueva bifurcación.

Salto retardado:

Modificación del ciclo de instrucción, que requiere reordenamiento del código por parte del compilador.

Luego de cada instrucción de salto hay un *branch delay slot*: la instrucción siguiente se ejecuta SIEMPRE.

60-80% efectivo, si no NOP.



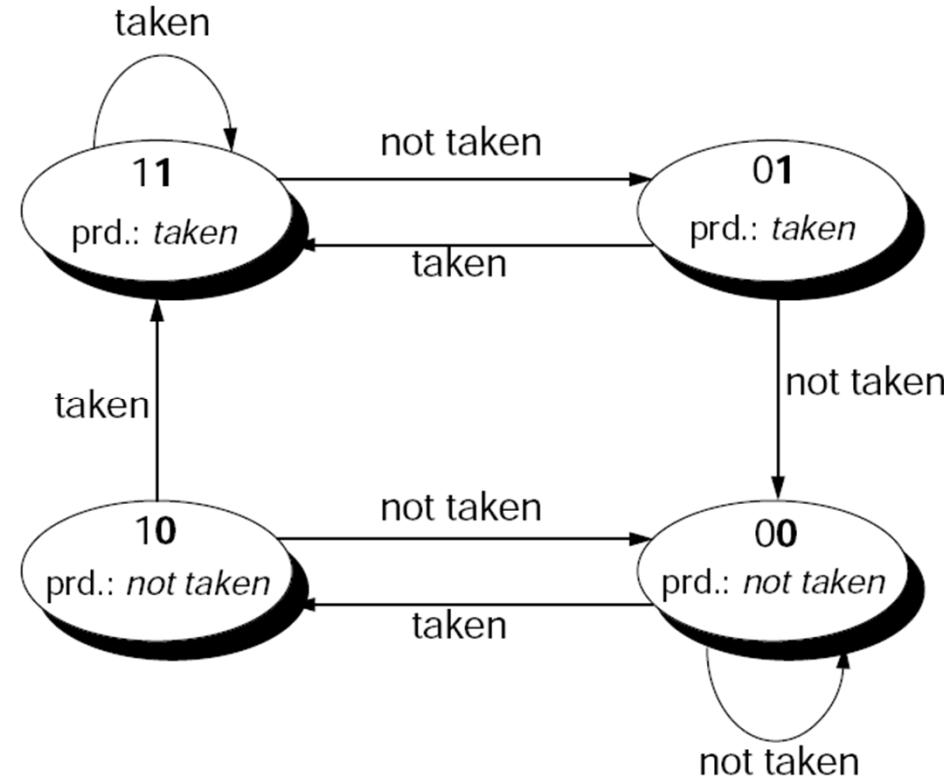
REDUCCIÓN DE LOS EFECTOS

Predicción de saltos:

- **Estática**
 - Siempre salta o nunca salta
 - Depende de la dirección
- **Dinámica**
 - Uno o dos bits (HW) asociados a cada instrucción de salto
 - Tabla de historia de saltos (memoria cache: tabla con dirección de la instrucción de bifurcación + bits de historia + destino)



REDUCCIÓN DE LOS EFECTOS



Esquema típico de predicción dinámica con dos bits: cambiar la predicción solo si suceden dos predicciones incorrectas consecutivas

