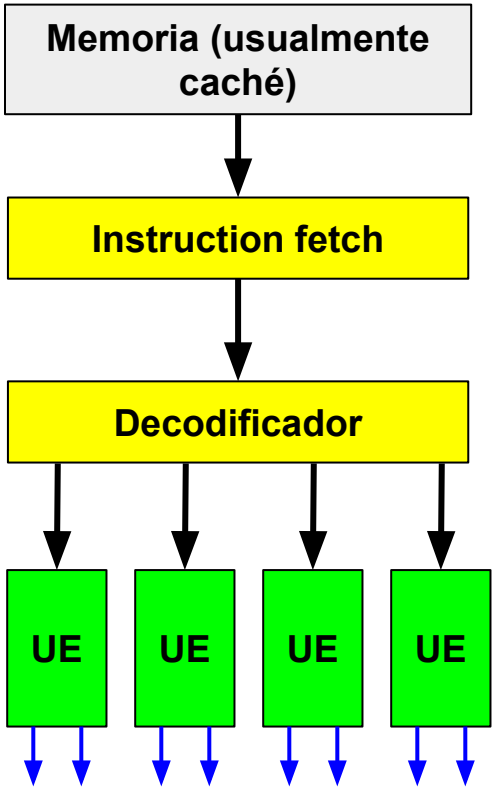
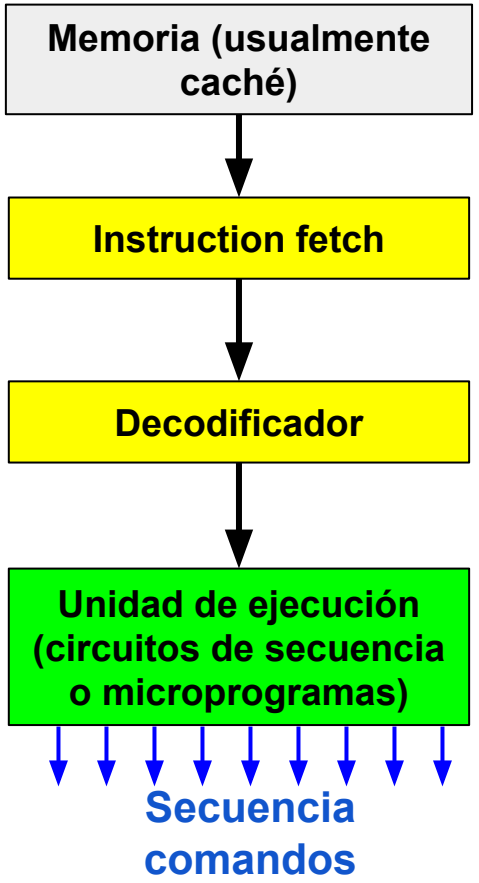
Arquitectura de Computadoras

Unidad 5:
Microprocesadores

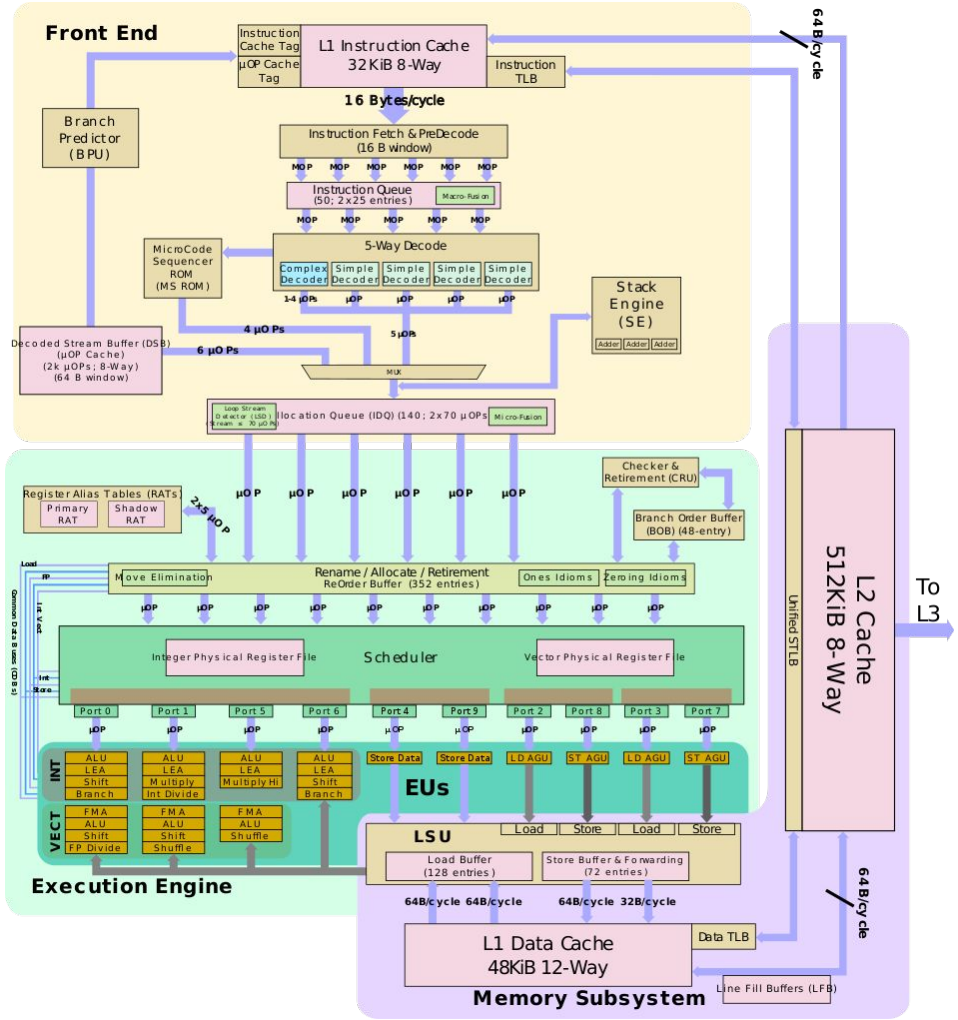
Componentes microarquitectura básicos



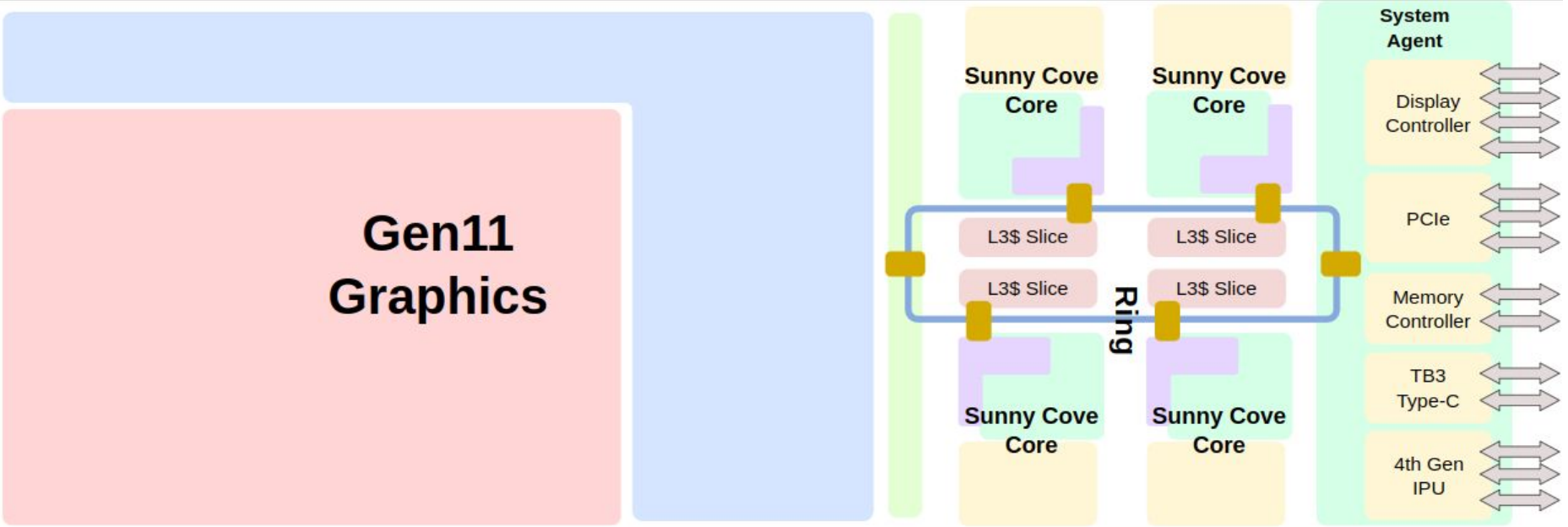


Ejemplo de microarquitectura moderna: Núcleo Sunny Cove

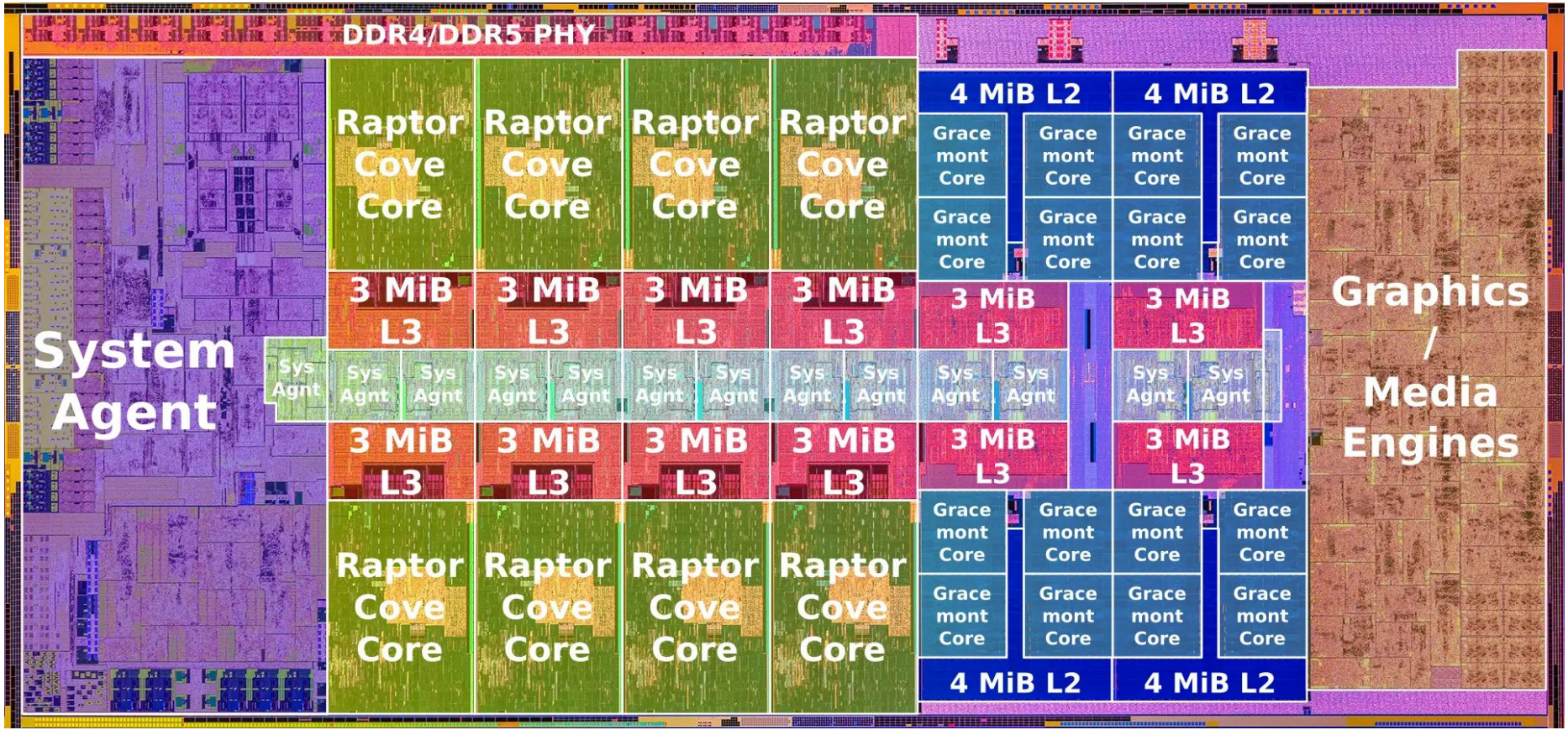
- 2019
- Procesadores Intel Core (i3, i5 e i7, i9, Xeon, etc.)
- 10 nm
- Arquitectura: x86-64



System on Chip (SoC) Ice Lake.

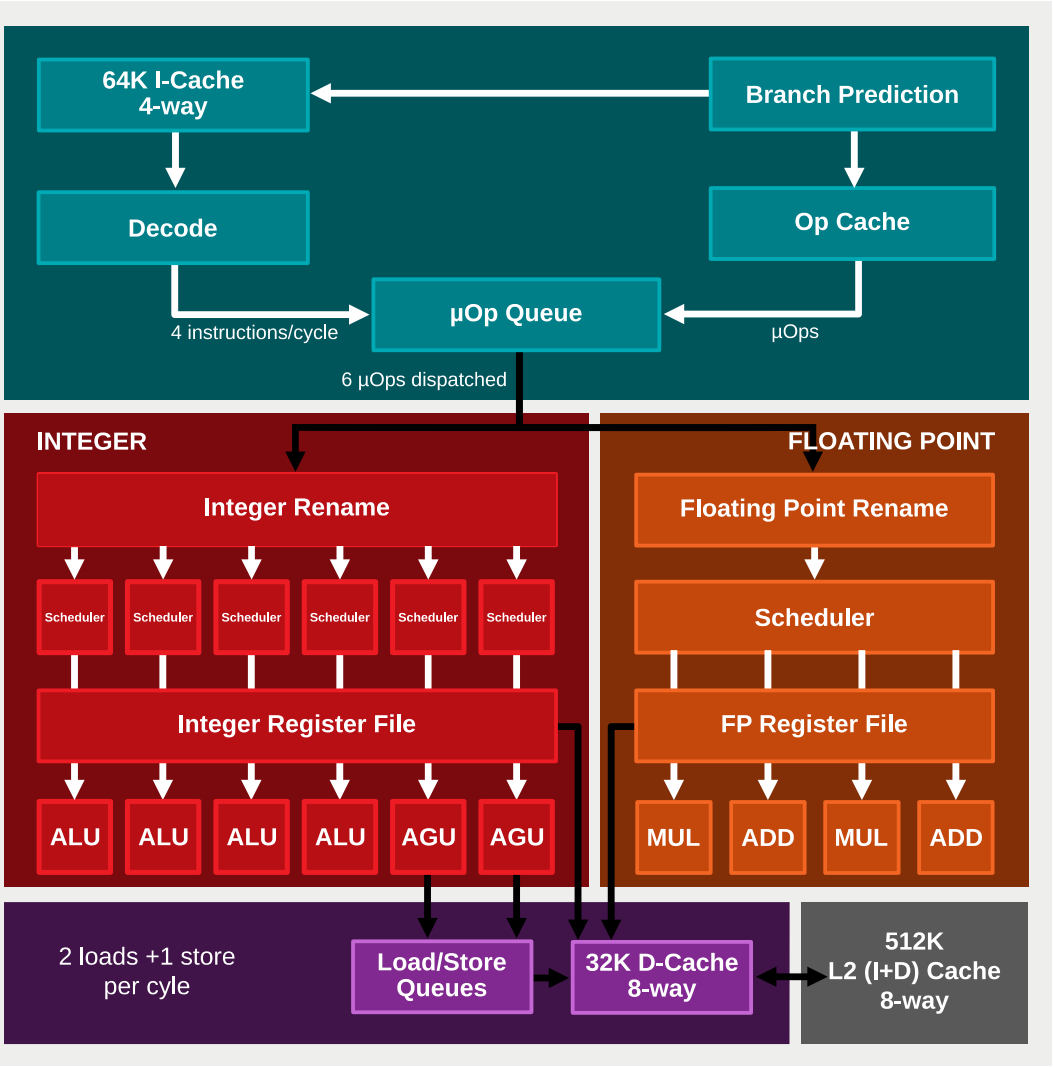


Microarquitectura Raptor Lake (2022)



Ejemplo de Arquitectura moderna: AMD Zen

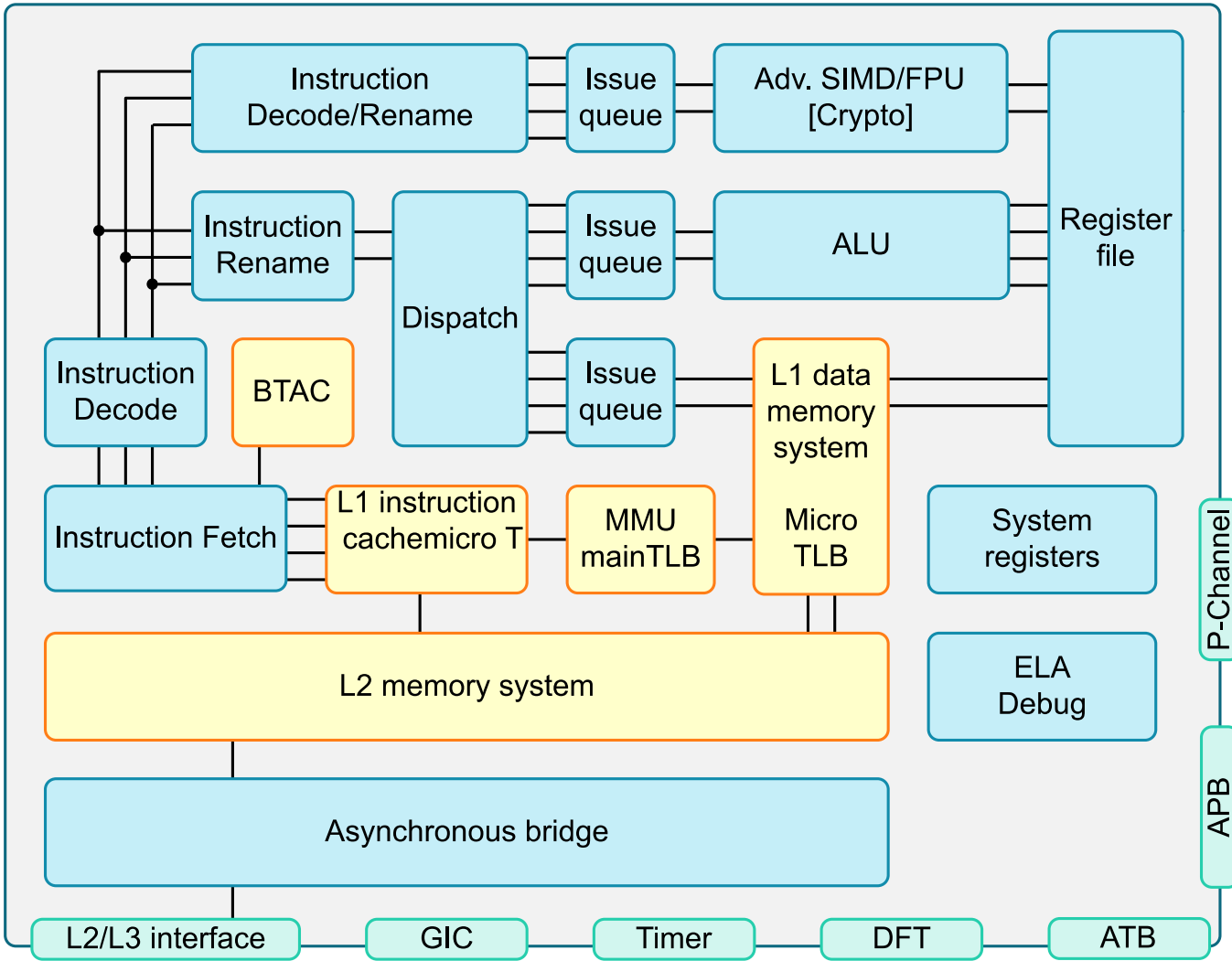
- **2016**
- **Procesadores AMD Ryzen**
- **14 nm**
- **Arquitectura: x86-64**
- Zen 2, Zen 3, Zen 4, Zen 5 (futuro).



Ejemplo de Arquitectura moderna: Cortex A75

- **Junio 2017**
- **Procesadores: Varios fabricantes**
- **7 nm**
- **Arquitectura: ARM-v**

BTAC: Branch Target Address Cache
ELA: Embedded Logic Analysis



Registros

- **Registros visibles al usuario: Pueden direccionarse mediante instrucciones (lenguaje de máquina o ensamblador).**
  - **MOV R1,0x0F34**

- **Registros NO visibles al usuario: NO pueden direccionarse mediante instrucciones (lenguaje de máquina o ensamblador).**

- **Algunos procesadores hacen visibles registros que otros procesadores no, por ejemplo el PC (contador de programa) en algunos procesadores es visible, y en otros no.**

Registros

Registros comúnmente (pero no siempre) visibles al usuario:

- **Registros de datos: Solo contienen datos.**
- **Registros de dirección: contienen direcciones**
  - **Registros de índice: direccionamiento indexado.**
  - **Stack pointer (Puntero de pila).**
- **Registros de estado, Registro de banderas (flags), registros de códigos de condición.**
- **Registros de propósito general: Pueden cumplir varias funciones.**

Registros

Registros comúnmente NO visibles al usuario (Registros de control y estado):

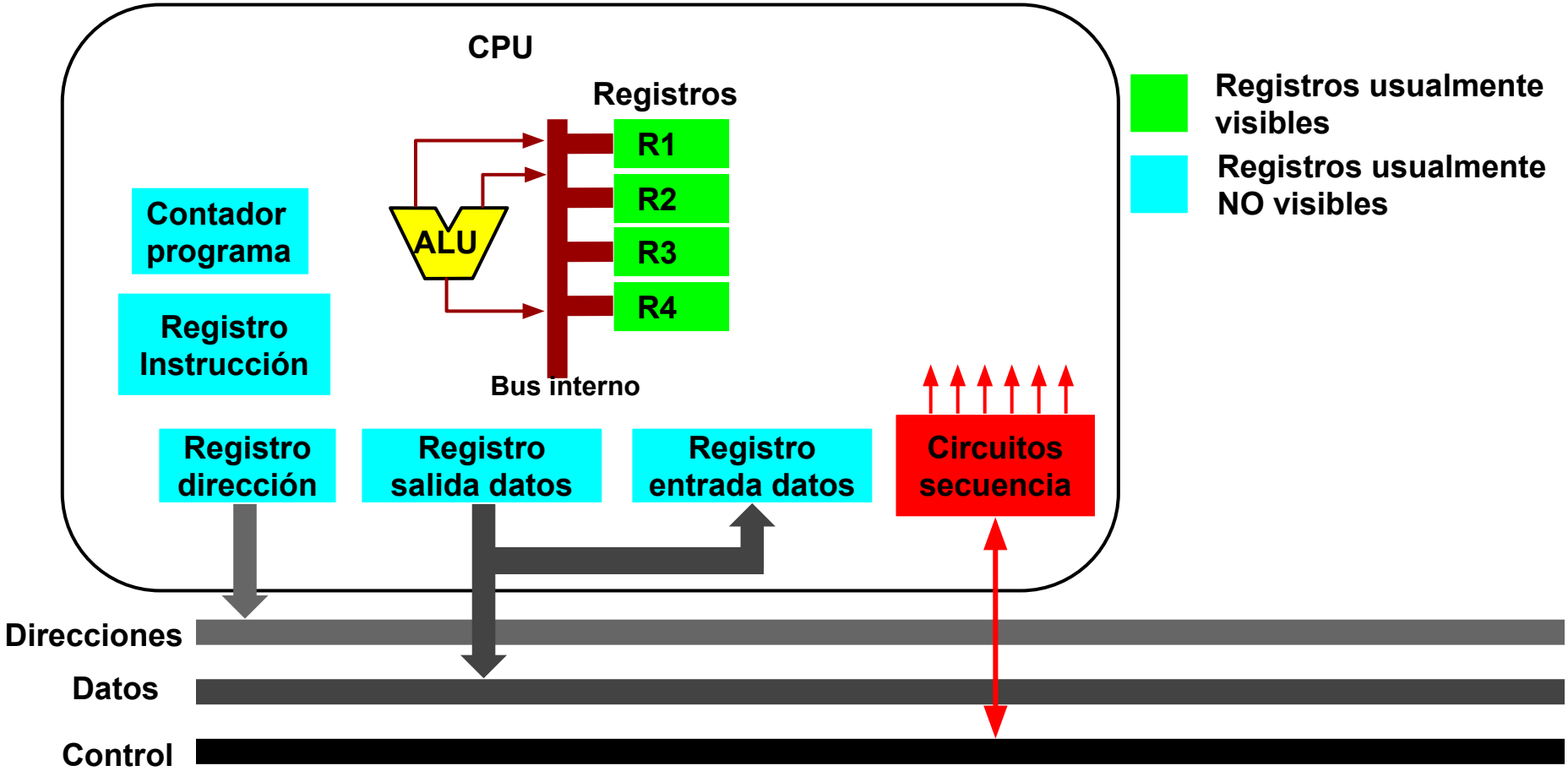
- **Contador de programa (PC)**
- **Registro de Instrucción (IR)**
- **Registro de dirección de memoria (MAR)**
  - **MAR -> bus direcciones**
- **Registro de buffer de memoria (MBR)**
  - **MBR -> bus datos**
- **Registro de vector de interrupción**
- **Stack pointer (puntero de pila)**
- **Algunos registros de estado.**

Registro de Estado

Registros de estado (PSW: program status word). Puede o no ser visible. Contiene varios bits, entre estos:

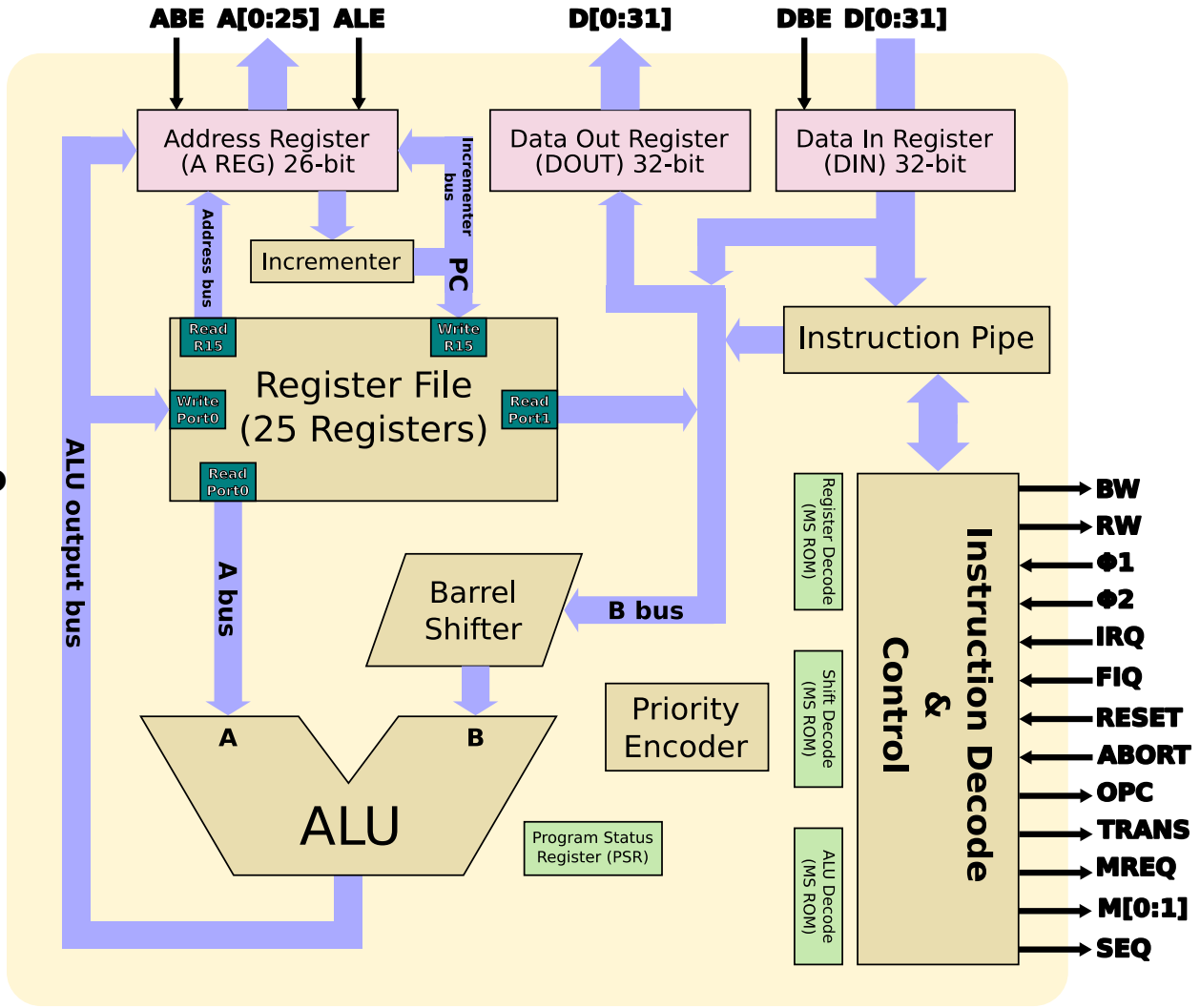
- **Sign (S): Bit de signo de la última operación aritmética.**
- **Sign (N): Set cuando la última operación aritmética es negativa.**
- **Zero (Z): Set cuando el resultado de una operación aritmética es cero.**
- **Carry (C): Set cuando se produce un acarreo.**
- **Equal (Q): Set cuando una comparación lógica da como resultado igual.**
- **Overflow (V): Desborde.**
- **Interrupt Enable/Disable.**
- **Supervisor: Indica si el procesador está ejecutando instrucciones en modo usuario o supervisor.**

Componentes básicos de un procesador



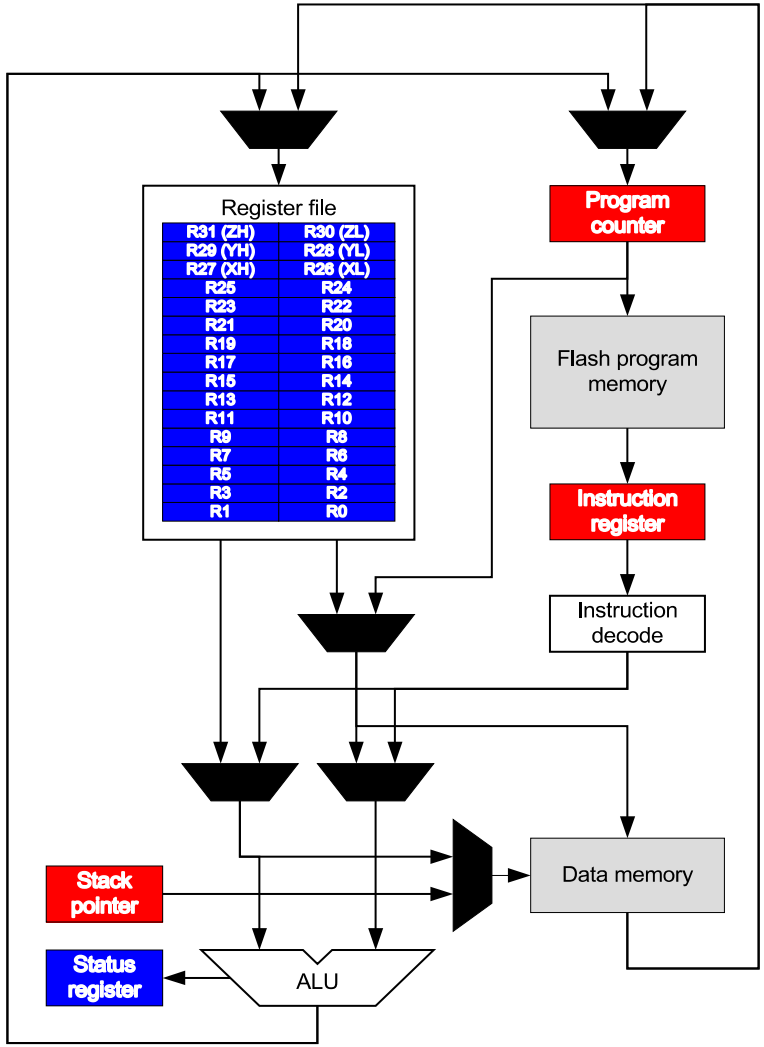
Ejemplo de componentes básicos: ARM1

- Primer ARM
- 1985
- 32 bits
- En lugar de un registro de instrucciones, posee un buffer de instrucciones (Instruction pipe)



Registros: Ejemplo 2

AVR ATmega 328
Arduino UNO



Registros NO visibles

Registros visibles

Figura obtenida de hoja de datos del AVR ATmega 328, pag 25

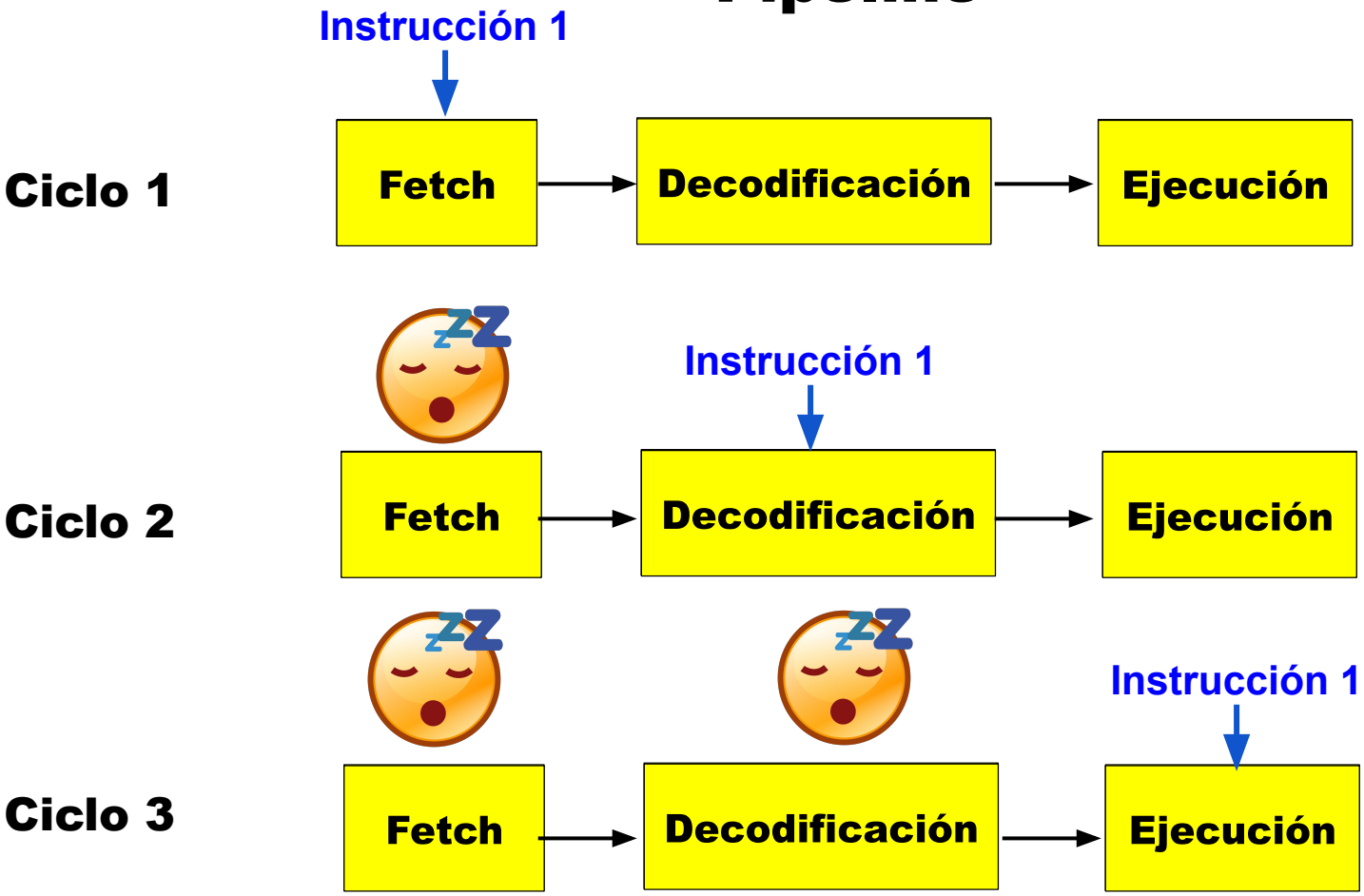
Paralelismo a nivel de instrucción

- **La velocidad puede incrementarse de dos maneras:**
  - **Aumentando la velocidad del reloj (más GHz)**
    - **La distancia máxima que las señales pueden viajar depende del reloj (velocidad de las señales 20cm/nseg).**
      - **1 GHz -> máxima distancia 20 cm**
      - **10 GHz -> máxima distancia 2 cm**
      - **100 GHz -> máxima distancia 2 mm**
    - **Problema: mientras más pequeño el procesador, mayor cantidad de calor por unidad de superficie -> mayor temperatura**
  - **Realizando más de una tarea a la vez.**

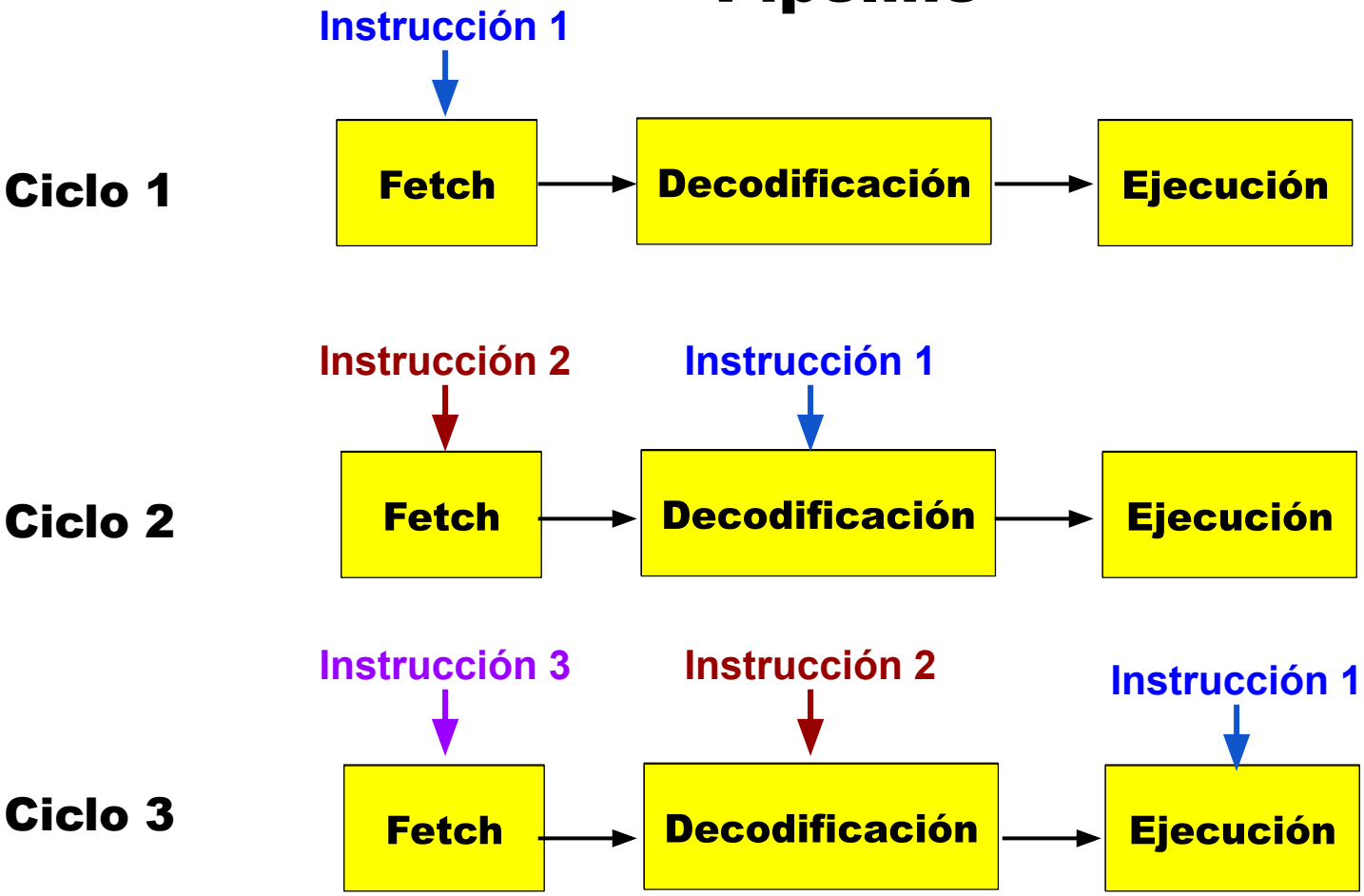
Pipeline

- La ejecución de instrucciones puede dividirse en estados.
- Los estados tienen igual duración (1 ciclo)
- Los estados se ejecutan secuencialmente.
- Permite la paralelización a nivel de instrucción e incremento del speed-up

Pipeline


Instrucción 1
Ciclo 1
Fetch
Decodificación
Ejecución
Instrucción 1
Ciclo 2
Fetch
Decodificación
Ejecución
Instrucción 1
Ciclo 3
Fetch
Decodificación
Ejecución

Pipeline


Instrucción 1
Ciclo 1
Fetch
Decodificación
Ejecución
Instrucción 2
Instrucción 1
Ciclo 2
Fetch
Decodificación
Ejecución
Instrucción 3
Instrucción 2
Instrucción 1
Ciclo 3
Fetch
Decodificación
Ejecución

Pipeline

Sin Pipeline

Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5	Ciclo 6	Ciclo 7	Ciclo 8	Ciclo 9
F1	D1	E1	F2	D2	E2	F3	D3	E3

Con Pipeline

	Ciclo 1	Ciclo 2	Ciclo 3	Ciclo 4	Ciclo 5
Instrucción 1	F1	D1	E1		
Instrucción 2		F2	D2	E2	
Instrucción 1			F3	D3	E3

Tiempo TOTAL

- **Sin pipeline: 9 ciclos**
- **Con pipeline: 5 ciclos**

$$\text{speedup} = \frac{\text{tiempo sin pipeline}}{\text{tiempo con pipeline}} = (9/5) = 1.8$$

El primer procesador que implementó el mecanismo de pipeline fue el Intel 80486. El mismo no era superescalar.

Pipeline:

- **La etapa de ejecución puede ser de mayor duración que las anteriores, porque:**
  - **Puede requerir acceder a memoria para buscar uno o varios operandos.**
  - **Puede requerir acceder a memoria para almacenar un dato o resultado.**
  - **Puede utilizar direccionamiento indirecto -> Por cada operando pueden ser necesarios dos accesos a memoria.**
  - **Puede operar sobre una cadena de caracteres.**
  - **La operación a realizar puede requerir varios ciclos (por ejemplo: operaciones matemáticas complejas o con datos en punto flotante).**

- **Solución:**
  - **Dividir la etapa de ejecución en varias sub-etapas y ejecutar estas sub-etapas en paralelo (Pipeline).**

Ciclo de instrucción

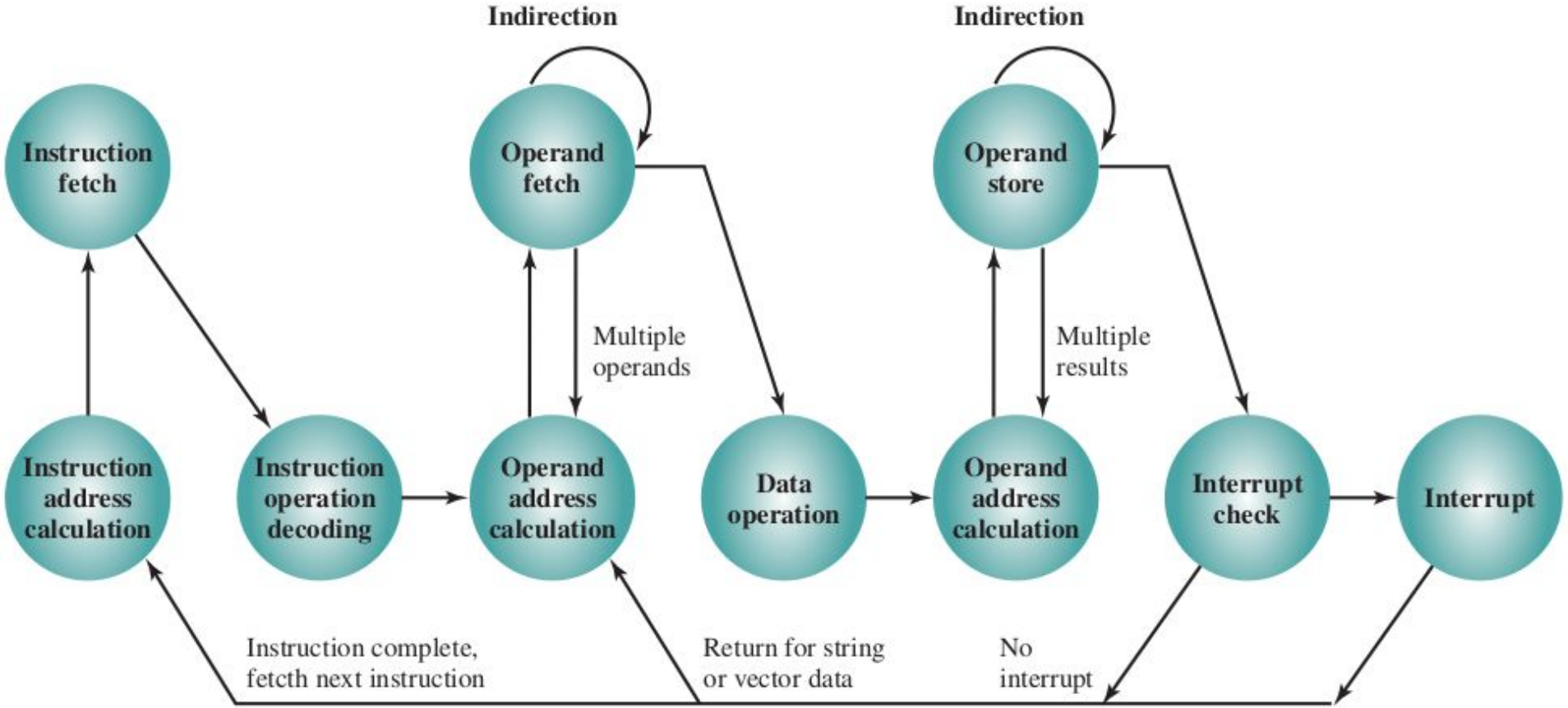
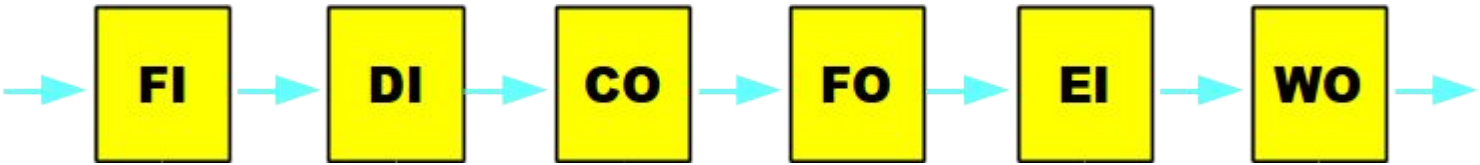


Figura obtenida de Computer Organization and Architecture 9th Edition William Stallings, página 493

Pipeline: Ejemplo 2

Pipeline ideal de 6 estados

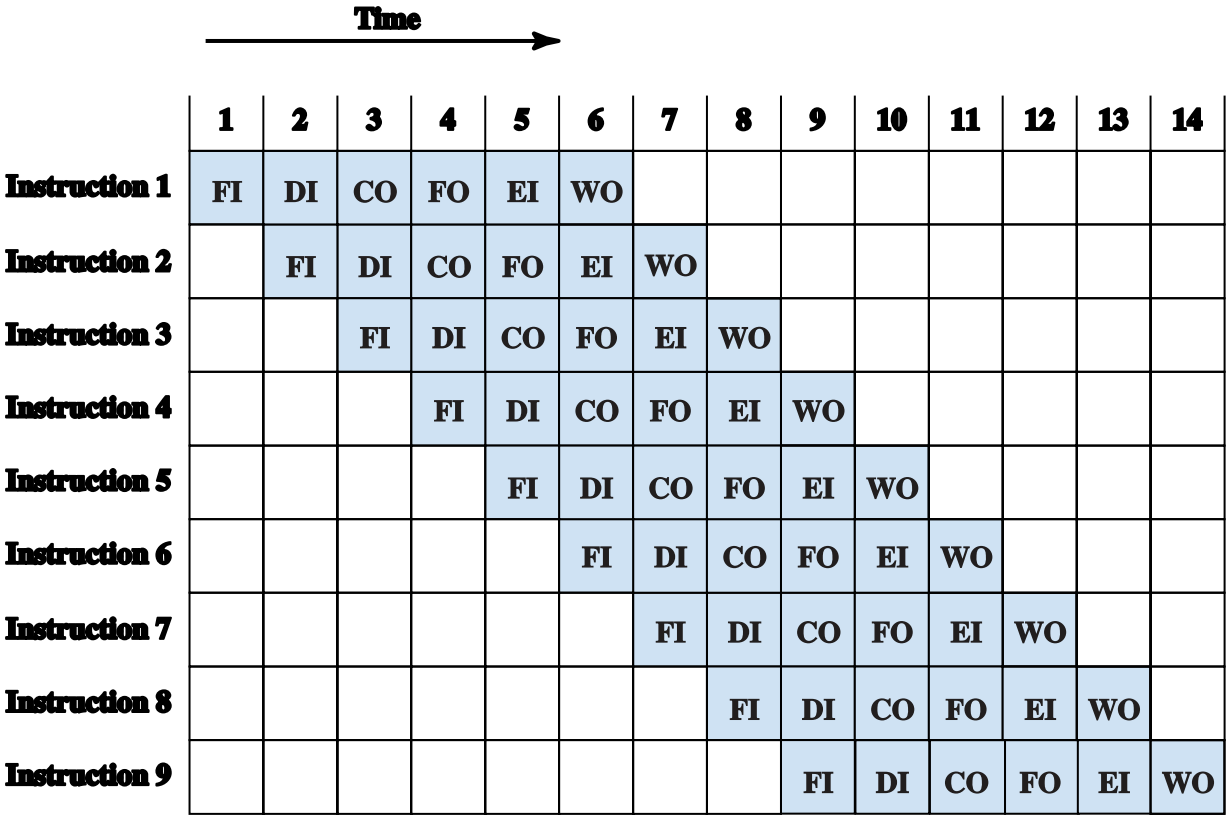
1. **FI: Fetch instruction**
2. **DI: Decode instruction**
3. **CO: Calculate operands (calcula la dirección del operando, direccionamiento indirecto e indexado)**
4. **FO: Fetch operands (trae el operando desde memoria)**
5. **EI: Execute instruction**
6. **WO (Write Operand): Escribir resultado en memoria.**



Ejemplo obtenido de Computer Organization and Architecture 9th Edition William Stallings

Pipeline: Ejemplo 2

Pipeline ideal de 6 estados



Ejemplo y figura obtenidos de Computer Organization and Architecture 9th Edition William Stallings, pag 497

Pipeline: Ejemplo 2

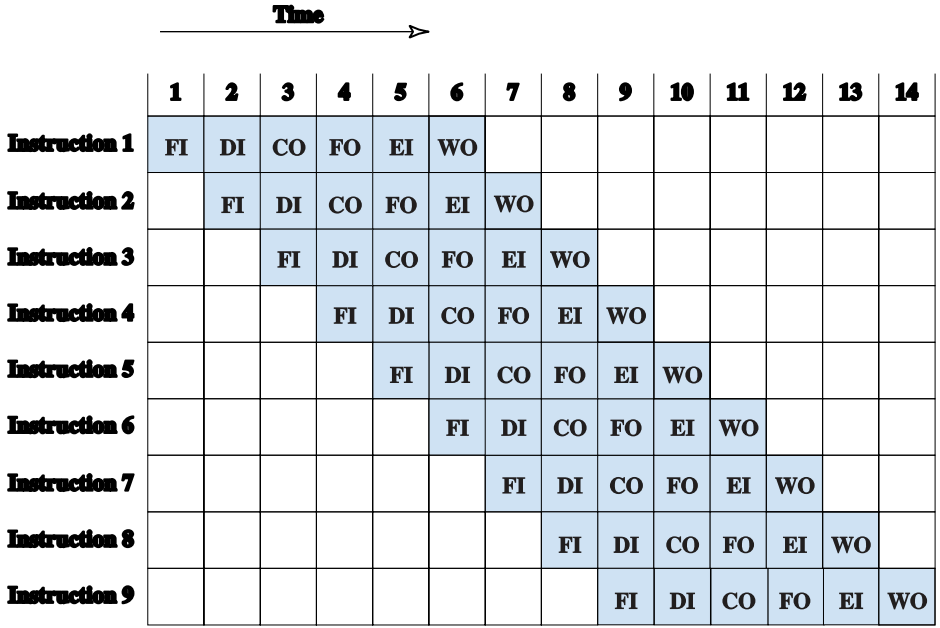
- **Tiempo de ejecución (para 9 instrucciones)**
  - **Sin pipeline: 6 ciclos * 9 Instrucciones = 54 ciclos**
  - **Con pipeline: 14 ciclos (ver figura filmina anterior)**
  - **Speedup=(54/14)=3.86**
- **Salida: aproximadamente una instrucción por ciclo**

Diferencias con un Pipeline Real

- **No todas las instrucciones son iguales**
  - **Algunas pueden no requerir acceso a memoria**
  - **Algunas pueden requerir un acceso a memoria (direccionamiento directo)**
  - **Algunas pueden requerir varios accesos a memoria (direccionamiento indirecto o varios operandos)**
- **Las instrucciones pueden ser de diferente longitud (Ejemplo: x86)**

Pipeline Speedup (Speedup factor)

$$\text{speedup} = \frac{\text{tiempo sin pipeline}}{\text{tiempo con pipeline}}$$



T: tiempo de un ciclo
n: número de instrucciones
k: número de estados
d: overhead

$$\text{speedup} = \frac{n*k*T}{[k+(n-1)]*T}$$

Si $n \to \infty \Longrightarrow$ speedup $\to k$

Ejemplo Pipeline : Intel

Microarquitectura	Año	Procesador	Prof. Pipeline	Vias superescalar	En orden	Bus	Tecnología
486	1989	i486,	5	1	En orden	32 bits	1 µm
P5	1993	Pentium	5	2	En orden	32 bits	0,6 µm
P6	1995	Pentium Pro, Pentium II y III	10-14*	5	Fuera de orden	32 bits	350 nm
NetBurst (P7)	2000	Pentium IV, Celeron	20-31	6	Fuera de orden	32 y 64 bits	180 nm
Nehalem	2008	I3, i5, i7	20	5	Fuera de orden	64 bits	45 nm
SkyLake	2017	I3, i5, i7	14	5	Fuera de orden	64 bits	14 nm

* Algunos procesadores usaron superpipeline

Ejemplo Pipeline : Intel

Microarquitectura	Año	Procesador	Prof. Pipeline	Vias superescalar	En orden	Bus	Tecnología
Ice Lake	2019	I3, i5, i7	14-19	5	Fuera de orden	64 bits	10 nm

Ejemplo Pipeline: AMD

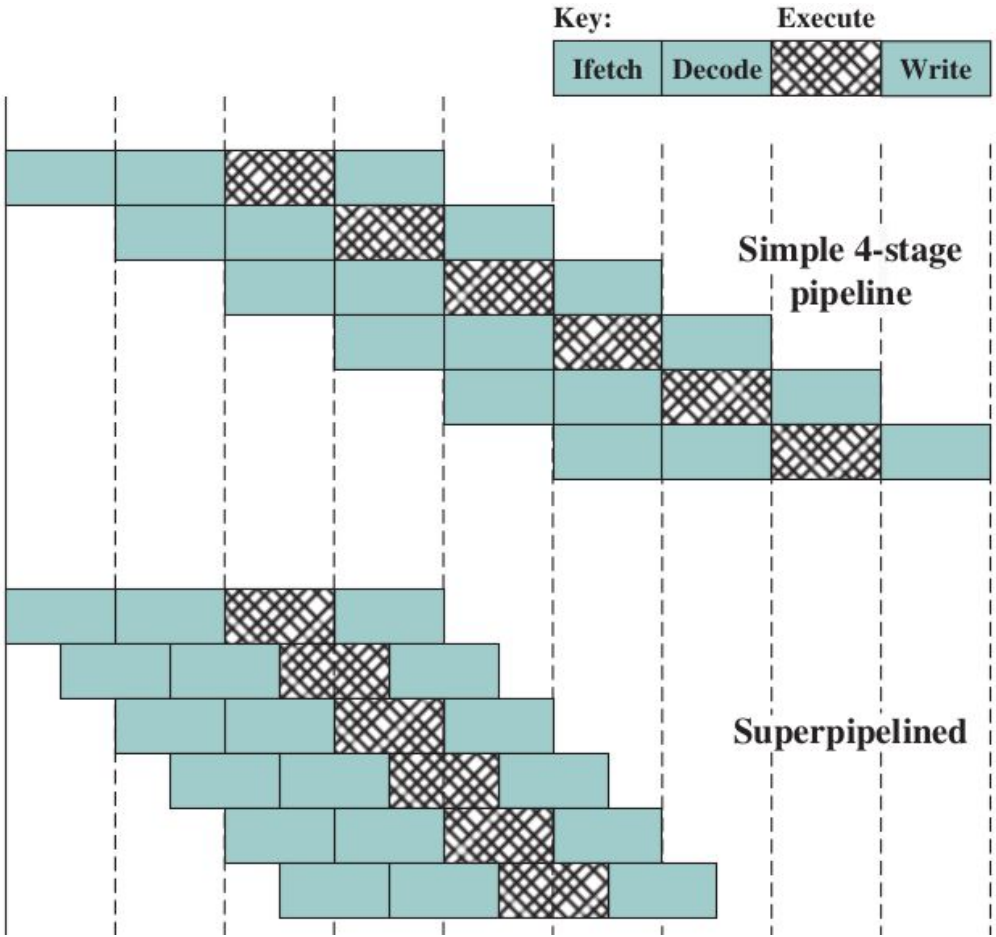
Microarquitectura	Año	Procesador	Prof. Pipeline	Vias supere scalar	En orden	Bus	Tecnología
K5	1996	K5	5	3	Fuera de orden	32 bits	350-500 nm
k6	1997	K6	6	4	Fuera de orden	32 bits	350 m,
k8	2003	Sempron, Opteron, Athlon	12	4	Fuera de orden	64 bits	65 nm
Bulldozer	2011	AMD FX, Opteron	20	4	Fuera de orden	64 bits	32 nm
Zen	2017	Ryzen, EPYC	20	5	Fuera de orden	64 bits	14 nm
Zen 3	2020	Ryzen	19	7	Fuera de orden	64 bits	7 nm

Antes del K5, AMD utilizaba las mismas microarquitecturas de Intel (bajo licencia)

Ejemplo Pipeline: ARM

Microarquitectura	Año	Procesador	Prof. Pipeline	Vias superescalar	En orden	Bus	Tecnología
ARM Cortex-A9	2007	Apple A5	8-12	2	Fuera de orden	32 bits	40 -65 nm
ARM Cortex-A53	2012	AMD Opteron A1100, Qualcomm Snapdragon 810, Nvidia Tegra X1	8	2	En orden	32 bits	10-40 nm
ARM Cortex-A76	2017	Samsung Exynos 990, Qualcomm Snapdragon 855	11-13	4	Fuera de orden	64 bits	5-12 nm
ARM Cortex-A78	2020	Qualcomm Snapdragon 888 Samsung Exynos 1080	13	4	Fuera de orden	64 bits	5 nm

Super pipeline



Superpipeline

- Muchos tareas requieren menos de un ciclo de reloj.
- Se divide cada estado en n partes (sub-estados) no superpuestas (pueden ejecutarse en paralelo).
- Se emplea un reloj adicional cuya frecuencia es n veces de la frecuencia del reloj principal.
- Ejemplo: MIPS R4000 (1991, 64 bits).
- Este enfoque no fue muy desarrollado por la aparición de los procesadores superescalares, los cuales fueron superiores en cuanto a aumentar el speedup a menor costo.

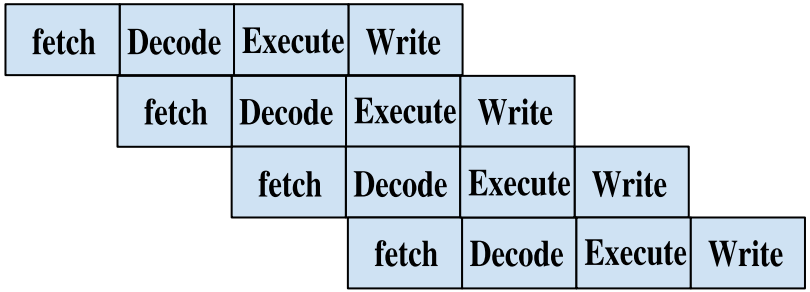
Procesadores Superescalares

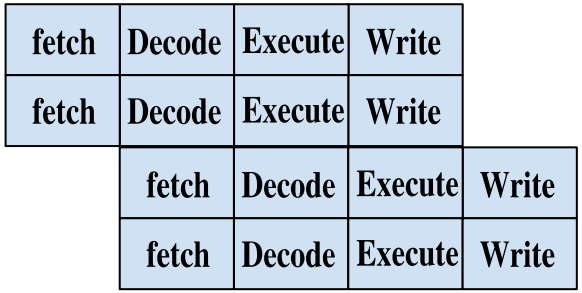
- **Limitaciones de la técnica de pipeline:**
  - **La máxima velocidad que puede alcanzarse con un pipeline perfecto es 1 instrucción por ciclo.**
  - **El enfoque de superpipeline incrementa en gran medida la complejidad del procesador, incrementando su costo.**
- **Fundamentos de los procesadores superescalares:**
  - **No todas las instrucciones son iguales, podrían ejecutarse dos instrucciones diferentes al mismo tiempo.**
  - **Pueden duplicarse unidades de ejecución y ejecutar instrucciones iguales al mismo tiempo.**

Un procesador en pipeline ejecuta instrucciones en paralelo, pero cada instrucción en un estado diferente. Un procesador superescalar ejecuta instrucciones en paralelo, estando dos o más instrucciones en el mismo estado.

Procesadores Superescalares

- **Procesador superescalar: Las instrucciones son iniciadas simultáneamente y ejecutadas independientemente y simultáneamente en diferentes pipelines.**





Pipeline: solo una instrucción en fetch, solo una en Decode, solo una en Execute, etc. al mismo tiempo.

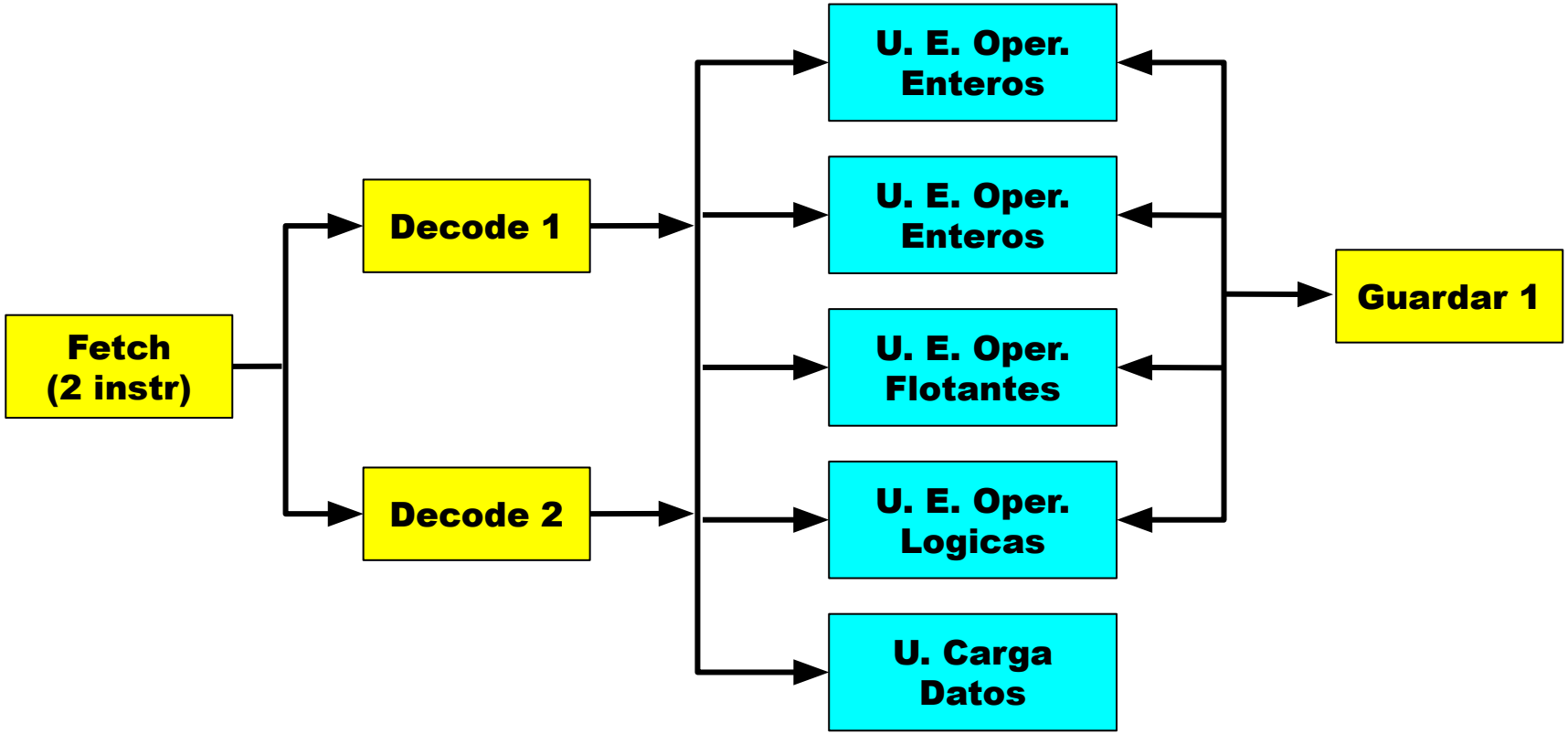
Speedup (máximo ideal) = k

Superescalar: dos instrucciones en fetch, dos en Decode, dos en Execute, etc al mismo tiempo.

Speedup (máximo ideal) = m*k

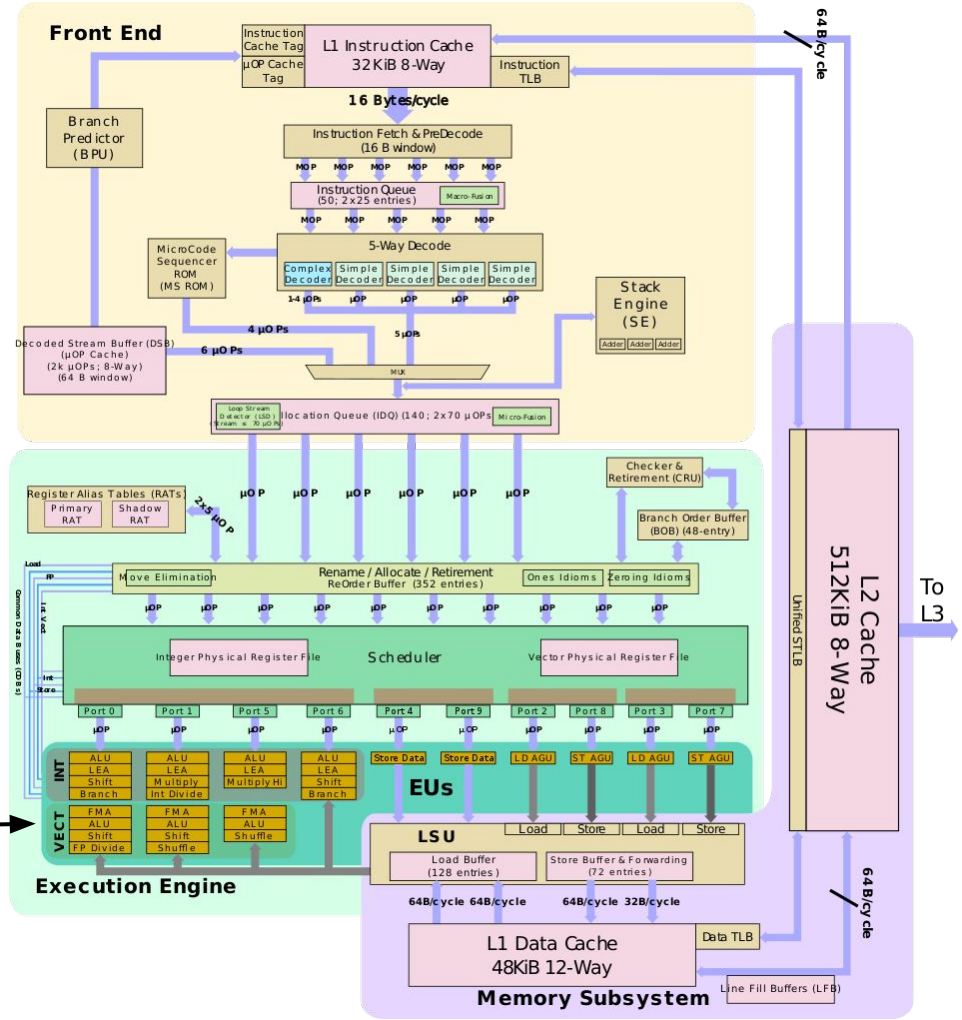
El primer procesador superescalar fue el Intel Pentium (1993), el cual tenía dos unidades de ejecución de enteros separados.

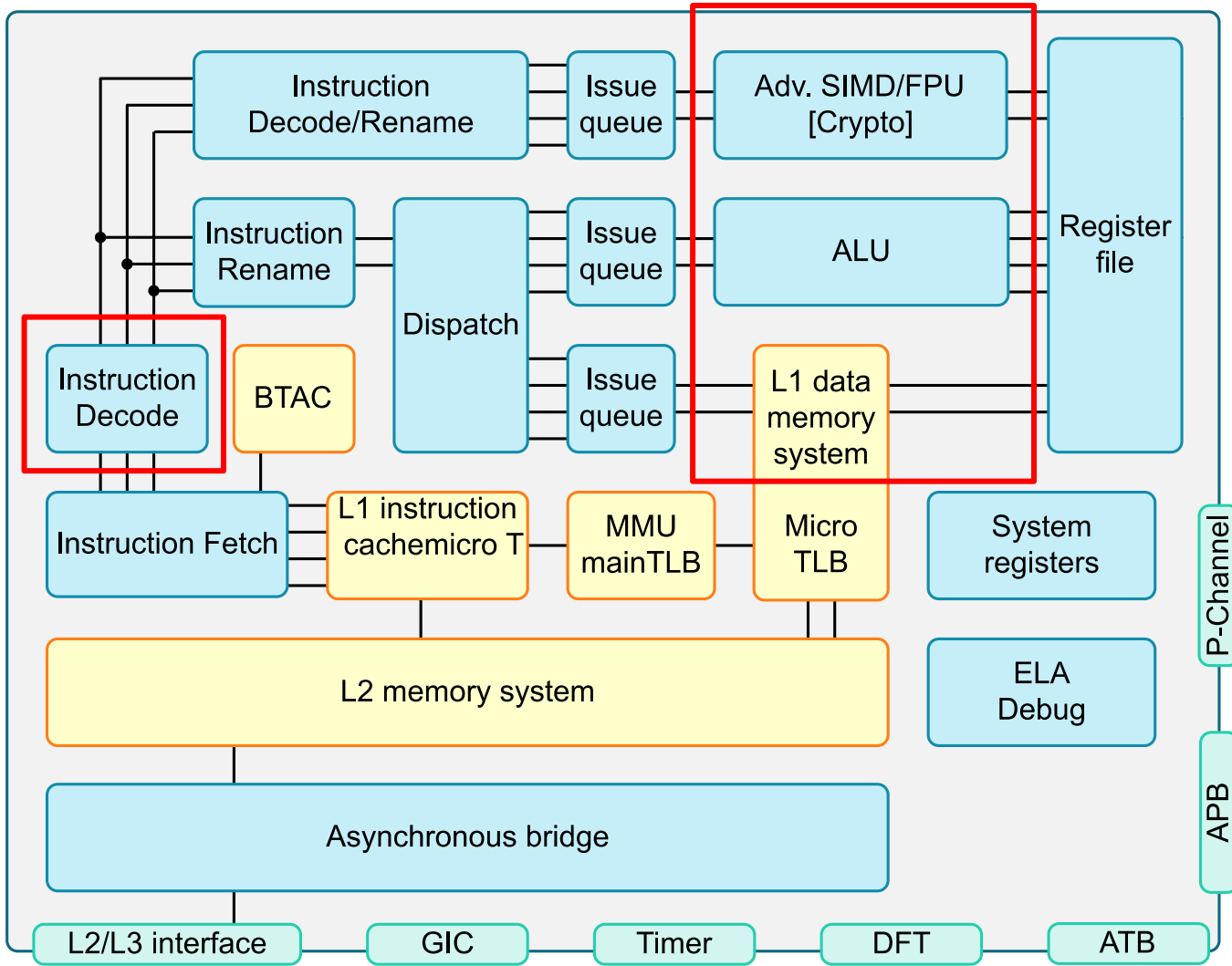
Procesadores Superescalares



AGU: address generation unit
AES, SHA: Algoritmos de encriptación

Unidades de ejecución




Instruction Decode/Rename
Issue queue
Adv. SIMD/FPU [Crypto]
Register file
Instruction Rename
Dispatch
Issue queue
ALU
Instruction Decode
BTAC
Issue queue
L1 data memory system
Instruction Fetch
L1 instruction cachemicro T
MMU mainTLB
Micro TLB
System registers
L2 memory system
ELA Debug
Asynchronous bridge
L2/L3 interface
GIC
Timer
DFT
ATB
P-Channel
APB

Problema 1: Dependencia de datos
(Dependencia RAW o dependencia Real)

- **Problema que afecta a ambas técnicas (pipeline o superescalar)**
- **La instrucción "n+1" depende del resultado de la instrucción "n".**
  - **Ejemplo 1**

    3) ADD (0x2F03),r16 (0x2F03) <- r16 + (0x2F03)

    4) MOV r17,(0x2F03) r17 <- (0x2F03)

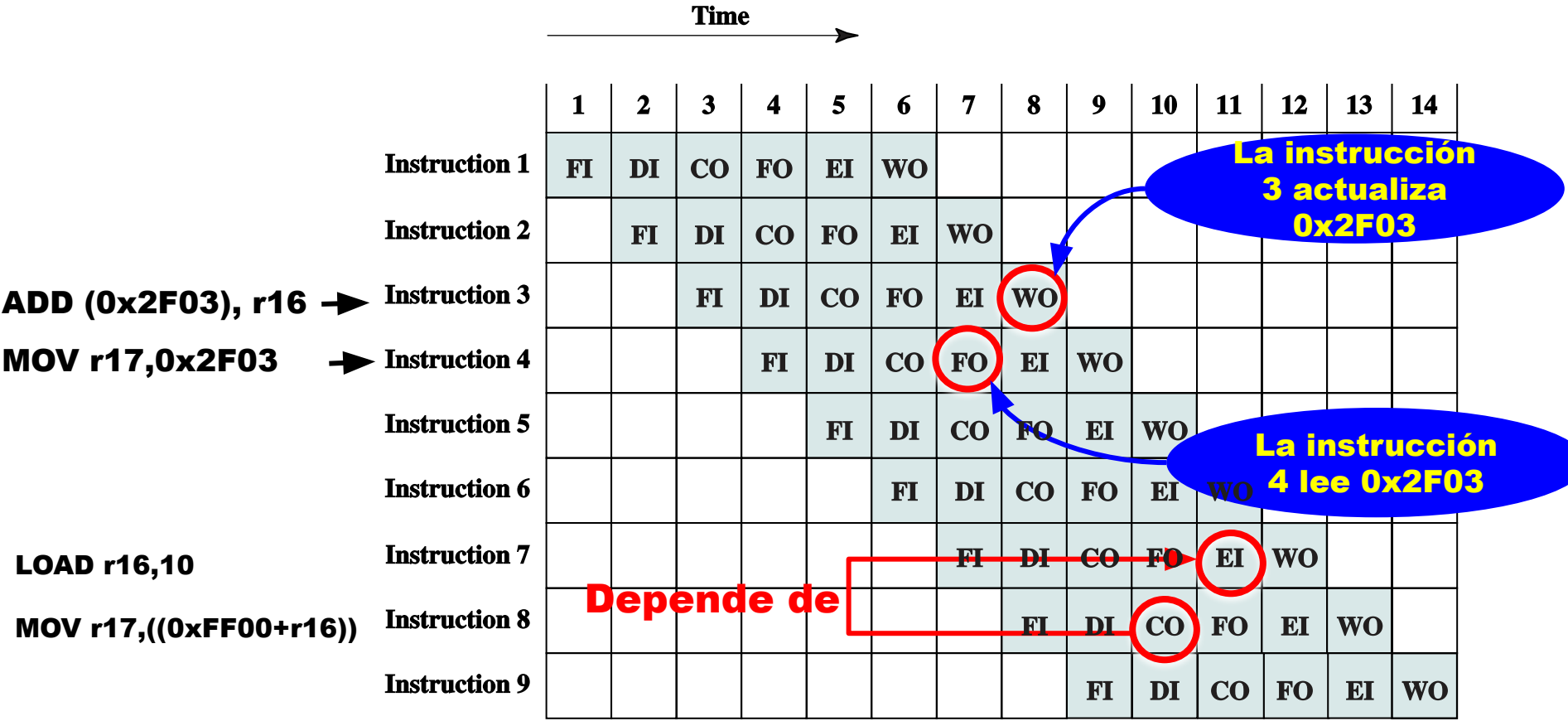
(0x2F03)=2
r16=3

(0x2F03) < 5
r17 < 2

Problema: Primero se debe realizar la suma y luego guardar el resultado en la posición de memoria 0x2F03. Pero la instrucción 2 lee el contenido de r16 antes de que la instrucción 1 lo haya actualizado con la suma (ver siguiente filmina)

  - **Ejemplo 2:**
    El cálculo de una dirección (CO) depende del resultado de una instrucción anterior.

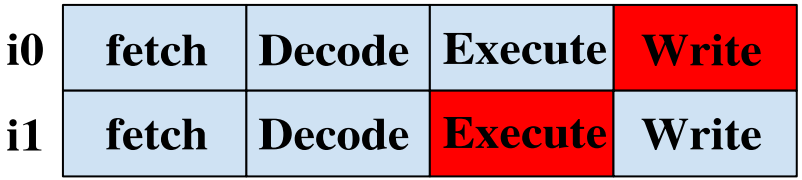
Dependencia de datos en el pipeline



Ejemplo y figura obtenidos de Computer Organization and Architecture 9th Edition William Stallings, pag 497

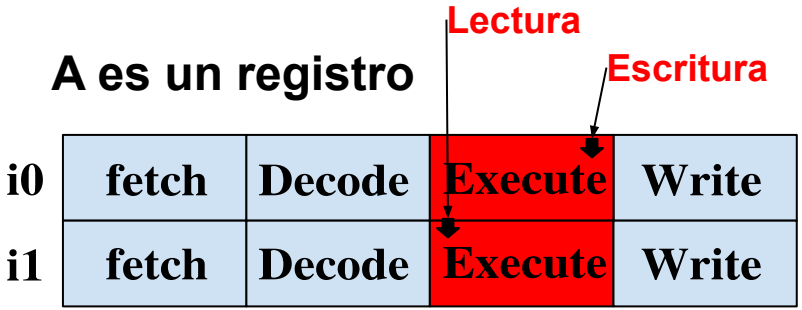
Dependencia de datos en superescalar

i0 escribe el resultado en una posición de memoria, i1 lee esa posición de memoria



i0 escribe el resultado en un registro, i1 lee ese registro

A es un registro

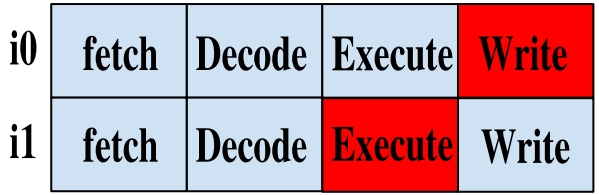


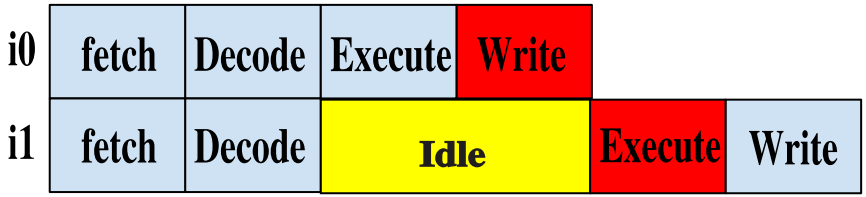
ADD A,B (A <- A+B)
MULI A,0x02 (A <- A*2)

Ejemplo y figura basados en Computer Organization and Architecture 9th Edition William Stallings, pag 504 y 578

Dependencia de datos

- **Solución 1: Retrasar el pipeline para esperar a que el conflicto se solucione (Estado Idle)**
- **Los procesadores poseen circuitos que detectan las dependencias de datos y retrasan el pipeline**





Dependencia de datos

- **Solución 1: Retrasar el pipeline para esperar a que el conflicto se solucione (Estado Idle)**

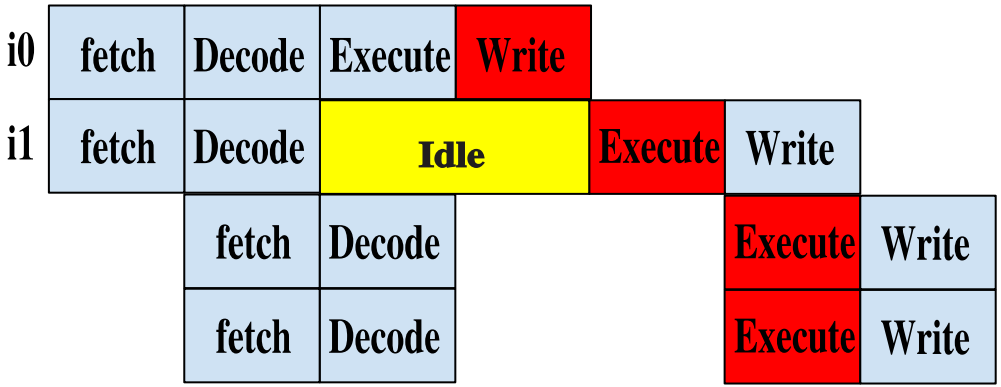
I1	FI	DI	FO	EI	WO						
I2		FI	DI	Idle		FO	EI	WO			
I3			FI			DI	FO	EI	WO		
I4						FI	DI	FO	EI	WO	
I5							FI	DI	FO	EI	WO

- **Esperar será una solución posible para todos los problemas que aparecen en la ejecución de instrucciones en paralelo.**
- **Esperar No es la solución óptima (retrasa el pipeline)**
- **Se necesitan circuitos adicionales que "detecten" las dependencias.**

Ejemplo y figura basados en Computer Organization and Architecture 9th Edition William Stallings, pag 504 y 578

Dependencia de datos

- **Si la instrucción tiene que buscar un dato en un registro, se pierde un ciclo. Si está en una Caché, se perderán varios ciclos (dos o más). Si está en la RAM, se perderán decenas de ciclos.**
- **En un procesador superescalar, se detienen TODAS las vías de ejecución de instrucciones**



Dependencia de datos

- **Solución 2: reordenar las instrucciones:**
  - **A nivel compilación**
  - **A nivel de ejecución: Procesador fuera de orden**

1) A = A + B
2) A = A*2
3) X = 2
4) Z = X + Y

1) A = A + B
3) X = 2
4) Z = X + Y
2) A = A*2

1) A = A + B
3) Z = X + Y
4) NOP (No Operation)
2) A = A*2

I1	FI	DI	FO	EI	WO			
I2		FI	DI	FO	EI	WO		
I3			FI	DI	FO	EI	WO	
I4				FI	DI	FO	EI	WO

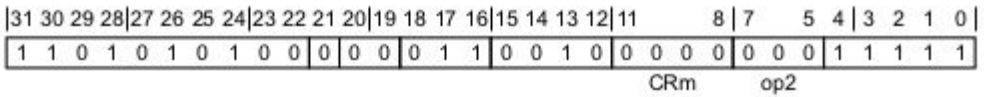
I1	FI	DI	FO	EI	WO			
I3		FI	DI	FO	EI	WO		
I4			FI	DI	FO	EI	WO	
I2				FI	DI	FO	EI	WO

C6.2.183 NOP

No Operation does nothing, other than advance the value of the program counter by 4. This instruction can be used for instruction alignment purposes.

Note

The timing effects of including a NOP instruction in a program are not guaranteed. It can increase execution time, leave it unchanged, or even reduce it. Therefore, NOP instructions are not suitable for timing loops.



System variant

NOP

Decode for this encoding

```
// Empty.
```

Operation

```
// do nothing
```

Figura obtenida de ARM Architecture Reference Manual, ARMv8, for ARMv8-A architecture profile. Página 817.

Problema 2: Conflicto de recursos
(Conflictos estructurales)

Dos instrucciones intentan acceder a un mismo recurso al mismo tiempo. Posibilidades:

- **Dos instrucciones intentan acceder al mismo recurso (UE), etc.**
  - **Solución 1: Retrasar el (o un) pipeline.**
  - **Solución 2: Duplicar recursos.**
  - **Solución 3: Cambiar orden de ejecución de las instrucciones.**
- **Dos instrucciones intentan leer o escribir un operando en memoria.**
  - **Solución 1: Retrasar el pipeline.**
  - **Solución 2: Cambiar orden de ejecución de las instrucciones.**
- **Una instrucción intenta leer o escribir un operando en memoria mientras se está buscando la siguiente instrucción (Fetch).**
  - **Solución: Caché L1 de instrucciones + Caché L1 de Datos.**

Nuevamente, retrasar el pipeline es siempre solución, pero se debe evitar empleando las soluciones mencionadas para no disminuir la performance

Pipeline: Acceso a un mismo recurso

Intenta leer un operando

Intenta leer la siguiente instrucción

	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			FI	DI	FO	EI	WO		
I4				FI	DI	FO	EI	WO	

Pipeline

Instrucción punto flotante

Instrucción punto flotante, pero hay una sola unidad de punto flotante

i0	fetch	Decode	Execute	Write
i1	fetch	Decode	Execute	Write

Superescalar

Ejemplos y figuras basados en Computer Organization and Architecture 9th Edition William Stallings pag 504 y 578

Pipeline: Acceso a un mismo recurso

Solución 1: Retrasar el pipeline

	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			Idle	FI	DI	FO	EI	WO	
I4					FI	DI	FO	EI	WO

	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2	FI	DI	FO	Idle	EI	WO			
I3		FI	DI	FO		EI	WO		
I4		FI	DI	FO		EI	WO		

Solución 2: Duplicar recursos

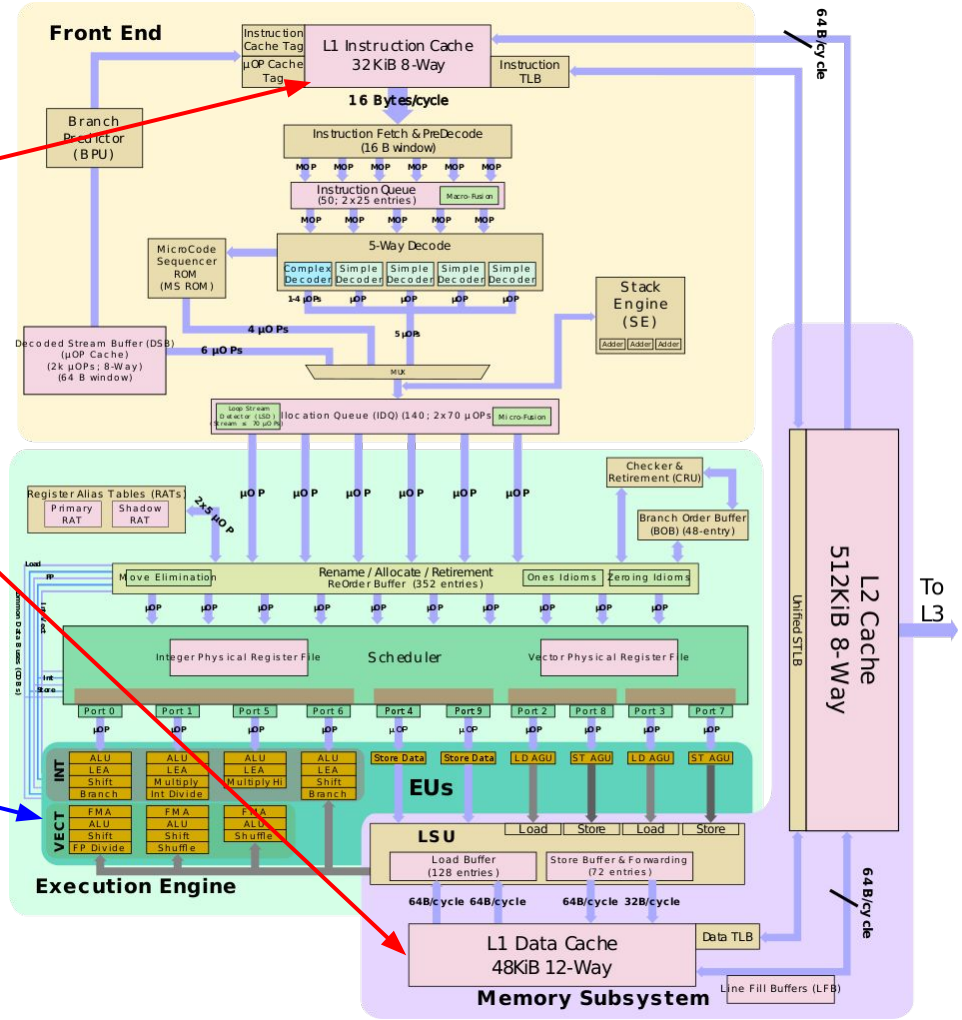
Ambas instrucciones Necesitan el mismo recurso, pero el recurso está duplicado

	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2	FI	DI	FO	EI	WO				
I3		FI	DI	FO	EI	WO			
I4		FI	DI	FO	EI	WO			

Ejemplos y figuras basados en Computer Organization and Architecture 9th Edition William Stallings pag 504 y 578

L1 dividida

Duplicación de recursos



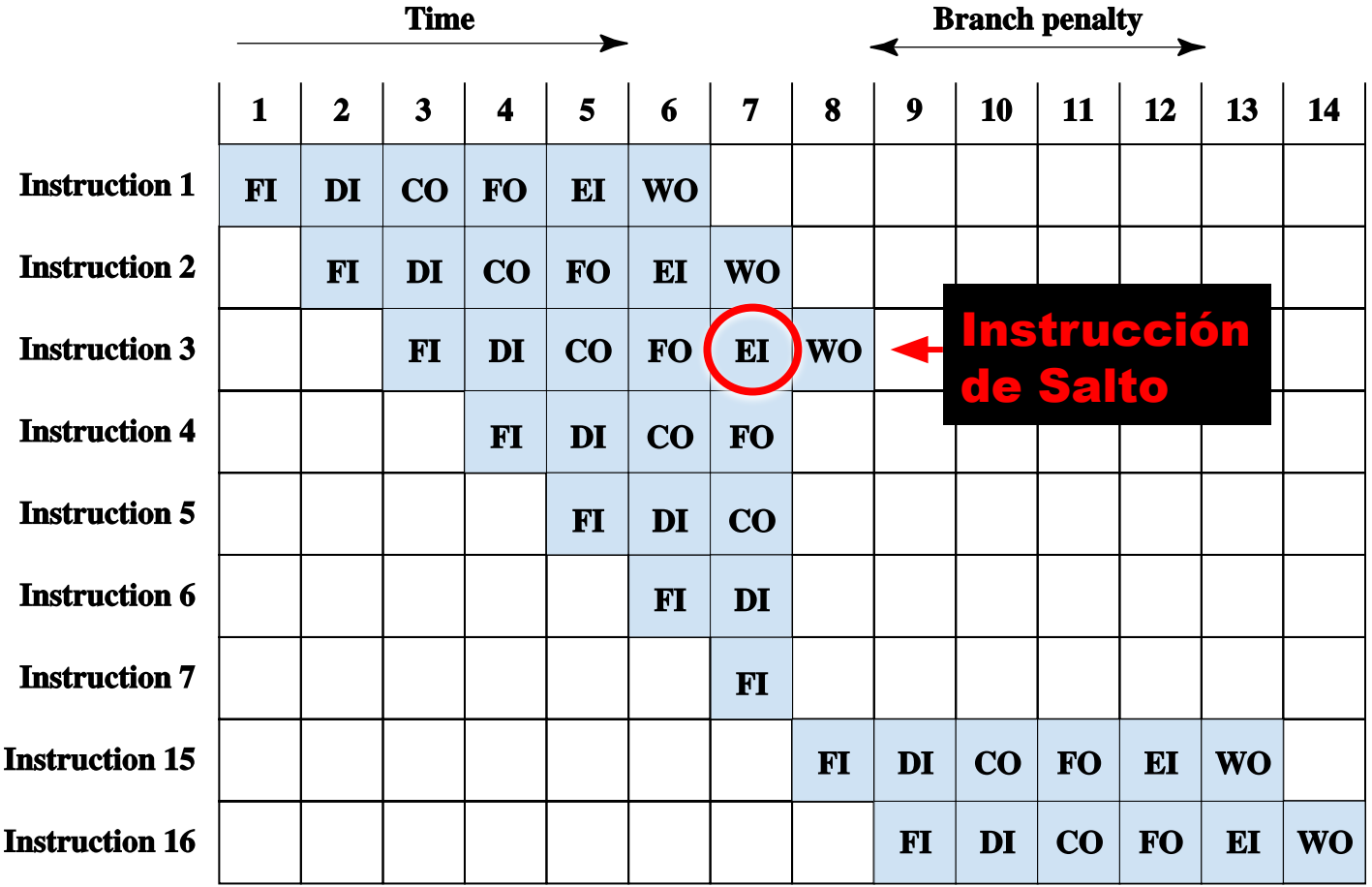
Problema 3: Dependencias de control
(riesgos de control, dependencias procedurales o branch hazard)

- **Afecta a ambas técnicas (pipeline o superescalar). Debida a:**
  a. **Saltos:**
     i. **Incondicionales (jump, goto).**
     ii. **Condicionales (jumpZ).**
  b. **Instrucciones de diferentes longitudes.**

- **Saltos condicionales: Solo en el ciclo de ejecución se sabrá si hay que hay que hacer un salto.**
  a. **Las instrucciones posteriores que hayan comenzado a procesarse (entrado al pipeline) se pierden -> pérdida de tiempo**
  b. **Se deben cargar desde el comienzo las nuevas instrucciones.**
  c. **En un programa el 25-30% de las instrucciones que se ejecutan son saltos[1] (muy utilizadas en lazos).**

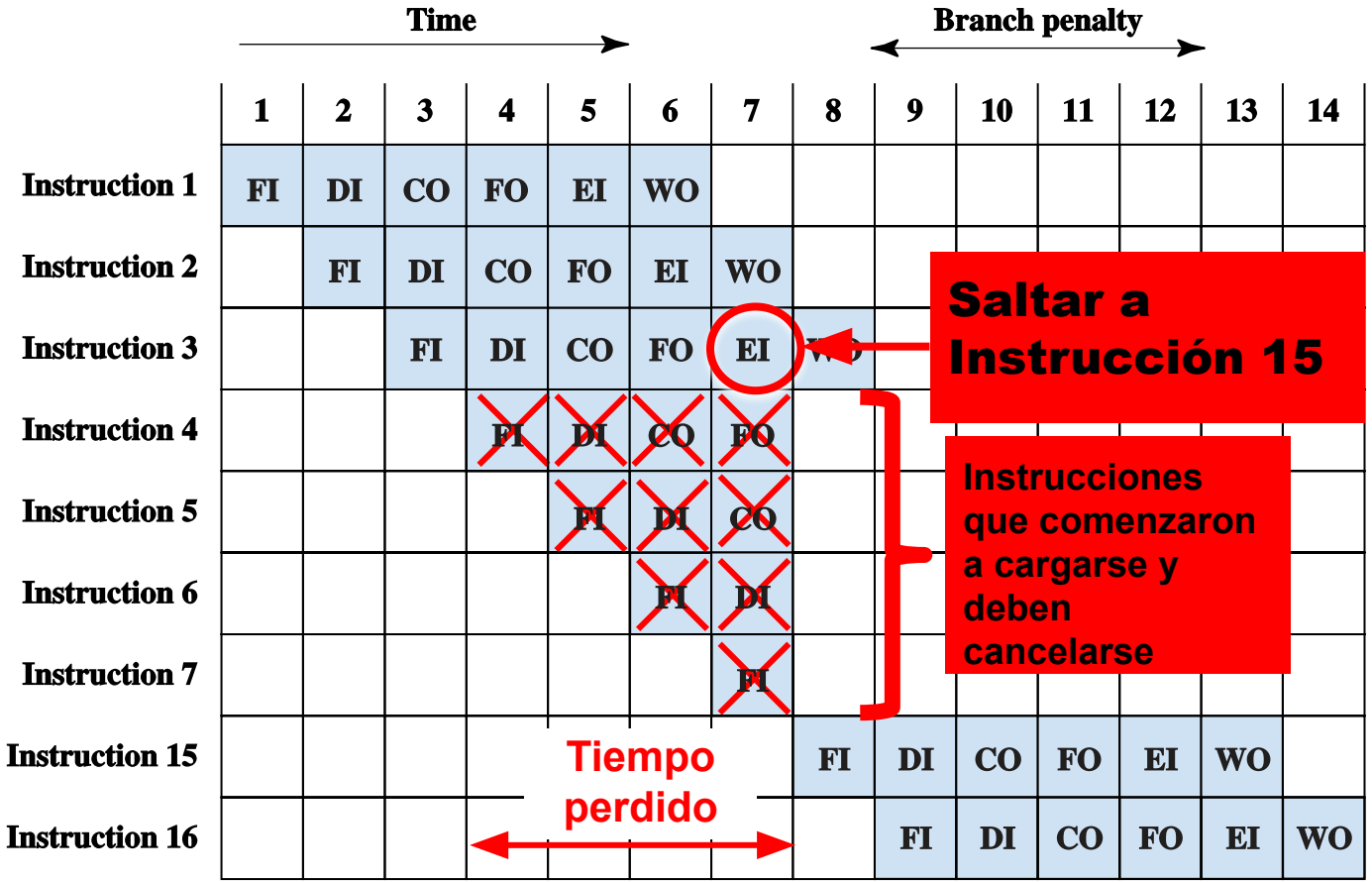
[1] "Arquitectura de computadoras", Patricia Quiroga, 1º edición, pag. 192

Saltos condicionales



Ejemplo y figura obtenidos de Computer Organization and Architecture 9th Edition William Stallings, pag 498

Problema 3: Saltos condicionales



Ejemplo y figura obtenidos de Computer Organization and Architecture 9th Edition William Stallings, pag 498

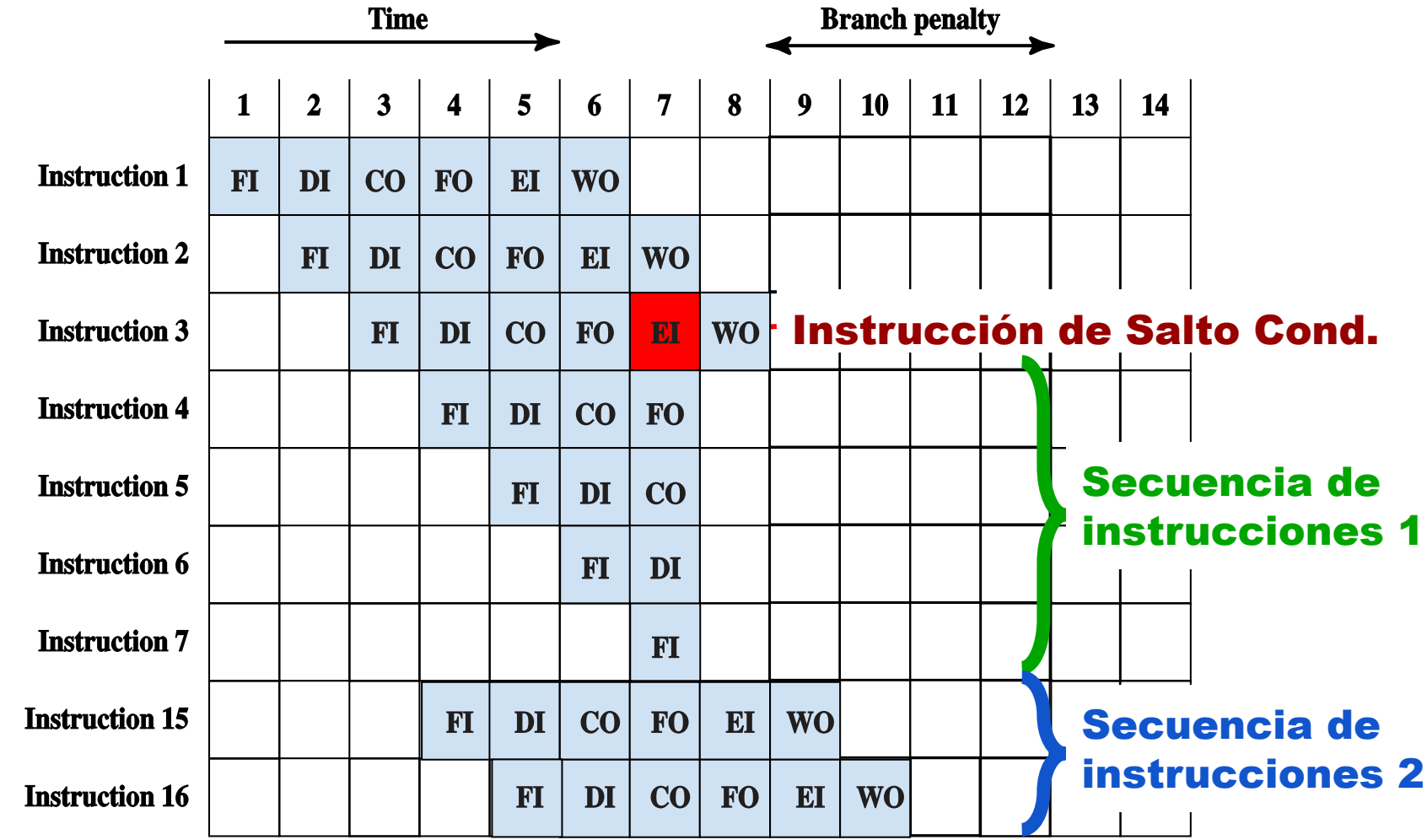
Salto condicional - Soluciones

1. **Múltiples secuencias (Multiple streams o ejecución especulativa). Ejemplos: microarquitectura Intel Bonnell (Atom), AMD k6.**
2. **Predicción de salto (Branch prediction). La mayoría de los procesadores actuales.**
3. **Salto demorado (Delayed branch). Procesadores ARM.**

Pipeline - Salto condicional - Soluciones

1) Múltiples flujos (Multiple streams) o ejecución especulativa

- **Cargar las dos posibles secuencias de instrucciones.**
- **Una vez que se conozca si se debe realizar el salto o no, se descarta la secuencia errónea y se sigue con la correcta**
- **Problemas:**
  - **Requiere duplicación (o multiplicidad) de recursos de hardware (primera parte del pipeline).**
  - **Más accesos a memoria y registros -> incremento en las demoras.**
  - **Una de las secuencias puede tener otros saltos condicionales -> necesidad de cargar más secuencias.**
  - **Muy costoso en procesadores con pipelines profundos (muchas etapas). No empleado en CISC modernos, si en algunos RISC.**
  - **Ejemplos: Intel 80486, IBM 3033, AMD k6, etc.**



Ejemplo y figura obtenidos de Computer Organization and Architecture 9th Edition William Stallings, pag 498

Pipeline - Salto condicional - Soluciones

2) Predicción de salto (Branch prediction):

- **Predicción estática:**
  - **Asumir que nunca se produce el salto (Predict never taken)**
  - **Asumir que siempre se produce el salto (Predict always taken)**
  - **Predicción basada en código de operación (Predict by opcode). Se asume que para algunos códigos siempre se producirá el salto, mientras que para otros no. Basado en datos estadísticos. (75% de aciertos)**
    - **Ejemplo: PowerPC 601 (1993).**
- **Predicción dinámica**
  - **Varios tipos.**
  - **Branch history table (empleado por procesadores x86 actuales)**

Pipeline - Salto condicional - Soluciones

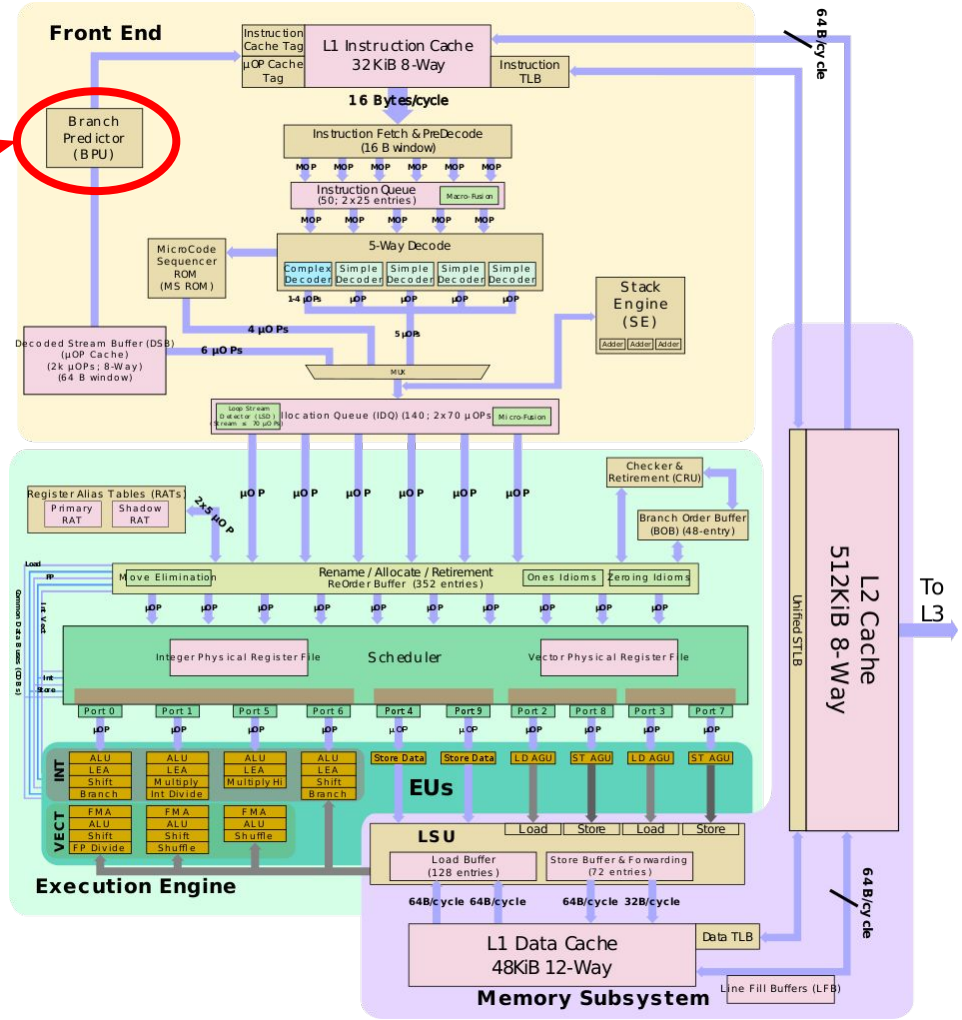
2) Predicción de salto - Branch history table o branch prediction unit

- **Tabla en una memoria de rápido acceso (caché).**

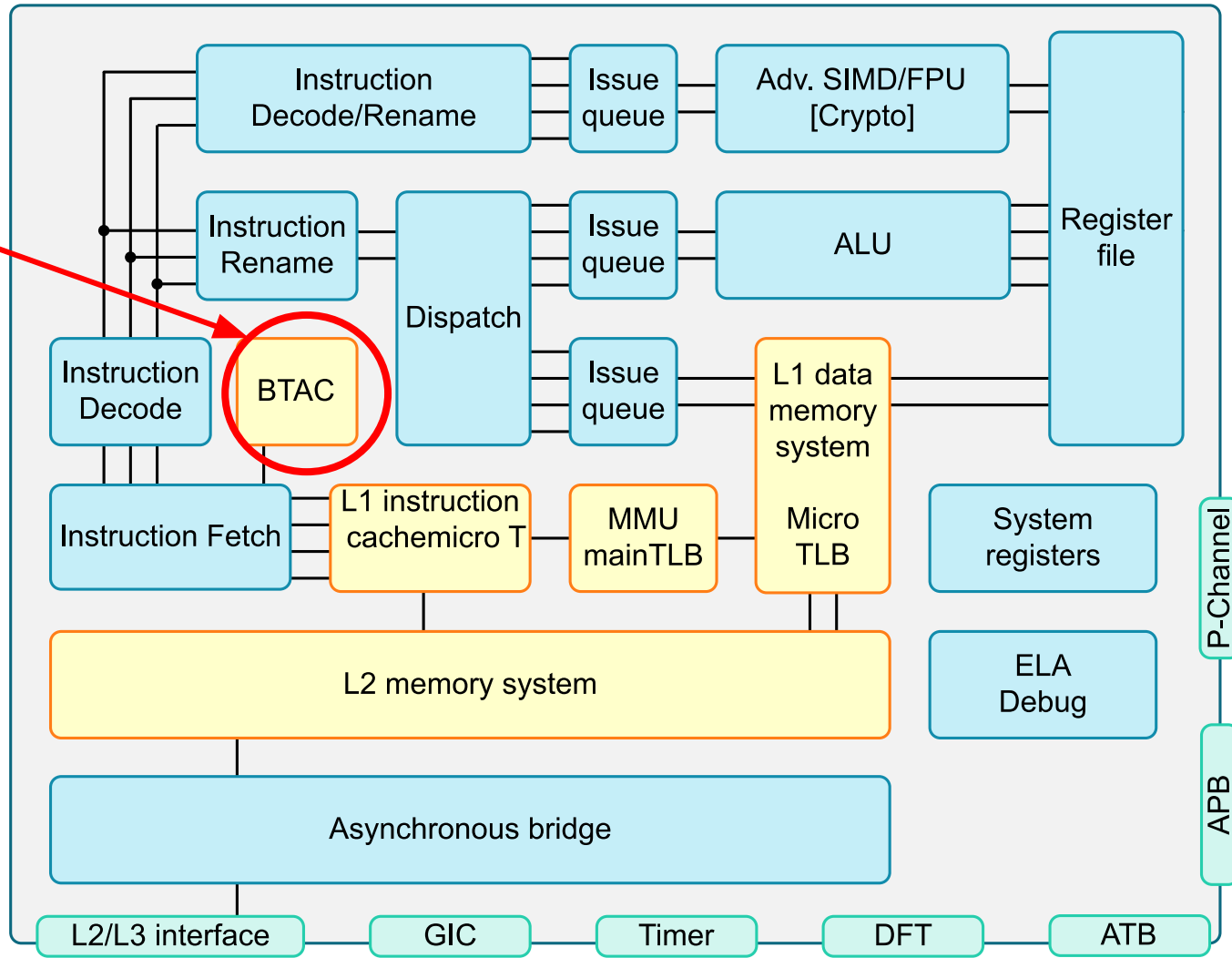
Dirección de la instrucción de salto	Dirección a la que se debe saltar si se cumple la condición	Probabilidad de salto

- **Probabilidad de salto (cuántas veces ha saltado).**
  - **Se actualiza computando aciertos y errores.**
  - **Varios algoritmos.**
- **Empleado por la mayoría de los CISC modernos.**
- **Ejemplos: Pentium 4, PowerPC 620 (1997), microarquitecturas CISC modernas (SkyLake, Nehalem, etc).**

Branch Prediction Unit



Branch Target Address Cache



Dependencias procedurales - Instrucciones de diferente longitud

- **Arquitectura de 16 bits, Memoria de 8 bits.**
- **Bus datos 16 bits (se acceden dos posiciones de memoria por ciclo de lectura o escritura).**

1)	8B1E0200	LDD BX,0x0200	3 ciclos ejecución
2)	0307	ADD AX,(BX)	3 ciclos ejecución
3)	D1E0	SHL AX	1 ciclos ejecución
4)	83C302	MUL AX,0x02	3 ciclos ejecución

1) 4 bytes
2) 2 bytes
3) 2 bytes
4) 3 bytes

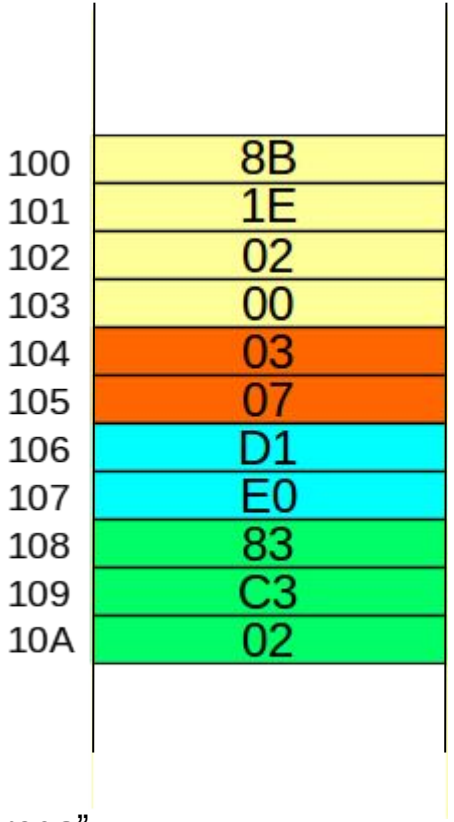


Figura y ejemplo basada en “Arquitectura de computadoras - Patricia Quiroga”

1)	8B1E0200	LDD BX,0x0200	3 ciclos ejecución
2)	0307	ADD AX,BX	3 ciclos ejecución
3)	D1E0	SHL AX	1 ciclos ejecución
4)	83C302	MUL AX,0x02	3 ciclos ejecución

8B
1E
02
00
03
07
D1
E0
83
C3
02

El procesador no puede leer los siguientes bytes hasta haber decodificado la instrucción 1, porque no sabe si los siguientes bytes son parte de la instrucción 1 (no debe escribir esos datos en el registro de instrucción), o son parte de una nueva instrucción. (debe escribirlos al registro de instrucción).

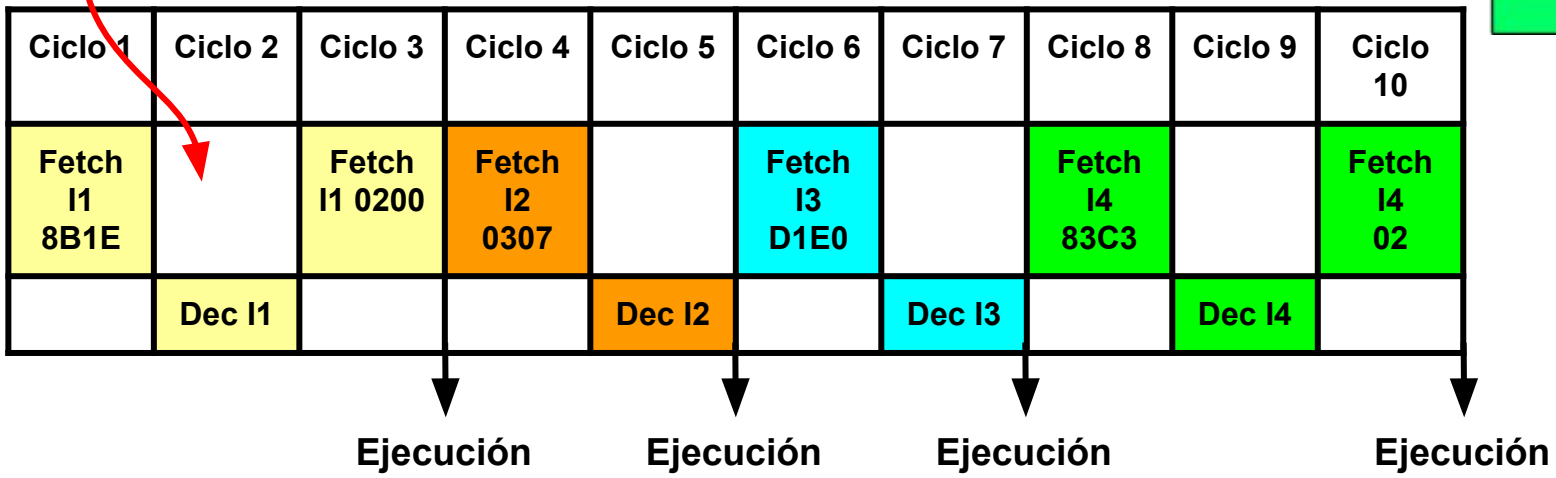


Figura basada en “Arquitectura de computadoras - Patricia Quiroga”

Dependencias procedurales - Instrucciones de diferente longitud

Especialmente grave en procesadores superescalares tipo CISC.

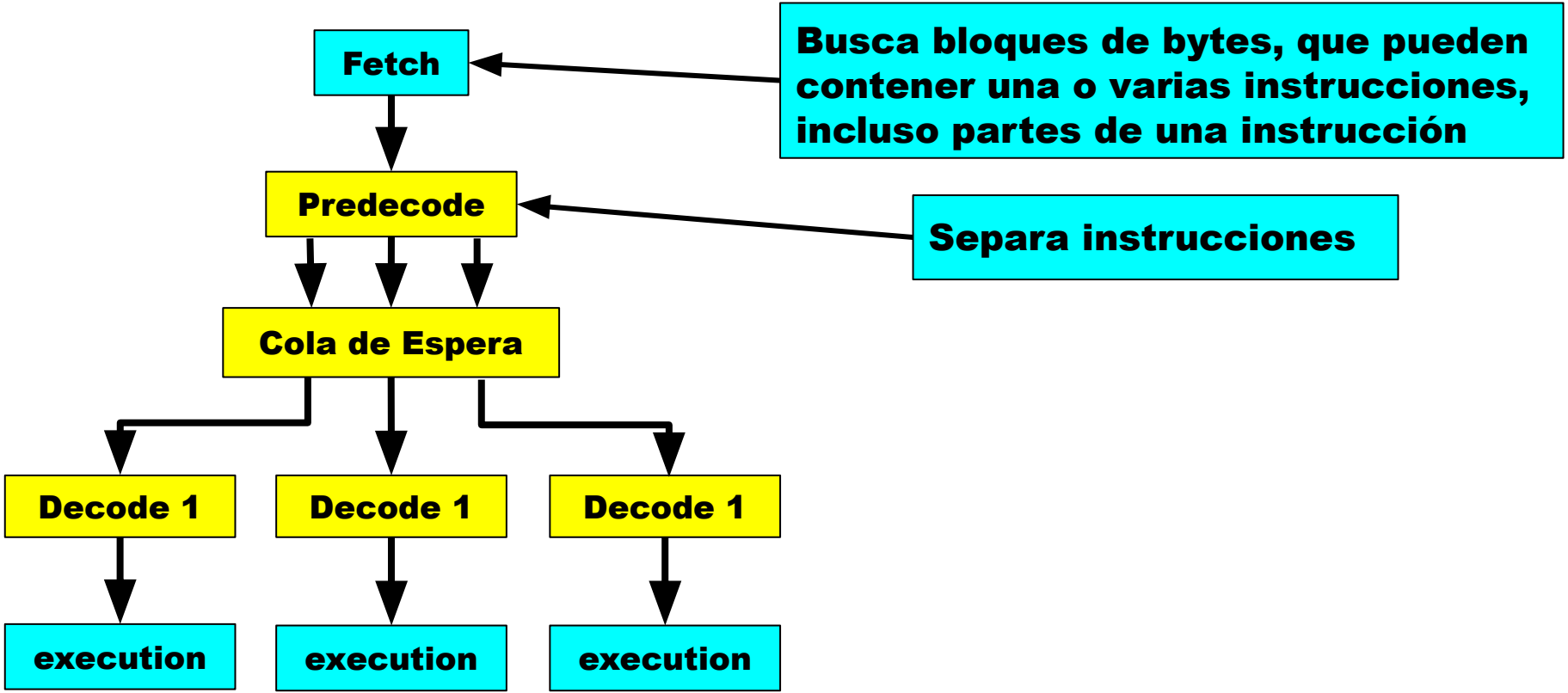
Ejemplo: x86

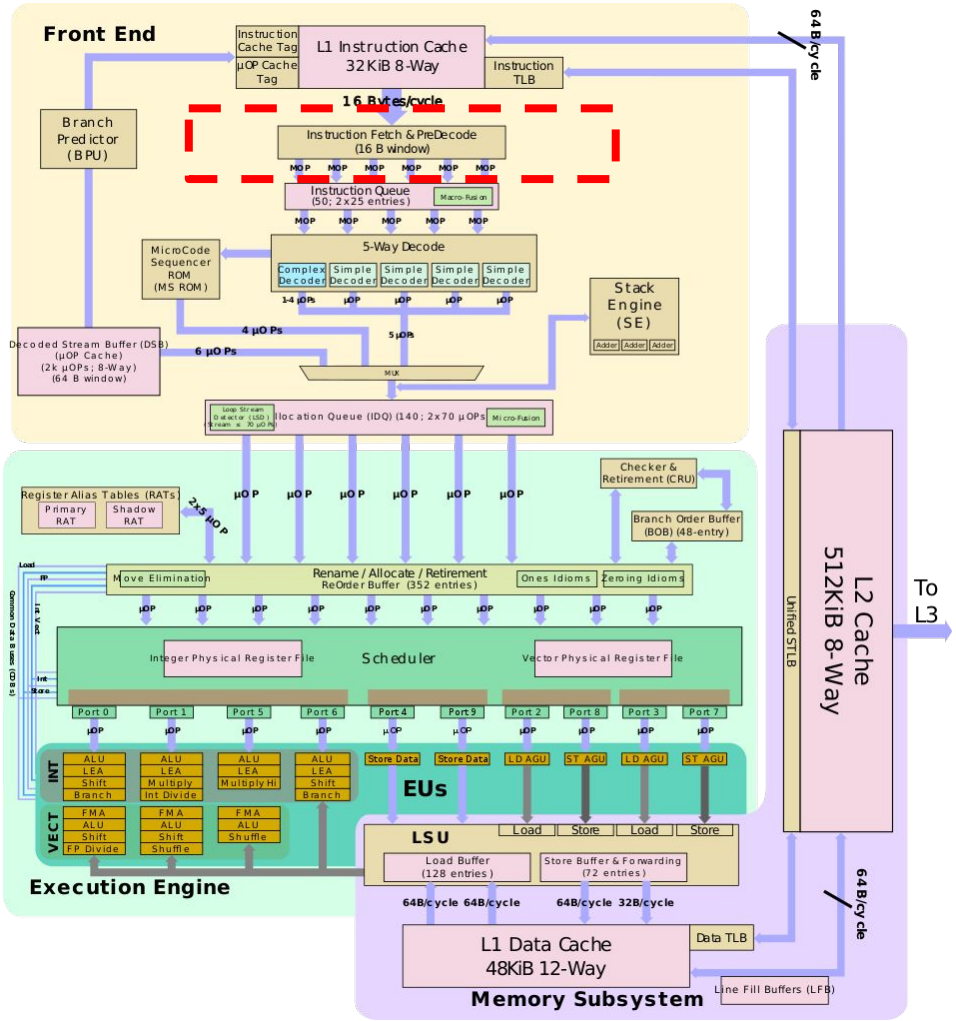
- **Instrucciones de longitud variable (1 a 17 bytes)**
- **Código de operación de longitud variable (1 a 3 bytes)**

Solución (x86):

- **Pre-decodificador + Campo de prefijo de instrucción:**
  - **Pre-decodificador: Separar instrucciones.**
  - **Campo de prefijo de instrucción: Ayuda en la pre-decodificación.**
  - **La etapa de fetch lee bloques de bytes, los cuales pueden incluir 1, 2, 3.5, etc. instrucciones.**

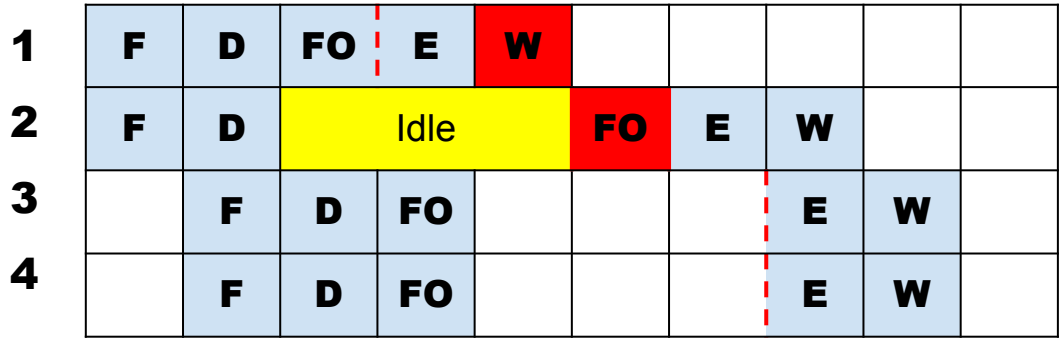
Dependencias procedurales - Instrucciones de diferente longitud

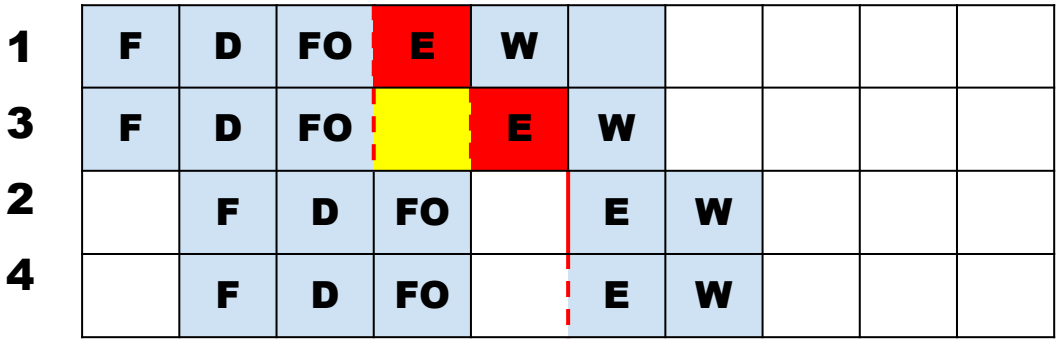



Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
64B/cycle
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP
Instruction Queue (50; 2x25 entries)
Macro-Fusion
MicroCode Sequencer ROM (MS ROM)
5-Way Decode
Complex Decoder
Simple Decoder
1-4 µOPs
µOP
Stack Engine (SE)
Adder
Decoded Stream Buffer (DSB) (µOP Cache) (2k µOPs; 8-Way) (64 B window)
4 µOPs
6 µOPs
5 µOPs
MUX
Loop Stream Detector (LSD) (Stream ≤ 70 µOPs)
Allocation Queue (IDQ) (140; 2x70 µOPs)
Micro-Fusion
Checker & Retirement (CRU)
Register Alias Tables (RATs)
Primary RAT
Shadow RAT
2-5 µOP
Branch Order Buffer (BOB) (48-entry)
Move Elimination
Rename / Allocate / Retirement
ReOrder Buffer (352 entries)
Ones Idioms
Zeroing Idioms
Integer Physical Register File
Scheduler
Vector Physical Register File
Port 0
Port 1
Port 5
Port 6
Port 4
Port 9
Port 2
Port 8
Port 3
Port 7
INT
ALU
LEA
Shift
Branch
Multiply
Int Divide
Multiply Hi
EUs
Store Data
LD AGU
ST AGU
VECT
FMA
FP Divide
Shuffle
LSU
Load
Store
Load Buffer (128 entries)
Store Buffer & Forwarding (72 entries)
64B/cycle
32B/cycle
Execution Engine
L2 Cache 512KiB 8-Way
Unified STLB
To L3
L1 Data Cache 48KiB 12-Way
Data TLB
Line Fill Buffers (LFB)
Memory Subsystem

Ejecución fuera de orden

- **Motivación: Dependencias de datos RAW y conflicto de recursos**





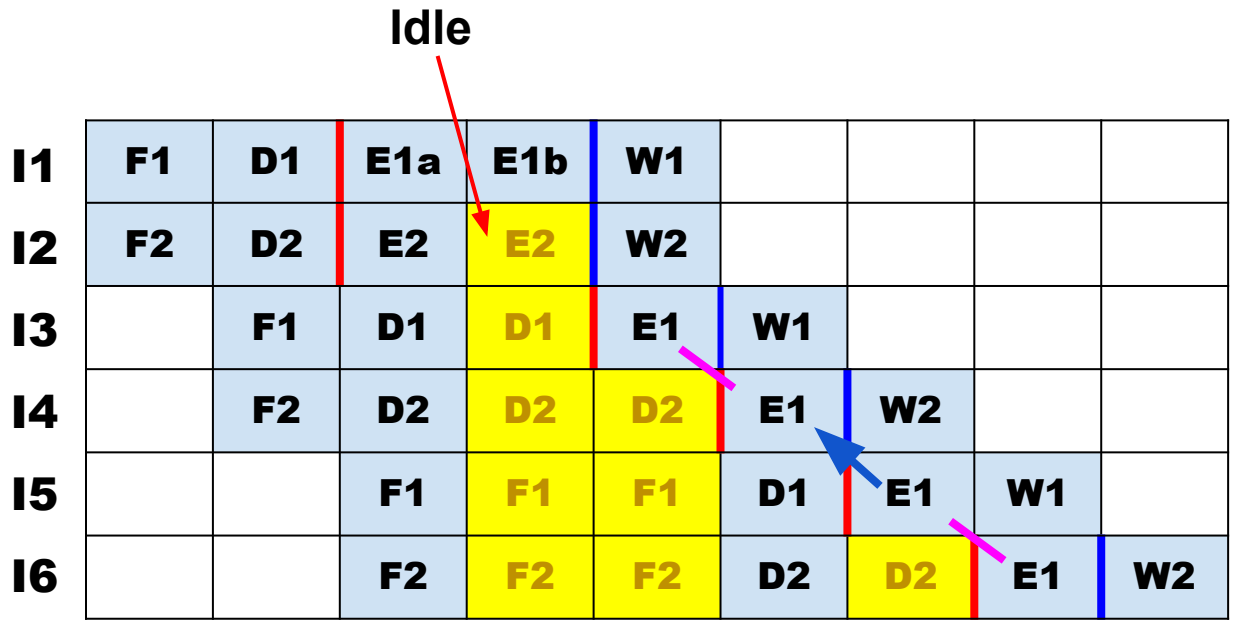
¿Si la instrucción 3 no tiene dependencia de datos con la instrucción 1, no podría ejecutarse primero la 3 y luego la 2?

Ejecución fuera de orden

- **Permite ejecutar instrucciones en un orden diferente al que fueron escritas para mitigar el problema de la dependencia de datos.**
- **Ejecución en orden: Las instrucciones: In-order issue, in-order completion**
  - **Comienzan su ejecución en orden. La instrucción x+1 puede comenzar después o al mismo tiempo que la instrucción x, pero no antes.**
  - **Terminan su ejecución en orden. La instrucción x+1 puede finalizar después o al mismo tiempo que la instrucción x, pero no antes.**
- **Tipos de ejecución fuera de orden:**
  - **In-order issue, out-of-order completion (comienzo en orden, finalización fuera de orden)**
  - **Out-of-order issue, out-of-order completion (Fuera de orden)**

Ejecución en orden

- **Las instrucciones se envían a la etapa de ejecución en el mismo orden en que fueron escritas, y finalizan en el mismo orden.**

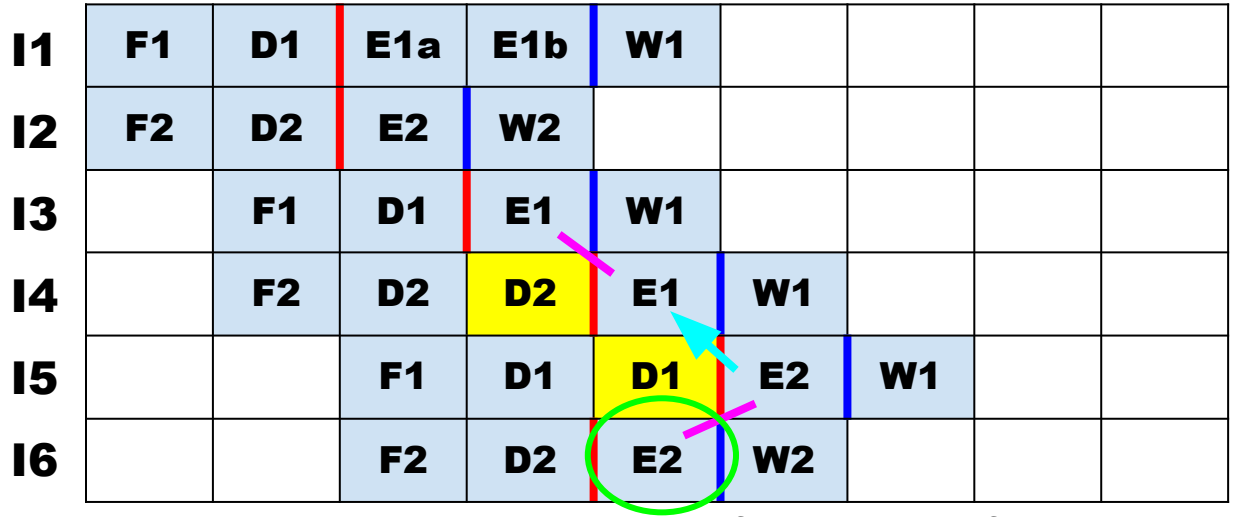


Dependencia de datos

Utilizan la misma unidad funcional

Out-of-order issue con out-of-order completion

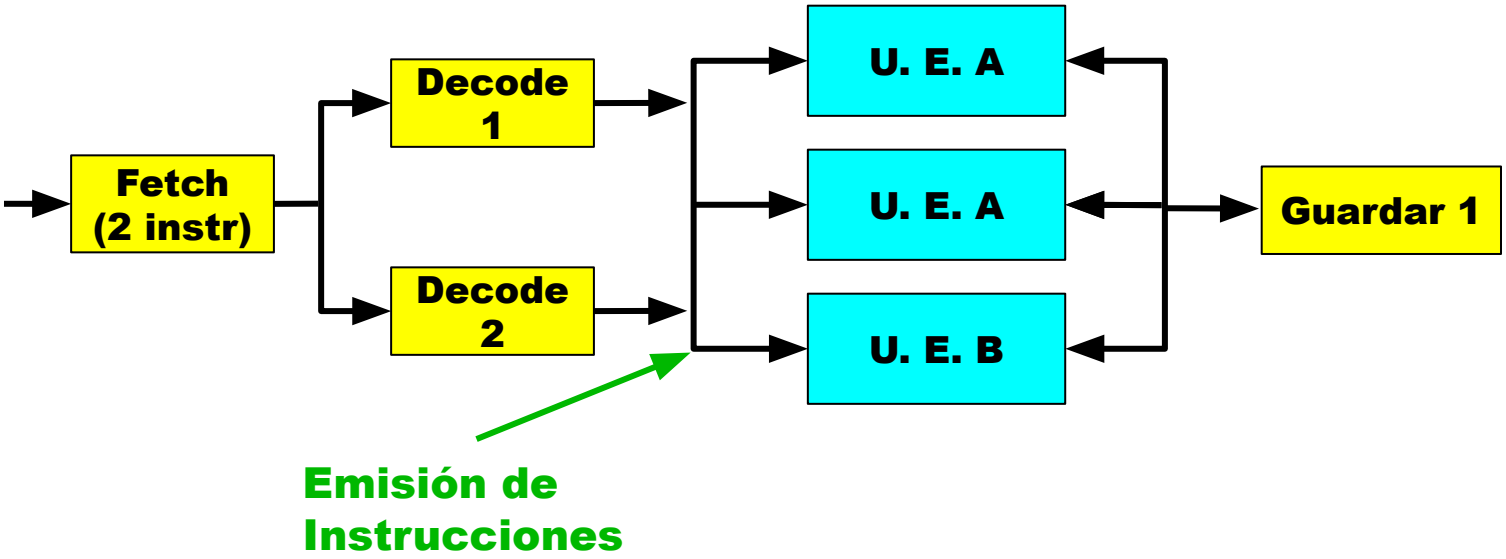
- **Para que una instrucción pase a la etapa de ejecución, se debe cumplir:**
  - **Que una unidad de ejecución como la que necesite esté libre.**
  - **Que ningún conflicto bloquee su ejecución.**



Dependencia de datos
Utilizan la misma unidad funcional

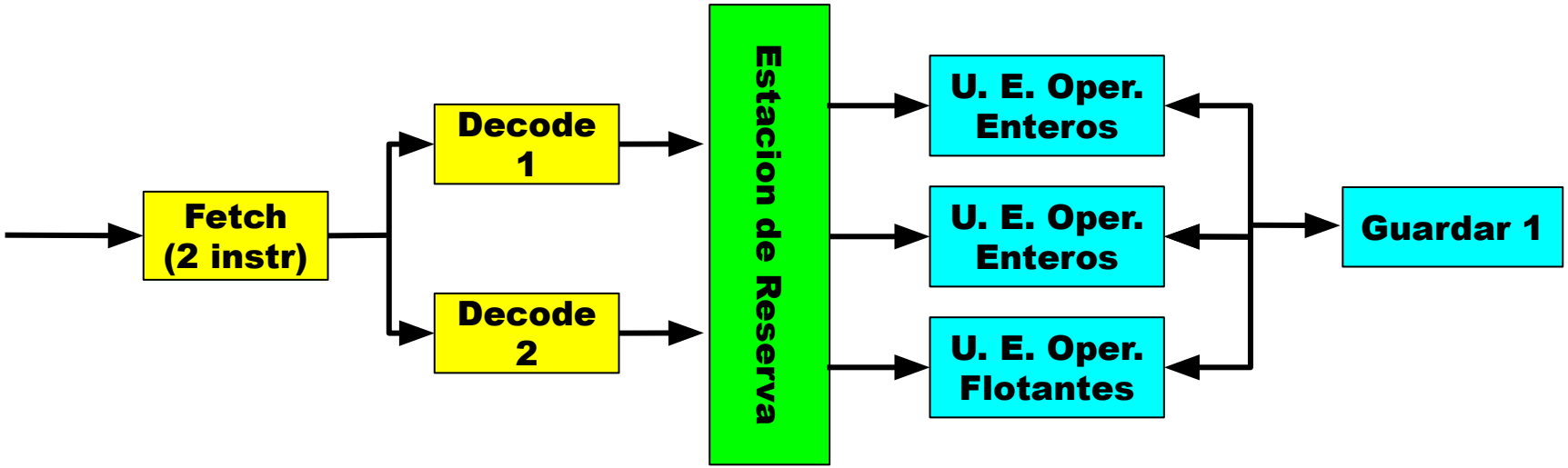
El primer procesador que implementó ejecución fuera de orden fue el Intel Pentium Pro (1995). El mismo implementó ejecución especulativa, Fue un procesador superescalar de dos vías con profundidad de pipeline de 14 estados.

Ejemplo



El pipeline anterior es imposible con el esquema de arriba, porque las instrucciones 5 y 6 no pueden pasar a decodificación mientras que la instrucción 4 no salga de decodificación.

Out-of-order issue con out-of-order completion



Nota: La estación de reserva NO es una etapa más del pipeline, las instrucciones no hacen nada mientras están en la estación de reserva. Pueden pasar directamente a las unidades de ejecución.

Out-of-order issue con out-of-order completion

(Llamado simplemente ejecución fuera de orden, o también llamada ejecución dinámica por Intel)

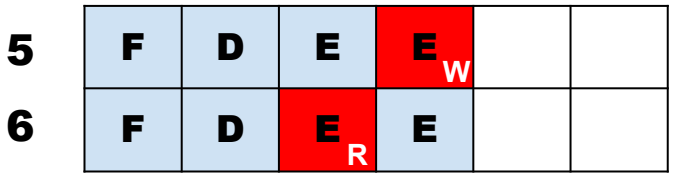
- **Las instrucciones pueden empezar su ejecución y ser finalizadas en diferente orden que el orden en que llegan al decodificador.**
  - **Nueva unidad entre el decodificador y las unidades de ejecución: planificador (scheduler), ventana de instrucciones (instruction window) o Estación de reserva.**
- **Ante una dependencia de datos o conflicto de recursos la ejecución de instrucciones no involucradas en el conflicto NO se detiene**

Procesadores fuera de orden

- Dependencia de datos tipo **RAW** o **Dependencia real**.

I5) MUL	R3,R6	(R3<-R6)
I6) ADD	R5,R3	(R5=R3+R5)
I7) ADD	R7,R3	(R7=R3+R7)

I6 necesita el dato que I5 escribe en R3, pero tomará el dato que había en R3 antes de I5



- Solución más simple: Demorar I6, pero no es la mejor solución.
- Solución 2: Procesador fuera de orden.
  - Aparecen otros problemas

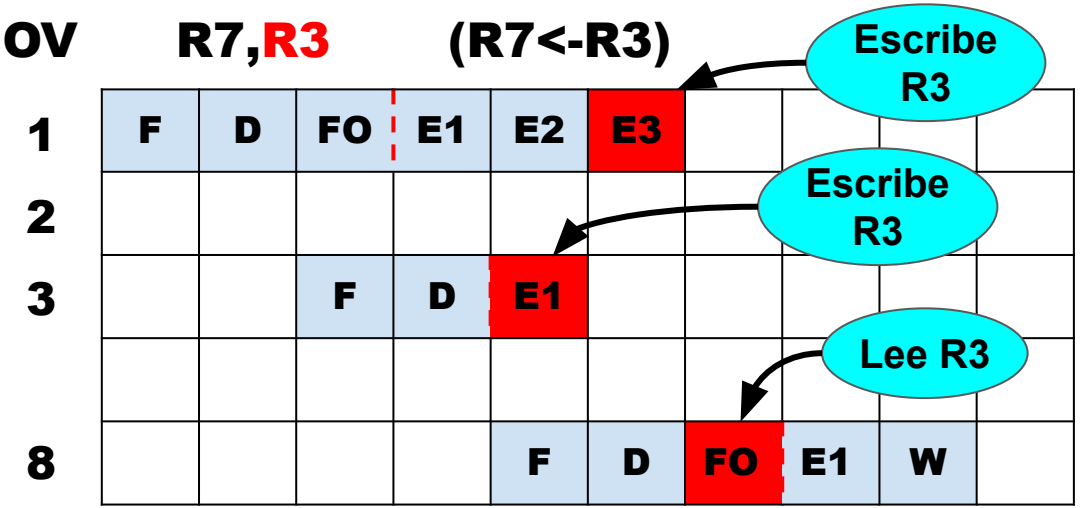
Procesadores fuera de orden

Desventajas

- Surge el problema de la dependencia de datos tipo **WAW** o **dependencia de salida**

```
I1) MUL    R3,R5      (R3=R3xR5)
I2) STR    0x200,R3
I3) CLR    R3         (R3=0)
.....      .....      .....
I8) MOV    R7,R3      (R7<-R3)
```

Si I3 finaliza antes que I1, el valor final en R3 será el que escriba I1, y no I3. I8 tomará el valor en R3 escrito por I1.

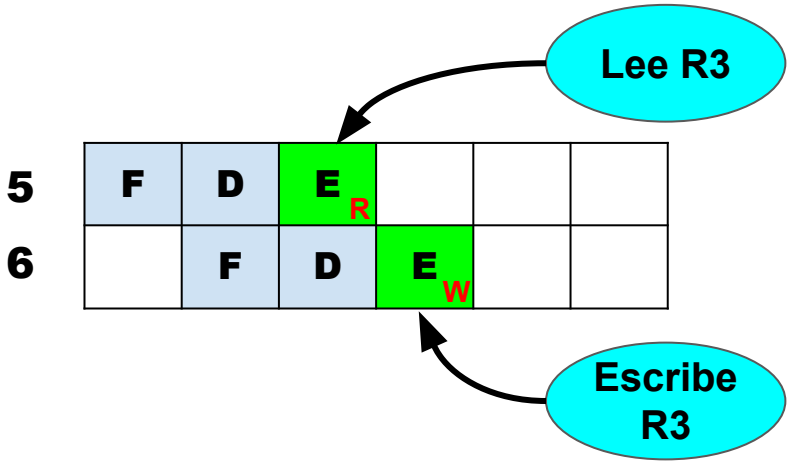


La solución más simple es demorar I3, pero no es la mejor solución.

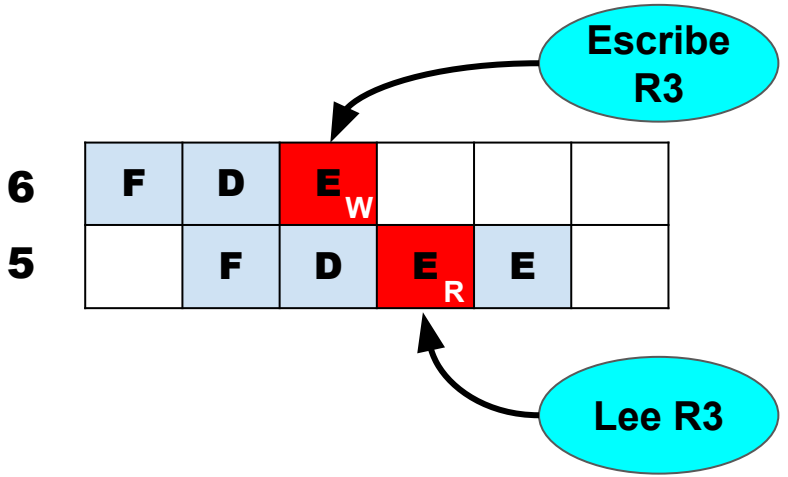
Procesadores fuera de orden

- **Dependencia de datos tipo WAR o Antidependencia.**

  I4) MUL R3,R2 (R3<-R2)
  I5) ADD R5,R3 (R5=R3+R5)
  I6) MOV R3,R6 (R3<-R6)
  I7) ADD R7,R3 (R7=R3+R7)



Ejecución **En Orden**: La I5 lee de R3 el valor escrito por I4, lo cual es **correcto**.



Ejecución **Fuera de Orden**: La I5 lee de R3 el valor escrito por I6, lo cual es **erróneo**.

Dependencia de datos - Tipos

Dependencias que afectan a procesadores en orden o fuera de orden:

- **RAW (Read After Write) o dependencia real: Se debe leer un dato después de escribirlo, pero se lee antes.**

```
ADD  r16,r17        (r16=r16+r17)
MOV r18,r16    (r18 <- r16)
```

La instrucción MOV lee R16 antes de que ADD lo escriba, leyendo un valor incorrecto

Dependencias que solo afectan a procesadores fuera de orden:

- **WAR (Write After Read) o antidependencia: Se debe escribir un dato después de leerlo, pero se escribe antes.**

```
ADD r16,r17    (r16=r16+r17)
LDI   r17,0x0F (r17=0x0F)
```

Si la instrucción LDI se ejecuta antes, y la instrucción ADD lee un valor incorrecto

- **WAW (Write After Write) o dependencia de salida: debe escribir primero primero Write1 y luego Write2, pero se realiza al revés.**

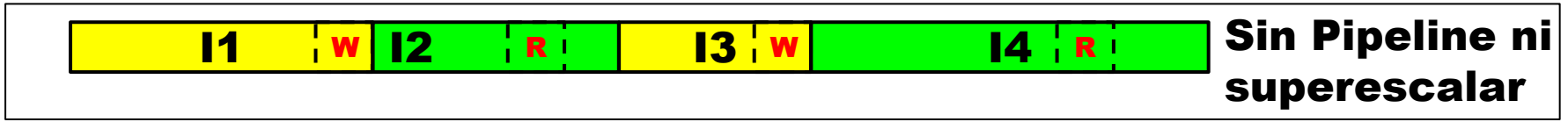
```
ADD   r16,(0x2F03)
INC    r16
```

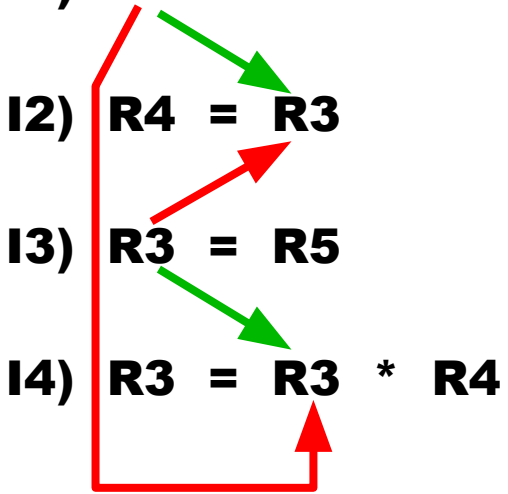
El valor final de r16 debe ser el que escriba la instrucción INC, pero si INC se ejecuta antes, en r16 quedará incorrectamente el valor que escriba ADD

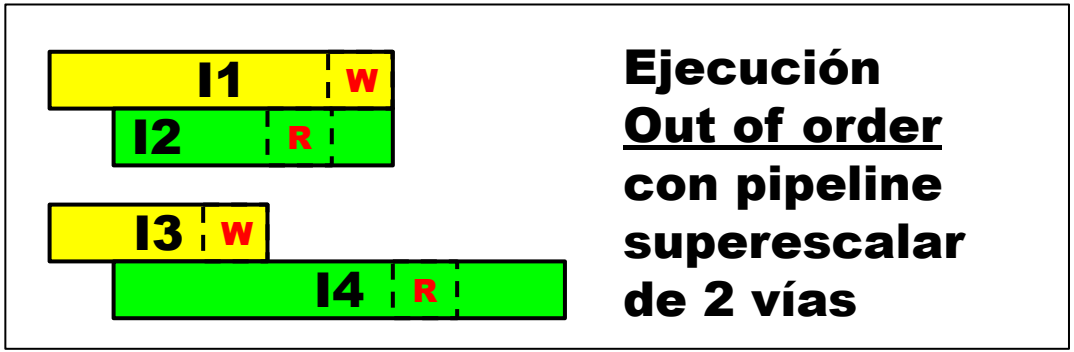
Reasignación de Registros

(Register Renaming, Register Allocation)

Solución a las dependencias de datos tipo WAW y WAR:







— Dependencias correctas (sin pipeline)

— Dependencias incorrectas (puede ocurrir con ejecución fuera de orden)

Solución: Renombrado de registros

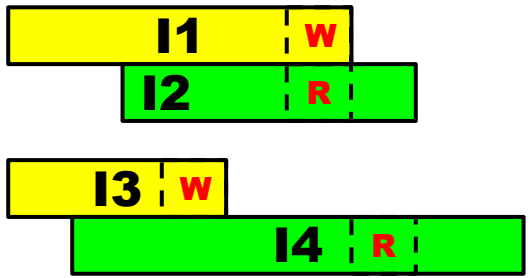
- **Problema: ¿Cómo le decimos a la I2 que utilice el valor que I1 escribe en R3, y no el valor que escribe I3 por ejecutarse antes?**
- **Solución: Utilizar subíndices, esto se llama Renombrado de registros.**
- **Se debe incrementar el número de registros (ya no existe un solo R3, hay un R3a, un R3b, un R3c, etc.)**

I1) R3b = R3a + R5a

I2) R4a = R3b

I3) R3c = R5a

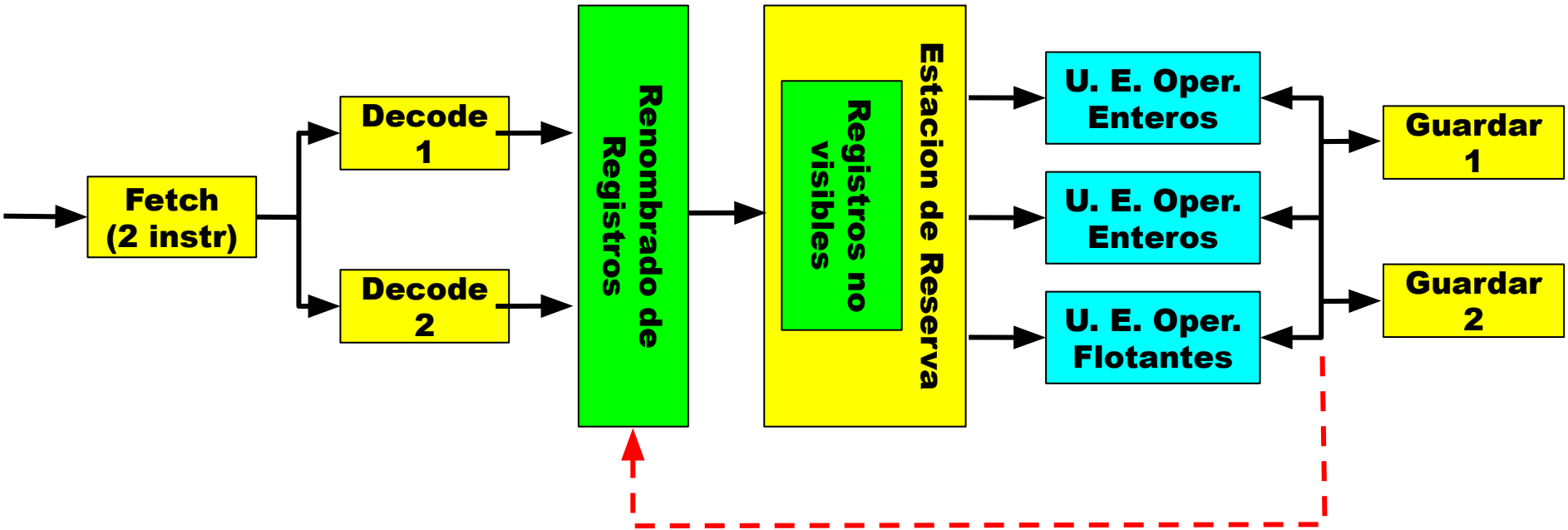
I4) R3c = R3c * R4b



Solución: Renombrado de registros

- **Las instrucciones se refieren a registros direccionables o visibles al programador (por ejemplo: R3)**
- **El renombrado de registros es un procedimiento interno del procesador, transparente al usuario, que asocia dinámicamente registros no visibles a los registros direccionables (por ejemplo: asocia a R3 los registros no visibles R3a, R3b, R3c, R3d, etc).**
- **Funcionamiento:**
  - **Cada instrucción que escribe en un registro utiliza un nuevo subíndice.**
  - **Cada instrucción que lee un registro lee el registro con el último subíndice asignado por una instrucción anterior.**
- **Aparece un nuevo bloque denominado Renombrado de registros (Register rename, Rename, Allocation, Issue queue, etc.)**

Solución: Renombrado de registros

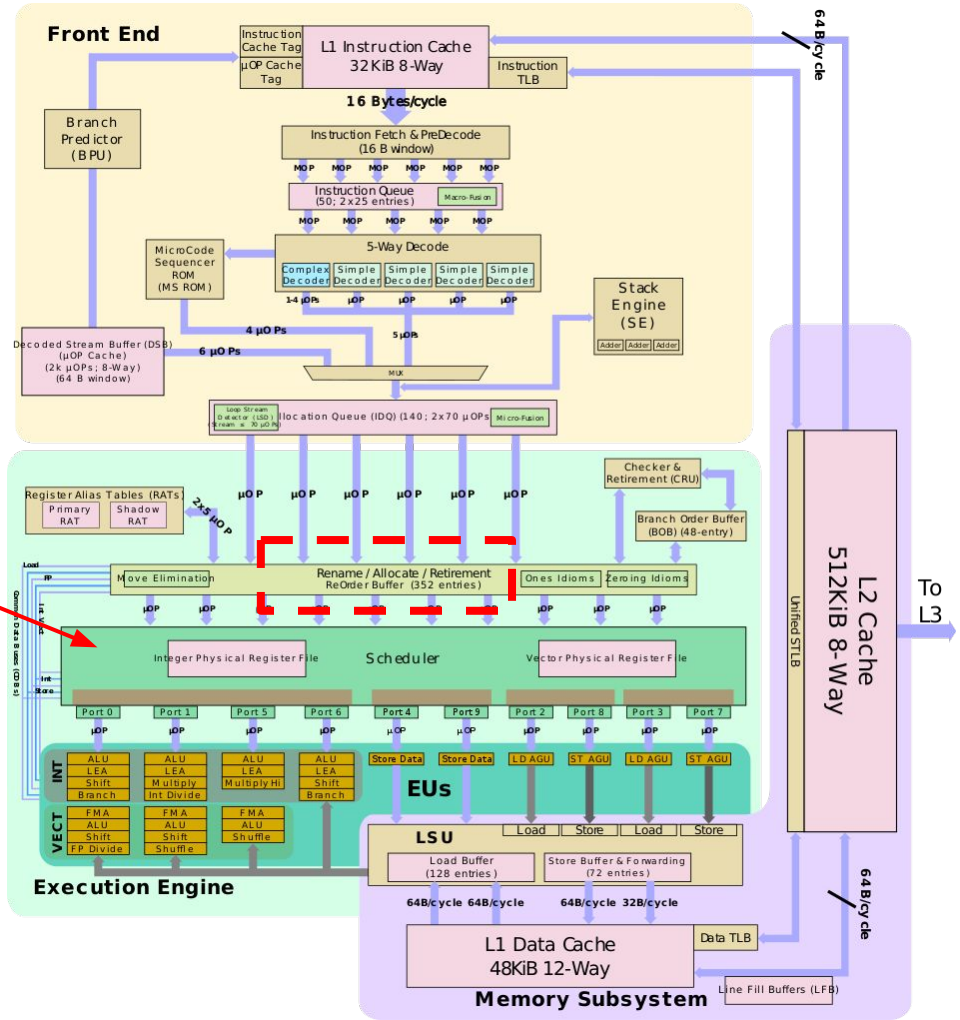


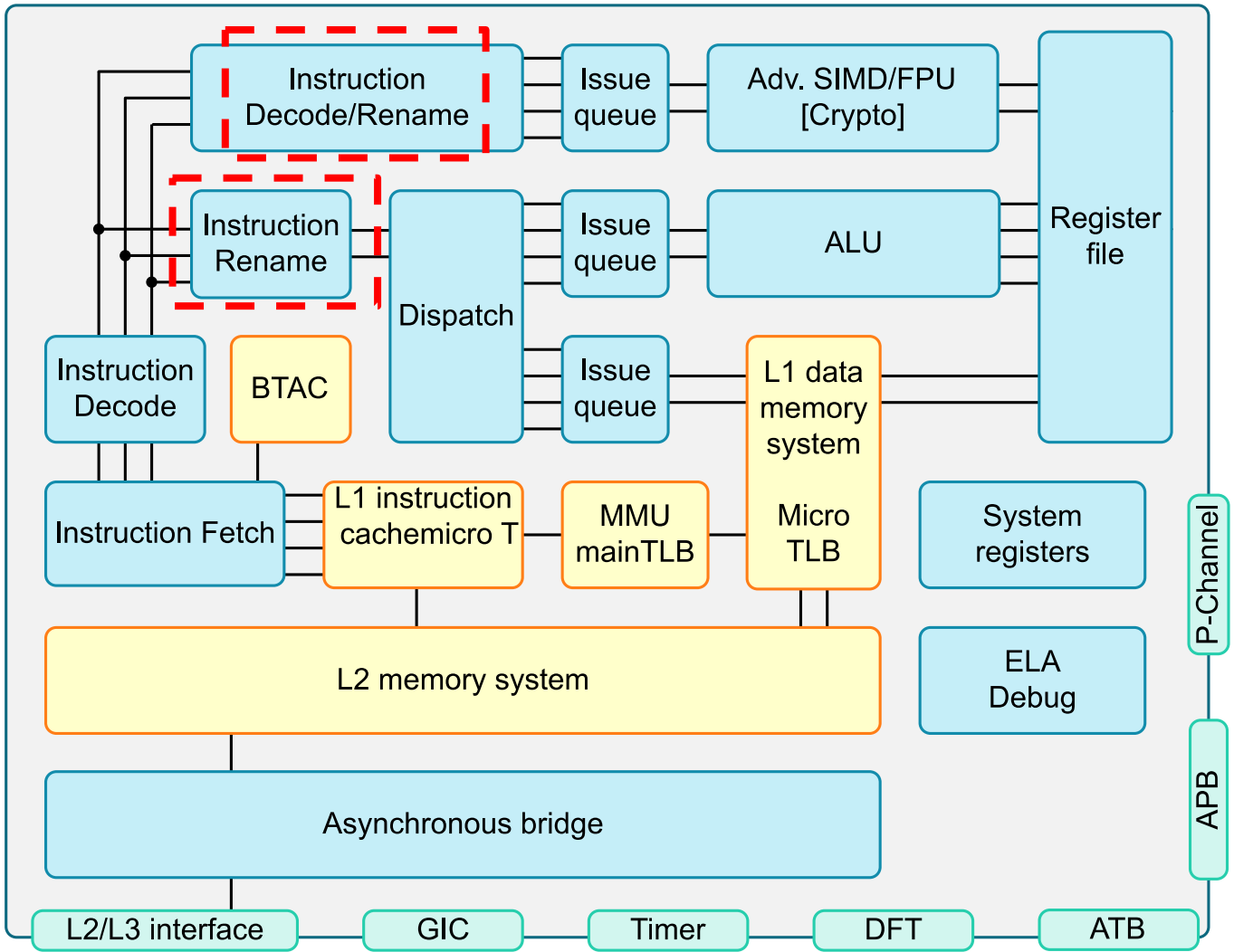
Renombrado de registros

Comentarios adicionales:

- **Soluciona las dependencias de datos tipo WAW (dependencia de salida) y WAR (antidependencia) en procesadores con ejecución fuera de orden con pipeline y/o superescalares.**
- **La dependencia de datos RAW (dependencia Real) No se soluciona, se mitiga su efecto con los procesadores fuera de orden.**
- **Usado en las microarquitecturas actuales de procesadores CISC y RISC (Intel, ARM, AMD, etc) desde 2008 hasta la fecha (2022).**
- **Ejemplos:**
  - **Pentium IV: 16 registros "direccionables" que se reasignan en 128 registros físicos.**
  - **Microarquitectura Kaby Lake: 16 registros de 64 bits, 180 registros para enteros y 168 para flotantes.**

Estación de reserva




Instruction Decode/Rename
Issue queue
Adv. SIMD/FPU [Crypto]
Register file
Instruction Rename
Dispatch
Issue queue
ALU
Instruction Decode
BTAC
Issue queue
L1 data memory system
Micro TLB
Instruction Fetch
L1 instruction cachemicro T
MMU mainTLB
System registers
P-Channel
L2 memory system
ELA Debug
Asynchronous bridge
APB
L2/L3 interface
GIC
Timer
DFT
ATB

Arquitectura del conjunto de instrucciones
ISA: Instruction Set Architecture

- **Define el conjunto de características visibles al programador:**
  - **Conjunto de instrucciones**
  - **Tipos de datos (enteros, flotantes, caracteres, etc.)**
  - **Registros visibles**
  - **Arquitectura de memoria y modos de direccionamiento**
  - **Interrupciones**
  - **Operaciones matemáticas**
  - **Formato de instrucciones: longitud, número de direcciones, tamaño de los campos, etc.**
  - **Todo aquello que puede realizarse o direccionar mediante las instrucciones del conjunto de instrucciones.**

Arquitectura del conjunto de instrucciones

- **Diferentes microarquitecturas pueden implementar el mismo conjunto de instrucciones (ejemplo x86 es implementado por un gran número de microarquitecturas de diferentes fabricantes como Intel, AMD, Cyrix, etc).**
- **Interfaz entre el desarrollador de un procesador y el programador.**
- **Afecta:**
  - **La eficiencia del procesador (Dependencias, etc).**

longitud de instrucción

- **Longitud de instrucción = múltiplo o submúltiplo del tamaño del bus de datos.**
  - **Longitud instrucción = tamaño del bus -> Se lee una instrucción por ciclo de lectura.**
  - **Longitud instrucción < tamaño del bus -> Se leen más de una instrucción por ciclo de lectura.**
  - **Longitud instrucción > tamaño del bus -> Se requieren n ciclos de lectura por instrucción.**

- **Mayor longitud -> mayor cantidad de instrucciones, más modos de direccionamiento -> mejor para el programador**

Instrucciones típicas de un set de instrucciones 1

Type	Operation Name	Description
Data transfer	Move (transfer)	Transfer word or block from source to destination
	Store	Transfer word from processor to memory
	Load (fetch)	Transfer word from memory to processor
	Exchange	Swap contents of source and destination
	Clear (reset)	Transfer word of 0s to destination
	Set	Transfer word of 1s to destination
	Push	Transfer word from source to top of stack
	Pop	Transfer word from top of stack to destination
Logical	AND	Perform logical AND
	OR	Perform logical OR
	NOT (complement)	Perform logical NOT
	Exclusive-OR	Perform logical XOR
	Test	Test specified condition; set flag(s) based on outcome
	Compare	Make logical or arithmetic comparison of two or more operands; set flag(s) based on outcome
	Set Control Variables	Class of instructions to set controls for protection purposes, interrupt handling, timer control, etc.
	Shift	Left (right) shift operand, introducing constants at end
	Rotate	Left (right) shift operand, with wraparound end

Tabla obtenida de Computer Organization and Architecture 9th Edition William Stallings, pag 419 y 420

Instrucciones típicas de un set de instrucciones 2

Type	Operation Name	Description
Arithmetic	Add	Compute sum of two operands
	Subtract	Compute difference of two operands
	Multiply	Compute product of two operands
	Divide	Compute quotient of two operands
	Absolute	Replace operand by its absolute value
	Negate	Change sign of operand
	Increment	Add 1 to operand
	Decrement	Subtract 1 from operand
Transfer of control	Jump (branch)	Unconditional transfer; load PC with specified address
	Jump Conditional	Test specified condition; either load PC with specified address or do nothing, based on condition
	Jump to Subroutine	Place current program control information in known location; jump to specified address
	Return	Replace contents of PC and other register from known location
	Execute	Fetch operand from specified location and execute as instruction; do not modify PC
	Skip	Increment PC to skip next instruction
	Skip Conditional	Test specified condition; either skip or do nothing based on condition
	Halt	Stop program execution
	Wait (hold)	Stop program execution; test specified condition repeatedly;
		resume execution when condition is satisfied
	No operation	No operation is performed, but program execution is continued

Tabla obtenida de Computer Organization and Architecture 9th Edition William Stallings, pag 419 y 420

Instrucciones típicas de un set de instrucciones 3

,

Type	Operation Name	Description
Input/output	Input (read)	Transfer data from specified I/O port or device to destination (e.g., main memory or processor register)
	Output (write)	Transfer data from specified source to I/O port or device
	Start I/O	Transfer instructions to I/O processor to initiate I/O operation
	Test I/O	Transfer status information from I/O system to specified destination
Conversion	Translate	Translate values in a section of memory based on a table of correspondences
	Convert	Convert the contents of a word from one form to another (e.g., packed decimal to binary)

Tabla obtenida de Computer Organization and Architecture 9th Edition William Stallings, pag 419 y 420

x86

- **Intel. 1976. Nace con el 8086 (16 bits).**
- **Gran éxito: inclusión del Intel 8088 (16 bits) en la IBM PC.**
- **Evolución:**
  - **Primera versión: 16 bits**
  - **32 bits o IA-32**
  - **64 bits (creada por AMD), luego adoptada por Intel (x86-64, AMD64; Intel 64; x64).**
- **Implementado por: Intel, AMD, Cyrix, Via, etc. mediante distintas microarquitecturas.**
- **Última implementación tiene más de 1300 instrucciones.**
- **Instrucciones especiales para entrada salida.**
- **Instrucciones para trabajar con cadena de caracteres (MOVS, LODS). Útil editor de textos.**

x86

- Instrucciones aritméticas y lógicas que operan sobre registros y posiciones de memoria (arquitectura register-memory).
- Instrucciones SIMD (PADDB: add packed byte integers)
- Gran cantidad de extensiones. En promedio: 1 instrucción por mes.
- Compatibilidad “hacia atrás” (solo se agregan instrucciones, no se retiran).
- Longitud de datos: 8, 16, 32, 64 y 128 bits.
- Datos no alineados.
- Memoria de 8 bits.
- Tipos de datos:
  - General (cadena de bits con contenido arbitrario)
  - Enteros con signo y sin signo.
  - BCD
  - Números en punto flotante.

Formato de Instrucción IA-32 (x86)

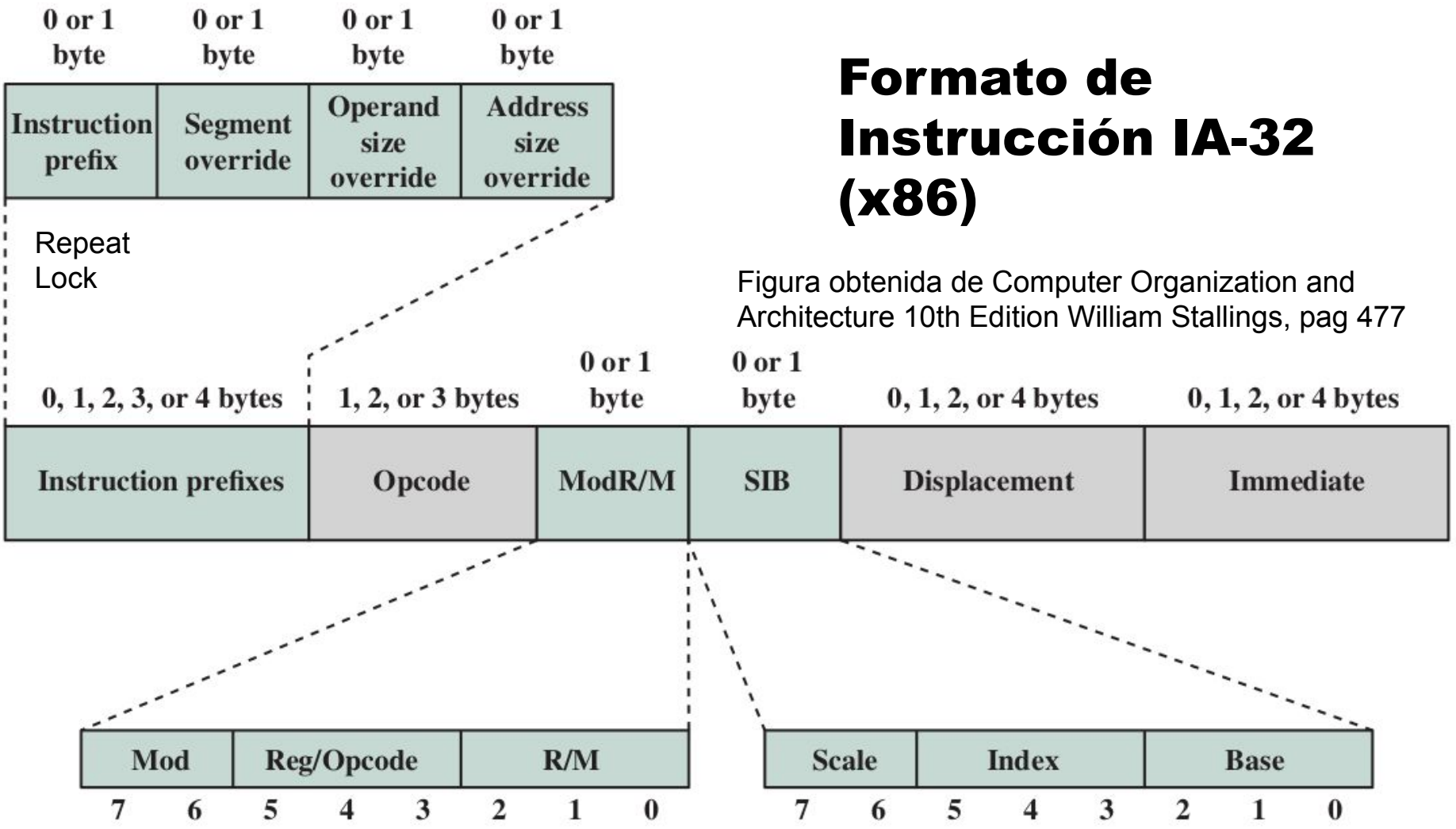


Figura obtenida de Computer Organization and Architecture 10th Edition William Stallings, pag 477

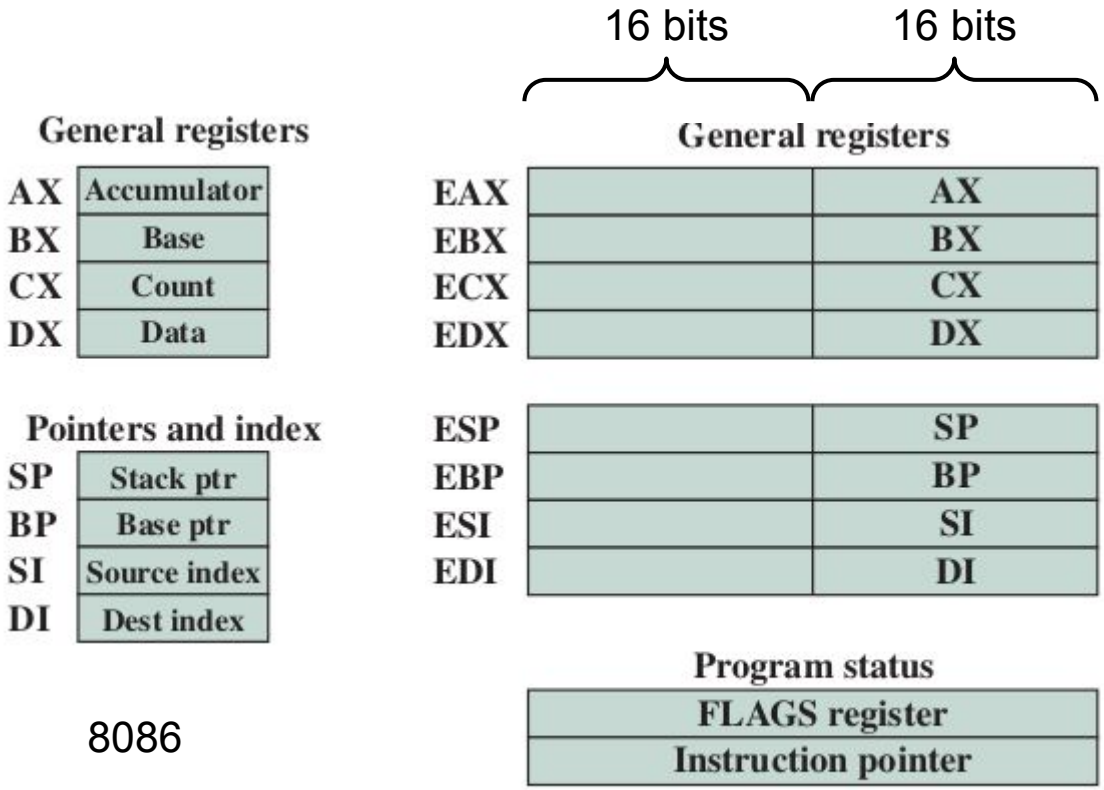
Formato de Instrucción x86 (explicación)
Prefijos de Instrucción

- **Prefijo de Instrucción: Diferentes opciones.**
  - **LOCK: bloqueo de memoria.**
  - **Repeat: Operaciones repetidas sobre un string.**
    - **Gran utilidad en editores de texto.**
- **Segment override: Si está presente, indica el registro de segmento a utilizar.**
- **Operand size: Selecciona el tamaño del operando (32 o 16 bits)**
- **Address size: Selecciona el tamaño de las direcciones (direccionamiento por desplazamiento) (32 o 16 bits)**
- **Opcode: Código de operación + opciones.**
- **Mod R/M: Información sobre direccionamiento. El significado de sus bits dependen de Opcode.**

Formato de Instrucción x86 (explicación)

- **SIB: Información direccionamiento indexado**
  - **Scale: Escalado usado en direccionamiento escalado indexado.**
  - **Index: Registro índice**
  - **Base: Registro base**
- **Displacement: Desplazamiento utilizado. Puede ser un valor con signo de 8, 16 o 32 bits.**
- **Immediate: Operando inmediato.**

Registros y compatibilidad hacia atrás



ARM

- **1983 Acorn. ARM (Acorn RISC Machine)**
- **Acorn + VLSI Technology + Apple -> ARM Holdings (Advanced RISC Machine).**
- **Instrucciones aritméticas y lógicas que operan sobre registros (arquitectura register-register).**
- **Accesos a memoria solo a través de instrucciones Load y Store.**
- **Instrucciones SIMD.**
- **Acceso a periféricos mediante mapeo en memoria.**
- **32 bits (ARMv7 y anteriores); 64 bits (ARMv8). Permite instrucciones de 16 bits (Extensiones Thumb y Thumb-2). La primera versión fue de 32 bits.**
- **Instrucciones de tamaño fijo.**
- **Varias extensiones.**

VLSI Technology: Empresa de semiconductores, comprada por Philips Electronics, actualmente NXP Semiconductors.

Modos de direccionamiento ARM

- **Un RISC pocos y simples modos de direccionamiento… menos ARM**
- **Instrucciones que acceden a memoria: Solo instrucciones de tipo Load y Store mediante:**
  - **Directo e inmediato**
  - **Offset: El offset se suma o resta a una dirección almacenada en un registro base.**
    - **Offset en el campo de dirección**
    - **Offset en un registro**
    - **Offset con escalamiento**
  - **Pre-indexado**
  - **Post-indexado**
  - **Carga o escritura de múltiples datos en varios (o todos) los registros.**
  - **Relativo al contador de programa.**

ARM

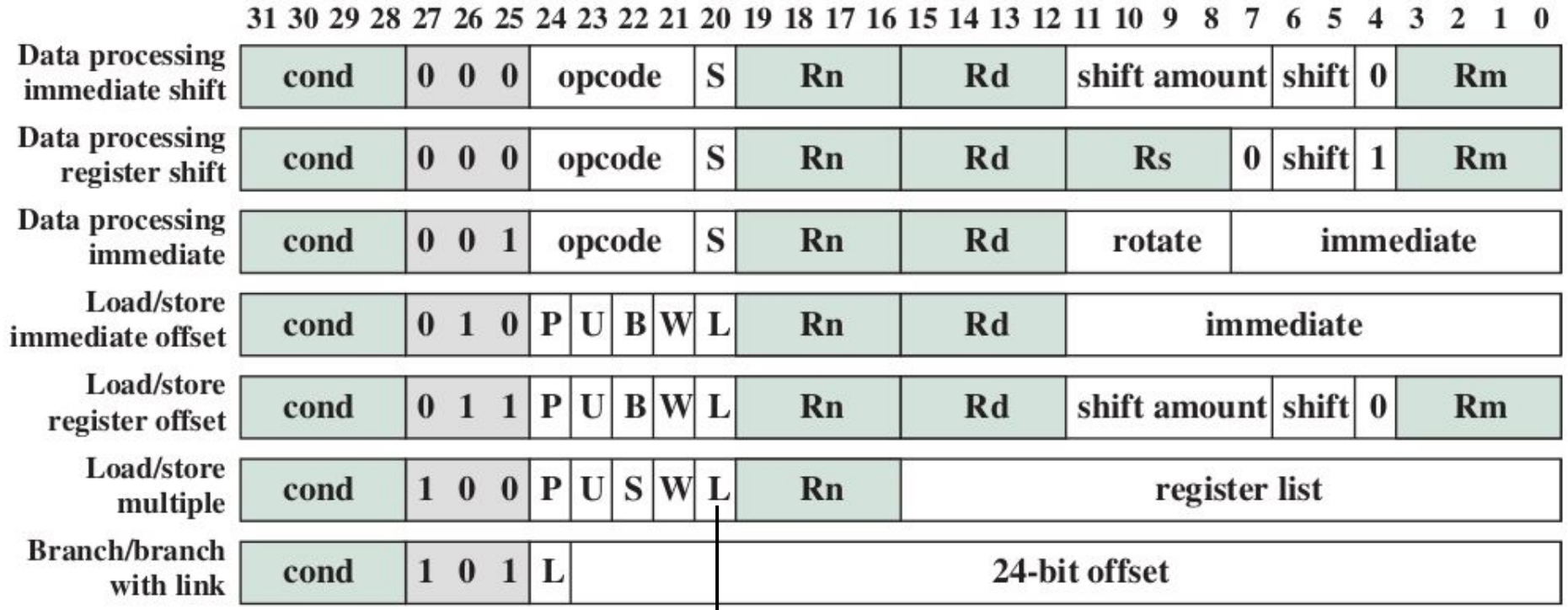


Figura obtenida de Computer Organization and Architecture 10th Edition William Stallings, pag 480

Formato de Instrucción ARM (explicación)

- Ejecución condicional (bits 28 a 31): La instrucción se ejecuta si se cumple la condición.
  - Se verifican las 4 banderas del registro de estado (N=Negative, Z=Zero, C=Carry y V=oVerflow). Ejemplos:
    - Cond=0000 -> Ejecutar si Z=1
    - Cond=0001 -> Ejecutar si Z=0
    - Cond=1000 -> Ejecutar si Z=0 y C=1
    - Cond=1110 y Cond=1111 -> Ejecutar siempre
- S = Instrucciones de procesamiento de datos, la instrucción actualiza los códigos de condición.
- S = Instrucciones load/store múltiples, indica si las instrucciones deben ejecutarse en modo supervisor.
- P, U, W = Diferentes tipos de direccionamiento.
- B = unsigned byte o word.

MLA

Multiply Accumulate multiplies two register values, and adds a third register value. The least significant 32 bits of the result are written to the destination register. These 32 bits do not depend on whether the source register values are considered to be signed values or unsigned values.

In an ARM instruction, the condition flags can optionally be updated based on the result. Use of this option adversely affects performance on many processor implementations.

Encoding T1 ARMv6T2, ARMv7

```
MLA<c> <Rd>, <Rn>, <Rm>, <Ra>
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	0	0	0	0	Rn				Ra				Rd				0	0	0	0	Rm			

```
if Ra == '1111' then SEE MUL;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  a = UInt(Ra);  setflags = FALSE;
if d IN {13,15} || n IN {13,15} || m IN {13,15} || a == 13 then UNPREDICTABLE;
```

Encoding A1 ARMv4*, ARMv5T*, ARMv6*, ARMv7

```
MLA{S}<c> <Rd>, <Rn>, <Rm>, <Ra>
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				0	0	0	0	0	0	1	S	Rd				Ra				Rm				1	0	0	1	Rn			

For the case when cond is 0b1111, see *Unconditional instructions* on page A5-214.

```
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  a = UInt(Ra);  setflags = (S == '1');
if d == 15 || n == 15 || m == 15 || a == 15 then UNPREDICTABLE;
if ArchVersion() < 6 && d == n then UNPREDICTABLE;
```

Ejemplo obtenido de "ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition" página 481.

Otras arquitecturas de set de instrucciones

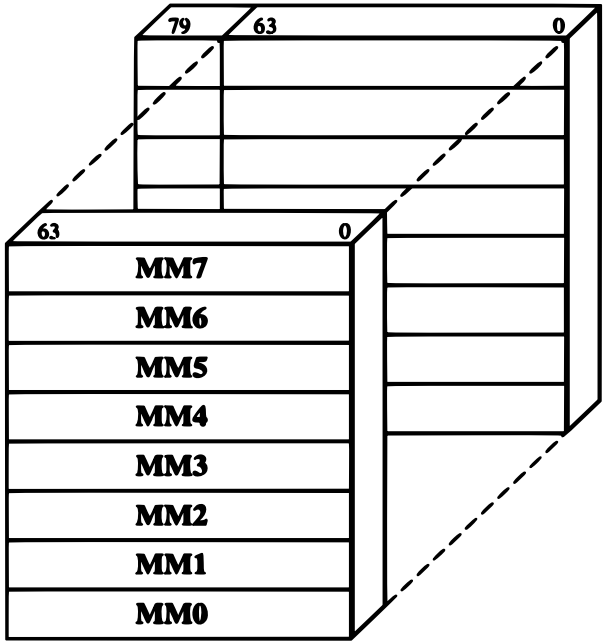
- **IA-64 (También llamada arquitectura Itanium): Intel y HP.**
  - **Procesadores Itanium (servidores y sistemas HPC)**
  - **Diseñado para lograr un alto grado de paralelismo**
  - **EPIC (explicit instruction-level parallelism). El compilador decide que instrucciones se ejecutan en paralelo.**
- **Power Architecture (Power PC): IBM**
  - **Tipo RISC**
  - **Empleado por: Power PC, Xbox 360, PlayStation 3, etc.**
  - **Implementado por: IBM, Freescale, Microsoft, Sony, etc.**
- **SPARC**
  - **Desarrollado por Sun Microsystems (1980).**
  - **Basado en la arquitectura Berkeley RISC I**
  - **Fue una de las arquitecturas RISC más exitosas, implementada por Atmel, Fujitsu, Texas Instruments, etc).**

x86 Extensiones: x87

- Las primeras versiones de x86 (hasta el 80386) no poseían instrucciones de punto flotante. Se ejecutaban en un coprocesador de punto flotante, cuyo set de instrucciones se denominaba x87 (coprocesadores i287, i387, etc.).
- A partir del 80486 se incluye al x86 la extensión x87:
  - El conjunto de instrucciones x87
  - Los registros del x87 (matriz 8x8 registros de 64 bits)

x86 Extensiones: MMX

- Intel. Destinado a aplicaciones multimedia.
- Agrega:
  - Un conjunto de instrucciones tipo SIMD.
  - 8 registros lógicos (mapeados en los registros x87).



x86 Extensiones

- **3DNow!**
  - **AMD. Destinado a visualización 3D**
  - **Procesamiento vectorial (capacidad de trabajar con arrays de datos)**
  - **Utiliza los mismos registros que MMX, pero empaquetando de a dos datos de punto flotante en cada registro.**
- **VT-x**
  - **Intel. Destinada a virtualización**
    - **Permite que uno o varios SO visitantes compartan el procesador como aplicaciones de un sistema operativo host.**
  - **Agrega 10 instrucciones que permite una mayor performance en la virtualización.**

ARM Extensiones:

- **NEON:**
  - **Añade instrucciones de 64 y 128 bits del tipo SIMD.**
  - **Nuevos tipos de datos tipo SIMD.**
  - **Nuevos registros**
  - **Destinado a visualización**
- **Thumb y Thumb-2: Instrucciones de 16 bits del tipo MISD**
- **Jazelle: Permite ejecutar bytecode de Java (Instrucciones compiladas que puede ejecutar la máquina virtual de Java).**

CISC vs RISC

CISC (Complex Instruction Set Computer)	**RISC** (Reduced Instruction Set Computer)
Set de instrucciones extenso (muchas instrucciones)	**Set de instrucciones reducido** (pocas instrucciones)
Instrucciones para **muchas operaciones matemáticas** (x86 posee instrucciones para multiplicar, dividir, seno, coseno, cálculo de CRC, encriptación, logaritmo, etc.)	**Instrucciones simples.**
Muchos modos de direccionamiento	**Pocos** modos de direccionamiento
Instrucciones de **diferentes longitudes** y formatos	Instrucciones de **igual longitud** y pocos formatos (o uno solo)

CISC (Complex Instruction Set Computer)	**RISC** (Reduced Instruction Set Computer)
Instrucciones aritméticas y lógicas con **operandos en memoria o registros**	Instrucciones aritméticas y lógicas con **operandos solo en registros**
Muchas instrucciones acceden a memoria	Acceden a memoria solo instrucciones LOAD y STORE
Microarquitectura basada en **ROM de microcódigos**.	Microarquitectura **cableada**.
Algoritmos de predicción de saltos	Algoritmos de predicción de saltos, salto demorado y ejecución especulativa
Pipeline complejo con muchas etapas	Pipeline simple con pocas etapas

CISC (Complex Instruction Set Computer)	**RISC** (Reduced Instruction Set Computer)
Unidad de búsqueda (fetch) compleja, requiere pre-decodificador (lee instrucciones de longitud diferente).	Unidad de búsqueda (fetch) simplificada (lee instrucciones todas iguales)
Entrada/Salida mediante instrucciones especiales y mapeo en memoria.	Entrada/Salida mediante mapeo en memoria.
Más fácil implementación de compiladores. (el set de instrucciones se asemeja al lenguaje de alto nivel [1]). Menos instrucciones por programa (Menos páginas. Porcentaje alto de instrucciones en caché.).	CISC un porcentaje bajo de instrucciones es utilizado de manera frecuente. Menor complejidad de microarquitectura. **Bajo consumo de energía**.

[1]Fuente: Intel® 64 and IA-32 Architectures Software Developer's Manual (disponible en Internet)

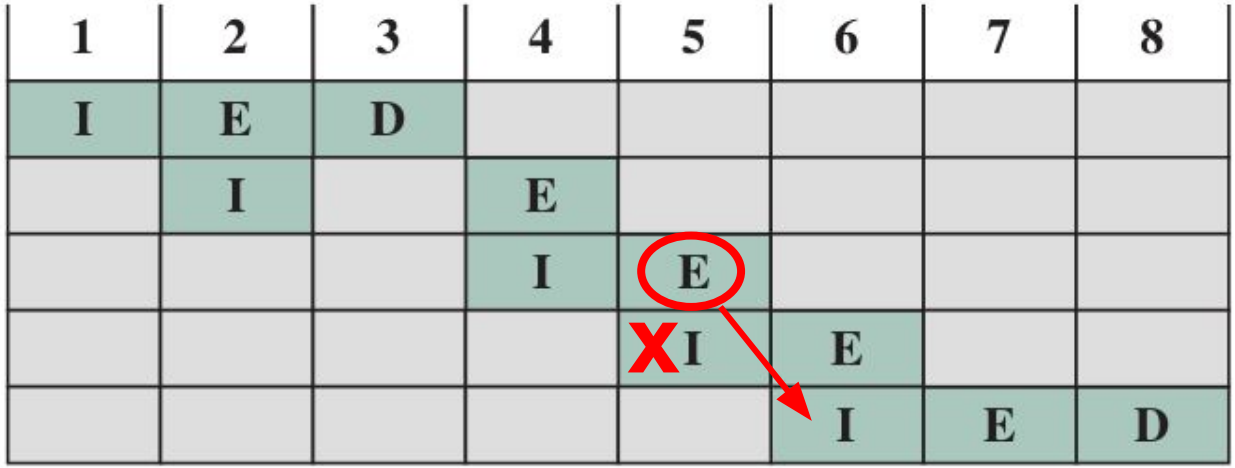
Procesadores RISC

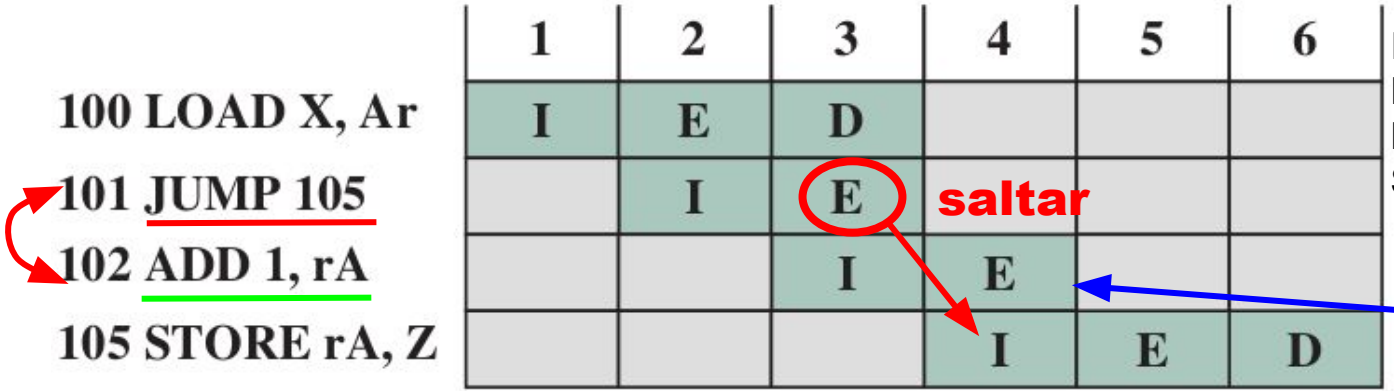
- **RISC: Gran número de registros de propósito general + Técnicas de compilación (de alto nivel) para optimizar el uso de registros**
  - **Si se pudiera asignar un registro a cada variable utilizada por el programa, se reducirían al mínimo posible los accesos a memoria (solo para traer las variables a los registros al comienzo del programa).**
  - **Si esto no es posible (el programa tiene más variables que registros el procesador), se utilizan técnicas de optimización del uso de registros para minimizar los accesos a memoria.**

Nota: Recordar que los registros son las unidades de almacenamiento más rápidas dentro de una computadora.

Pipeline de una Arquitectura RISC

Saltos en arquitectura RISC: Salto demorado





I: Instruction fetch.
E: Execute. Calculates address
D: Memory operation (Load o Store).

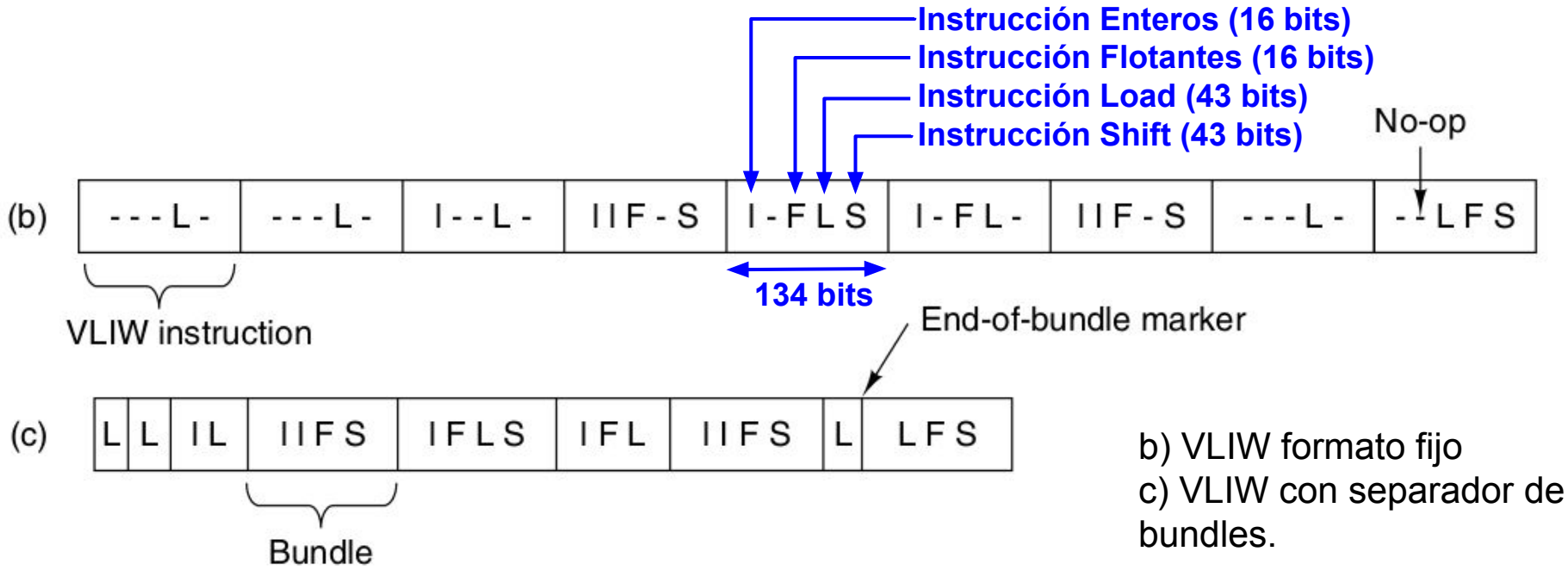
Se termina de ejecutar (se demora el salto)

Salto demorado

- El salto se “demora” N cantidad de instrucciones o ciclos.
  - El salto se ejecutará después de N instrucciones.
- Solo útil cuando las instrucciones son simples y similares (RISC)
- Ahorro del hardware necesario
- No se ejecuta ninguna instrucción no necesaria.
- No se pierden ciclos
- No se puede utilizar si el salto depende de la instrucción inmediatamente anterior
  - Si es posible (no hay dependencia de datos) se adelanta la instrucción de la cual depende el salto.
  - Si hay dependencia de datos, luego de la instrucción de salto se agrega una instrucción NOP y no se elimina del pipeline (ahorro del hardware necesario para vaciar el pipeline)

Procesadores VLIW
(Very Long Instruction Word)

- Cada palabra (llamada instrucción por algunos autores) posee varios códigos de operación y varios operandos (varias instrucciones).



b) VLIW formato fijo
c) VLIW con separador de bundles.

Figura obtenida de Tanenbaum, “Structured computer organization”. 6° ed. Página 556.

Procesadores VLIW
(Very Long Instruction Word)

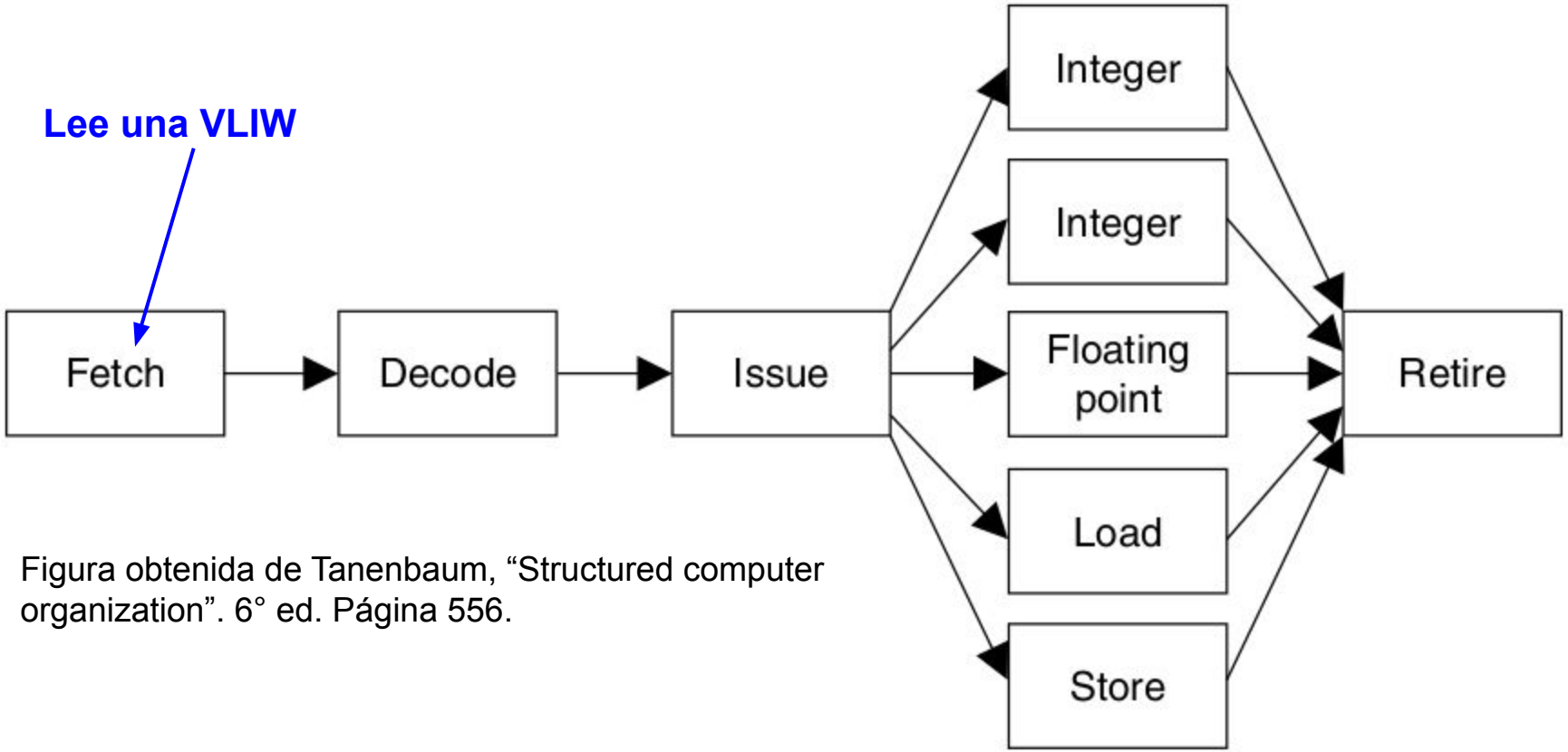


Figura obtenida de Tanenbaum, “Structured computer organization”. 6° ed. Página 556.

Procesadores VLIW
(Very Long Instruction Word)

- Diferencia con procesador superescalar:
  - Superescalar (en orden o fuera de orden): planificación (decisión de que instrucciones se ejecutan en paralelo) se realiza en **tiempo de ejecución**.
    - Necesidad de circuitos que realicen la planificación (circuitos de detección de dependencias, conflicto de recursos, predecodificadores, renombrado de registros, etc.).
  - VLIW (EPIC[1]): planificación se realiza en **tiempo de compilación**.
    - El compilador decide que instrucciones ejecutar en paralelo y las ubica en grupos (bundle[2]).
      - Detecta dependencia de datos, conflictos de recursos, etc.
    - N instrucciones en una única palabra (instrucción VLIW, bundle, etc.).
    - El diseño del procesador es más simple.

[1]EPIC: Explicitly Parallel Instruction Computing. Sigla creada por Intel y HP.

[2]bundle: manojo.

Procesadores VLIW
(Very Long Instruction Word)

Ejemplo: Procesador **TriMedia de Philips**

- Aplicaciones que realizan procesamiento intenso de audio, video.
- La palabra (instrucción) se divide en 5 slots. Cara slots puede contener 1 instrucción.
- El fabricante lo define como un procesador VLIW RISC.
- El procesador no realiza verificación de dependencia de datos, conflicto de recursos, etc.
- Tamaño de palabra: 2 a 28 bytes (16 a 224 bits).

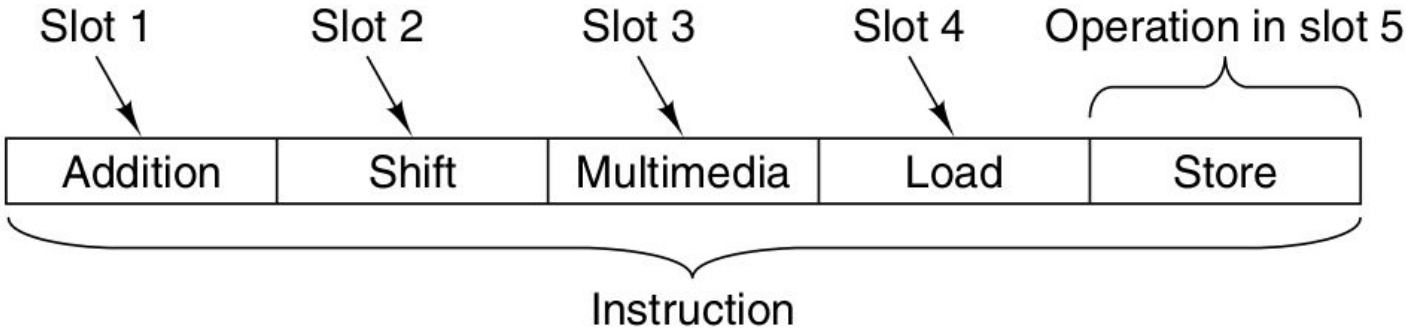


Figura obtenida de Tanenbaum, "Structured computer organization", edición 6°, página 557.

Procesadores VLIW
(Very Long Instruction Word)

Slots en los que puede correr

Unit	Description	#	Lat.	1	2	3	4	5
Constant	Immediate operations	5	1	x	x	x	x	x
Integer ALU	32-Bit arithmetic, Boolean ops	5	1	x	x	x	x	x
Shifter	Multibit shifts	2	1	x	x	x	x	x
Load/Store	Memory operations	2	3				x	x
Int/FP MUL	32-Bit integer and FP multiplies	2	3		x	x		
FP ALU	FP arithmetic	2	3	x			x	
FP compare	FP compares	1	1			x		
FP sqrt/div	FP division and square root	1	17		x			
Branch	Control flow	3	3		x	x	x	
DSP ALU	Dual 16-bit, quad 8-bit multimedia arithmetic	2	3	x		x		x
DSP MUL	Dual 16-bit, quad 8-bit multimedia multiplies	2	3		x	x		

Figura obtenida de Tanenbaum, "Structured computer organization", edición 6°, página 558.

Procesadores VLIW

Ejemplo: Arquitectura IA-64 (Itanium)

- **Intel y HP. 2001. Ultimo procesador: febrero 2017 (serie Itanium 9700 o Kittson) (4-8 núcleos a 2.66 GHz)**
- **3 instrucciones por palabra.**
- **Aplicaciones HPC y servidores**
- **Registros de propósito general: 128 enteros y 128 flotantes.**
- **4-8 unidades de ejecución de diferentes tipos:**
  - **Operaciones con enteros**
  - **Operaciones con flotantes**
  - **Carga y Escritura a memoria**
  - **Saltos.**

La arquitectura IA-64 fue Inicialmente pensada para reemplazar la arquitectura x86 de 32 bits. Pero el éxito de la versión de 64 bits del x86 desarrollada por AMD forzó a Intel a implementar dicho set de instrucciones en sus procesadores., relegando al IA-64 a aplicaciones de HPC y servidores.

Ejemplo de procesador VLIW: IA-64 (Itanium)

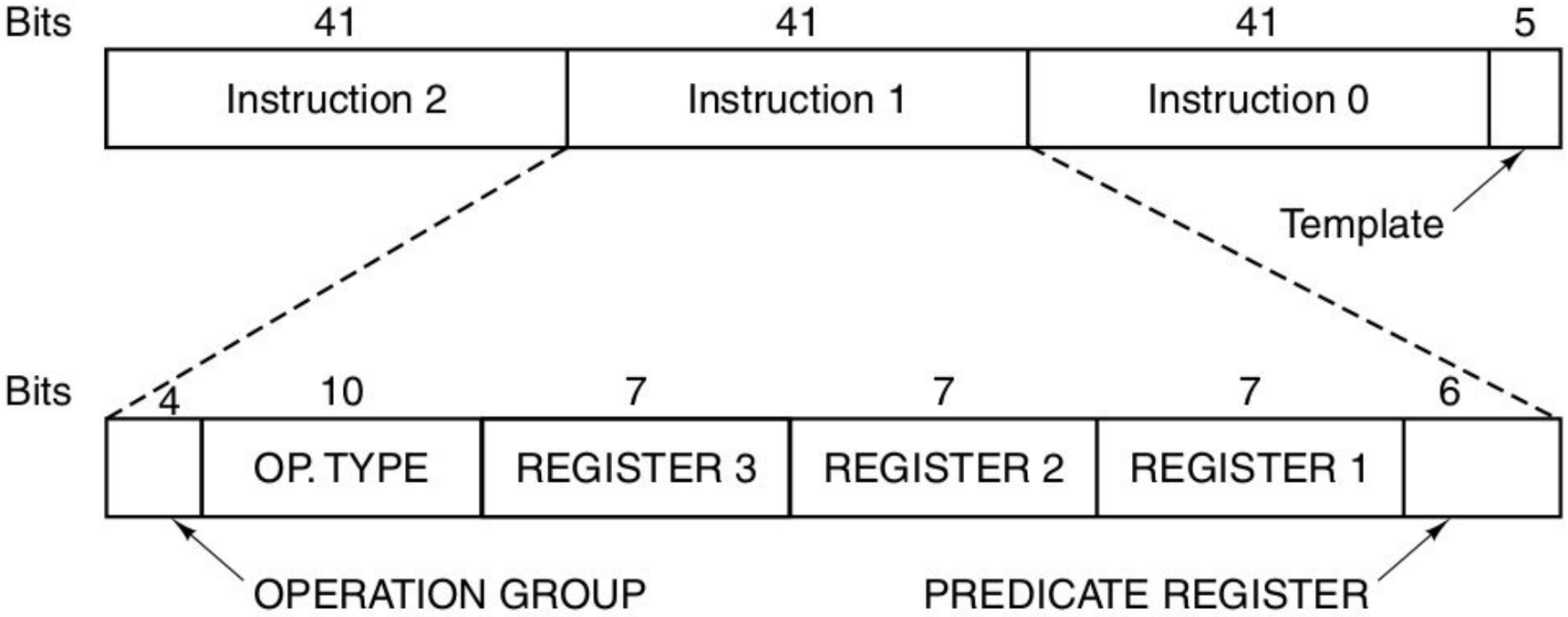


Figura obtenida de Tanenbaum, "Structured computer organization", edición 5°, página 417.

Ejemplo de procesador VLIW: IA-64 (Itanium)

Formato de instrucciones IA-64: Campo Template

Campo **Template**

- Mapea instrucciones con tipos de unidades de ejecución donde deben ejecutarse.
- Indica qué instrucciones pueden ejecutarse en paralelo y cuáles no.
    - Indica dependencia de datos o conflicto de recursos entre instrucciones.

Template	Slot 0	Slot 1	Slot 2
00	M-unit	I-unit	I-unit
01	M-unit	I-unit	I-unit
02	M-unit	I-unit	I-unit
03	M-unit	I-unit	I-unit
04	M-unit	L-unit	X-unit
05	M-unit	L-unit	X-unit
08	M-unit	M-unit	I-unit

M-unit: Instrucciones de acceso a memoria + ALU
I-unit: Operaciones aritméticas con enteros.
B-unit: instrucción de salto
F-unit: Instrucción de punto flotante.
X-unit: se unen dos o más slot para codificar una instrucción.

Procesadores VLIW

(Very Long Instruction Word)

- **Ventajas: No es necesario circuitos para manejar el paralelismo implementados en el procesador:**
  - **Procesador más simple**
  - **Menor consumo**
- **Compilador más complejo:**
  - **Debe decidir qué instrucciones ejecutar en paralelo (planificación).**
  - **Debe resolver dependencias y conflictos.**

Performance y Tests de performance

¿Qué es la “performance”?

1. **Tiempo que la computadora demora en ejecutar un programa o conjunto de programas.**
2. **Cantidad de resultados producidos por unidad de tiempo.**

¿De qué depende la performance del procesador?

- **Velocidad del procesador**
  - **Limitada por la velocidad máxima de operación de los componentes del procesador y el consumo de energía.**
- **Set de instrucciones**
- **Microarquitectura del procesador**
- **Eficiencia del compilador**
- **Lenguaje de programación**
- **Habilidad del programador**

Medidas comunes de Performance

Datos típicos brindados por los fabricantes

- **Ciclos por instrucción promedio (CPI): Número de ciclos de reloj promedio por instrucción ejecutada en un programa.**
  - **Depende del pipeline y/o superescalar y el set de instrucciones.**

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

$$T_{total} = I_c \times CPI \times \tau$$

CPI_i= CPI de la instrucción tipo "i" (dato) depende del pipeline.
I_i=Número total de instrucciones ejecutadas del tipo i
I_c=Número total de instrucciones ejecutadas
τ= Tiempo de un ciclo de reloj

Procesador escalar con pipeline: CPI (ideal)=1
Procesador superescalar con pipeline: CPI (ideal)<1

Medidas comunes de Performance

Datos típicos brindados por los fabricantes

- **Millones de instrucciones por segundo (MIPS [1])**

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

- **Floating-point operations per second (FLOPS)**
  - **Utilizados para comparar computadoras de HPC.**

[1] Los procesadores suelen entregar como dato el valor de BogoMIPS. Bogo es un programa de prueba (lazo que se repite millones de veces) necesario para calcular las demoras en un busy-loop. Bogo viene de "bogus" significa "falso".

Medidas de Performance

¿EL CPI, MIPS o FLOPS sirven para comparar procesadores?
NO (Salvo que sean similares).

RISC de 4 MIPS (0,25 microseg) vs CISC 1 MIPS (1 microseg)
¿es mejor el RISC por tener un MIPS mayor? -> NO

Ejemplo: Necesitamos resolver C = A + B (A, B y C son posiciones de memoria)

RISC	CISC
Load r16, A	Add C, A, B
Load r17, B	
Add r18, r16, r17	
Store C, r18	
demora=1 microseg	demora=1 microseg

Ejemplo basado en: “Computer Organization and Architecture”, 9th Edition, William Stallings

Tests de Performance

- La performance de un procesador (MIPS o FLOPS) puede ser diferente para diferentes aplicaciones.
- Solución -> Tests de performance (benchmark programs, benchmark).
- Conjunto de programas estandarizados usados para comparar computadoras, generalmente agrupados por tipos.

Tests de Performance

- **Características deseables de un Test de performance:**
  - **Escrito en lenguaje de alto nivel (Portable)**
  - **Diferentes tipos de programas (diferentes estilos de programación). Ejemplo:**
    - **Programas de sistema**
    - **Programas para aplicaciones comerciales**
    - **Programas que hacen uso intensivo de cálculo numérico**
    - **Programas que requieren uso de gráficos**
  - **Métricas fácilmente medible**
  - **Procedimientos normalizados para medir.**

Fuente: “Computer Organization and Architecture”, 9th Edition, William Stallings

Ejemplo de tests de Performance estandarizado

- **Tests performance de SPEC (System Performance Evaluation Corporation)**
  - **Consorcio industrial que define y mantiene un conjunto de test de performance ampliamente utilizado.**
  - **Acer, AMD, Amazon, Apple, ARM, Cisco, Dell, Gigabyte, Google, IBM, Intel, Microsoft, Oracle, Qualcomm, Samsung, etc.**
  - **www.spec.org**

Fuente: http://www.spec.org

Ejemplo: SPEC CPU2017

- **Aplicaciones que hacen uso intensivo del procesador (43 benchmarks).**
- **Lenguajes C, C++ y Fortran**
- **Objetivo: Estresar procesador, memoria y compilador.**
- **Se provee el código fuente -> Hay que compilar**
- **Dos métricas:**
  - **SPECspeed: tiempo para completar la ejecución de un conjunto de tareas**
  - **SPECrate: throughput (cantidad de tareas por unidad de tiempo)**
- **Unidades de los resultados: relación respecto a una máquina referencia: Sun Microsystems server, UltraSPARC 2.1 GHz.**
- **Resultados SPECrate punto flotante**
  - **Cisco UCS C480 M5 (104 núcleos de 2.5 GHz): 439**
  - **Dell PowerEdge R740xd (8 núcleos de 2.1 GHz): 8.40**

Fuente: http://www.spec.org

Otros Tests SPEC

- **SPEC CPU2006: Versión anterior de SPEC CPU2017**
- **SPEC Cloud IaaS 2016: performance del servicio de IaaS provisto por proveedores públicos o privados. Parámetros que mide: escalabilidad, elasticidad y tiempo de provisión de instancias.**
- **Tests de performance de presentación de gráficos 3D.**
- **HPC.**
- **JVM (Java Virtual Machine), para medir la performance de aplicaciones que corren sobre la JVM.**
- **Almacenamiento: Performance de servidores de archivos y tiempo de respuesta**
- **Virtualización**

Fuente: http://www.spec.org

Bibliografía

Libros:

- Computer Organization and Architecture; William Stallings; ediciones 8°, 9º y 10°.
- Structured computer organization; Andrew S. Tanenbaum, Todd Austin; 6° edición. 2013.
- Arquitectura de Computadoras; Patricia Quiroga; 1º edición; 2010; capítulo 10.

Páginas web:

- www.spec.org
- https://www.amd.com/es/technologies/zen-core-3

Especificaciones:

- Intel® 64 and IA-32 Architectures Software Developer's Manual (disponible en Internet)

Comandos útiles sobre procesadores

sudo cat /proc/loadavg: carga relativa promedio del procesador en los últimos 1, 5 y 10 minutos, el número de procesos total y el número en ejecución, y el PID del último proceso activo.

lscpu: información sobre el procesador

arch: información sobre la arquitectura del procesador