2º curso / 2º cuatr.

Grados en
Ing. Informática

Arquitectura de Computadores
Tema 1

Arquitecturas Paralelas: Clasificación y Prestaciones

Material elaborado por Mancia Anguita
Profesores: Mancia Anguita, Maribel García y Christian Morillas

ugr Universidad de Granada

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

ATC
Departamento de Arquitectura y Tecnología de Computadores
UNIVERSIDAD DE GRANADA

Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - Medidas usuales para evaluar prestaciones
  - Conjunto de programas de prueba (*Benchmark*)
  - Ganancia en prestaciones

Objetivos Lección 3

- Distinguir entre tiempo de CPU (sistema y usuario) de unix y tiempo de respuesta
- Distinguir entre productividad y tiempo de respuesta
- Obtener, de forma aproximada mediante cálculos, el tiempo de CPU, GFLOPS y los MIPS del código ejecutado en un núcleo de procesamiento
- Calcular ganancia en prestaciones/velocidad
- Aplicar la ley de Amdahl

Bibliografía

- Fundamental
  - Capítulo 1, M. Anguita, J. Ortega. Fundamentos y problemas de Arquitectura de Computadores, Editorial Técnica Avicam. ESIIT/C.1 ANG fun
  - Secciones 1.2,1.4, 7.5.1. J. Ortega, M. Anguita, A. Prieto. *Arquitectura de Computadores*, Thomson, 2005. ESIIT/C.1 ORT arq
- Complementaria
  - T. Rauber, G. Ründer. *Parallel Programming: for Multicore and Cluster Systems*. Springer 2010. Disponible en línea (biblioteca UGR): http://dx.doi.org/10.1007/978-3-642-04818-0

Contenido

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - **Medidas usuales para evaluar prestaciones**
    - Tiempo de respuesta
    - Productividad
  - Ganancia en prestaciones al realizar una mejora
  - Conjunto de programas de prueba (*Benchmark*)

Tiempo de respuesta de un programa en una arquitectura

- Real (*wall-clock time, elapsed time, real time*)
- *CPU time* = *user* + *sys* (no incluye todo el tiempo)
- Con un flujo de instrucciones
  - elapsed >= CPU time
- Con múltiples flujos de instrucciones
  - elapsed < CPU time, elapsed >= CPU time/nº flujos control

```
$ time ./program.exe
elapsed 5.4
user 3.2
sys 1.0
```

Elapsed >= CPU time

Tiempo de CPU de usuario (Tiempo en ejecución en espacio de usuario)

Tiempo de CPU de sistema (Tiempo en el nivel del kernel del SO)

Tiempo asociado a las esperas debidas a I/O o asociados a la ejecución de otros programas.

Comando **time** en Unix: 3.2u 1.0s 5.4

3.2+1.0 es el 78% del tiempo transcurrido (5.4)

Algunas alternativas para obtener tiempos

Función	Fuente	Tipo	Resolución aprox. (microsegundos)
time	SO (/usr/bin/time)	*elapsed, user, system*	10000
clock()/CLOCKS_PER_SEC	SO (time.h)	*CPU*	10000
gettimeofday()	SO (sys/time.h)	*elapsed*	1
clock_gettime()/clock_getres()	SO (time.h)	*elapsed*	0.001
omp_get_wtime()/ omp_get_wtick()	OpenMP (omp.h)	*elapsed*	0.001
SYSTEM_CLOCK()	Fortran	*elapsed*	1

La resolución depende de la plataforma

Tiempo de CPU


```
        ...
for (i=0; i<N; i++) {
    v3[i]=v1[i]+v2[i];
}
        ...
```

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

$$T_{CPU} = NI \times \frac{1}{IPC} \times T_{ciclo}$$

$$T_{CPU} = \frac{Nº ciclos\ código}{} \times T_{ciclo}$$

$$T_{CPU} = \left[\sum_i NI_i \times CPI_i\right] \times T_{ciclo}$$

$$T_{CPU} = NI \times \left(\frac{\sum_i NI_i \times CPI_i}{NI}\right) \times T_{ciclo}$$

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

Suma vectores de *doubles*

```
        ...
.L7:                    ; rax=0, rbx=8N
        movsd   v1(%rax), %xmm0
        addsd   v2(%rax), %xmm0
        movsd   %xmm0, v3(%rax)
        addq    $8, %rax
        cmpq    %rbx, %rax
        jne     .L7
        ...
```

i	NI_i	CPI_i
movsd m,r movsd r,m	2N	4
addsd m,r	N	5
addq i,r	N	1
cmp r,r	N	1
jne	N	1
	6N	

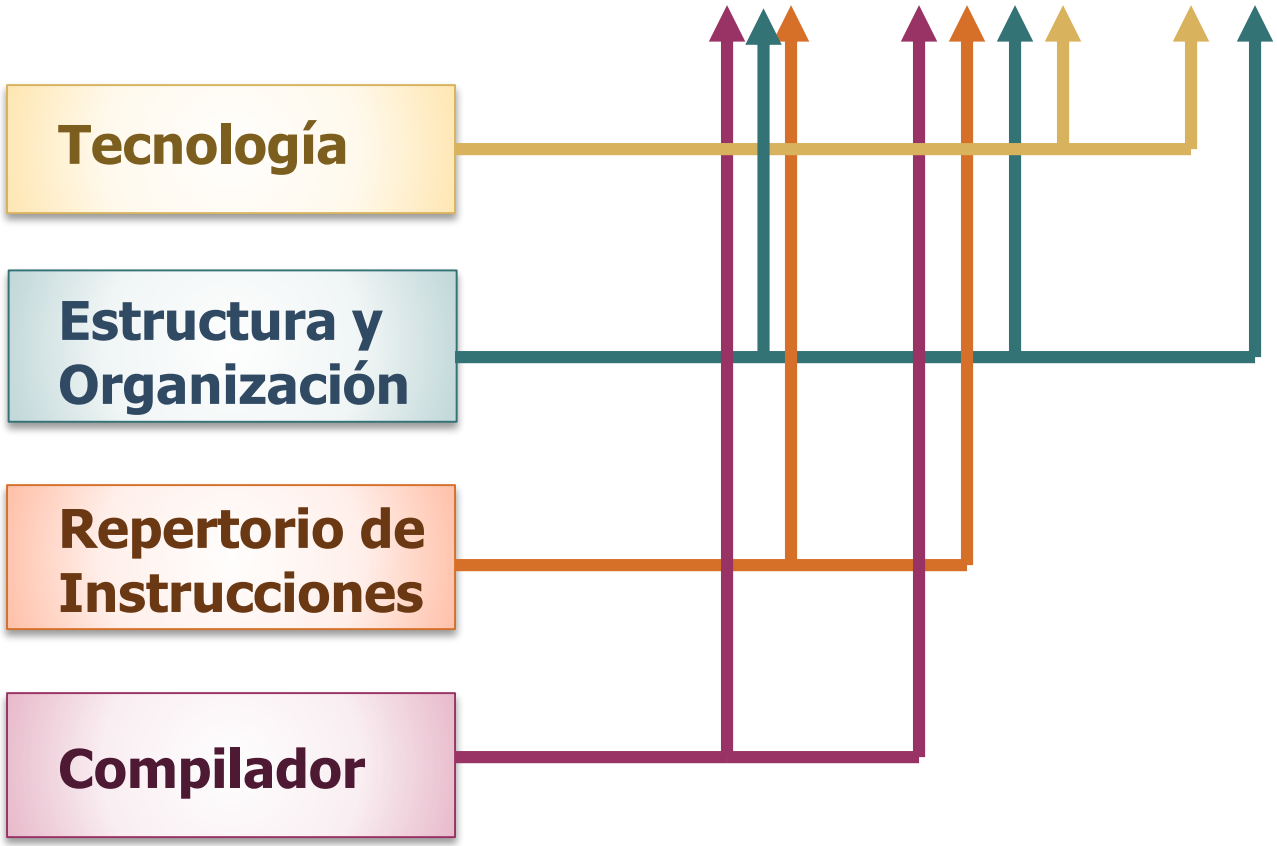
$T_{ciclo} = 1/F$

Para $N = 10^3$ y $F = 100MHz$ ($\Rightarrow T_{ciclo} = 10^{-8}$seg./ciclo):

$$T_{CPU} \approx \left[6N \times \left(\frac{2N \times 4 + N \times 5 + 3N \times 1}{6N}\right)\right] \times T_{ciclo}$$
$$= 10^3 \times 16\ ciclos/código \times 10^{-8} seg./ciclo$$
$$= 16 \times 10^{-5} seg./código$$

Tiempo de CPU

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$



MIPS y MFLOPS

- Millones de instrucciones por segundo (MIPS):

$$MIPS = \frac{NI}{T_{CPU} \times 10^6}$$

$$MIPS = \frac{NI}{NI \times CPI \times T_{ciclo} \times 10^6} = \frac{F}{CPI \times 10^6}$$

  - **Depende del repertorio de instrucciones** (difícil la comparación de máquinas con repertorios distintos)
  - **Puede variar inversamente con las prestaciones** (mayor valor de MIPS corresponde a peores prestaciones)

- Millones de operaciones punto flotante por segundo (MFLOPS):

$$MFLOPS = \frac{n^{\underline{o}}\ FP}{T_{CPU} \times 10^6}$$

MIPS y FLOPS

```
...
for (i=0; i<N; i++) {
   y[i]=a*x[i]+y[i];
}
...
```


-O2

```
;r12=&x,r13=&y,rax=0,rbp=N,xmm1=a
.L6:
  movsd (%r12,%rax,8), %xmm0
  mulsd %xmm1, %xmm0
  addsd (%r13,%rax,8), %xmm0
  movsd %xmm0, (%r13,%rax,8)
  addq  $1, %rax
  cmpl  %eax, %ebp
  jg .L6
```

$T(N=2^{26})=0.182$ seg.

$$GIPS = \frac{NI}{T_{CPU} \times 10^9} = \frac{N \times 7}{0.182 \times 10^9} = \frac{2^{26} \times 7}{0.182 \times 10^9} \approx 2.58\ GIPS$$

$$GFLOPS = \frac{nº\ FP}{T_{CPU} \times 10^9} = \frac{N \times 2}{0.182 \times 10^9} = \frac{2^{26} \times 2}{0.182 \times 10^9} \approx 0.737\ GFLOPS$$

-O3

```
;r12=&x,r13=&y,rax=0,rbp=N/2,xmm1=a
.L7:
  movapd (%r12), %xmm0
  addq   $1, %rax
  addq   $16, %r12
  addq   $16, %r13
  mulpd  %xmm1, %xmm0
  addpd  -16(%r13), %xmm0
  movaps %xmm0, -16(%r13)
  cmpl   %ebp, %eax
  jb .L7
```

$T(N=2^{26})=0.178$ seg.

$$GIPS = \frac{NI}{T_{CPU} \times 10^9} = \frac{(N/2) \times 9}{0.178 \times 10^9} = \frac{2^{25} \times 9}{0.178 \times 10^9} \approx 1.7 GIPS$$

$$GFLOPS = \frac{2^{26} \times 2}{0.178 \times 10^9} \approx 0.754\ GFLOPS$$

Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - Medidas usuales para evaluar prestaciones
  - **Ganancia en prestaciones al realizar una mejora**
  - Conjunto de programas de prueba (*Benchmark*)

Mejora o Ganancia en Prestaciones (*Speed-up* o ganancia en velocidad)

Si se incrementan las prestaciones de un sistema, el incremento en prestaciones (velocidad) que se consigue en la nueva situación, ***p***, con respecto a la previa (**sistema base**, ***b***) se expresa mediante la ganancia en prestaciones o *speed-up*, ***S***

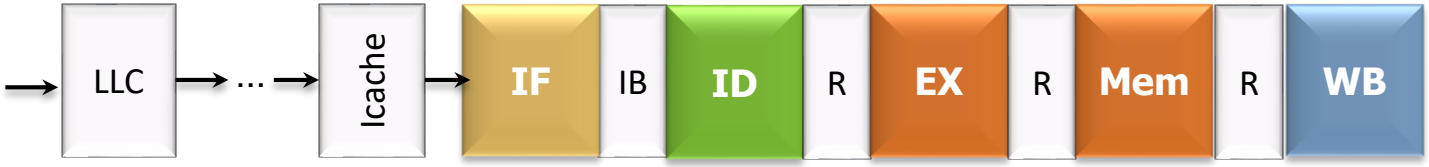
$$S = \frac{V_p}{V_b} = \frac{T_b}{T_p}$$

$$S = \frac{T_{CPU}^b}{T_{CPU}^p} = \frac{NI^b \times CPI^b \times T_{ciclo}^b}{NI^p \times CPI^p \times T_{ciclo}^p}$$

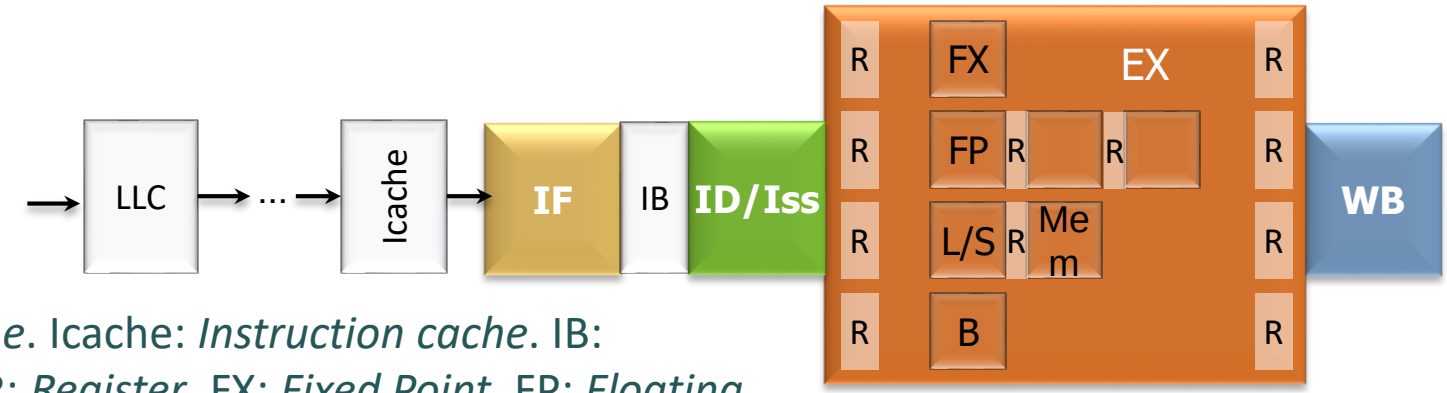
V_b Velocidad de la máquina base
V_p Velocidad de la máquina mejorada (un factor p en uno de sus componentes)
T_b Tiempo de ejecución en la máquina base
T_p Tiempo de ejecución en la máquina mejorada

Arquitecturas con paralelismo a nivel de instrucción (ILP)

Escalar segmentada



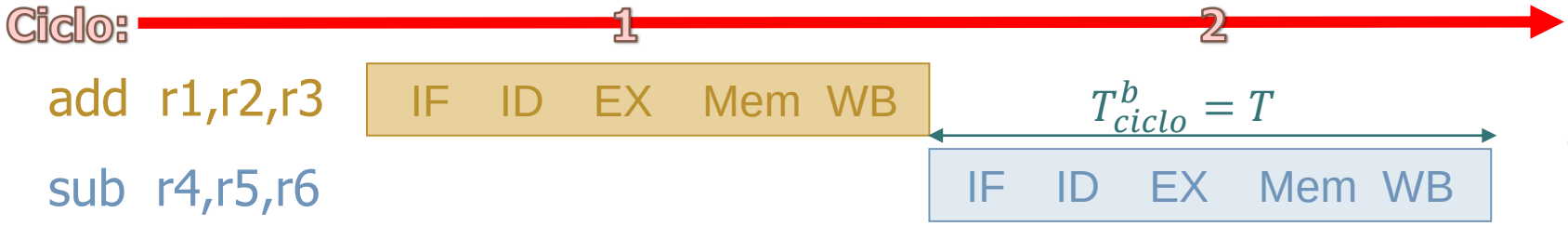
VLIW y superescalar



LLC: *Last Level Cache*. Icache: *Instruction cache*. IB: *Instruction Buffer*. R: *Register*. FX: *Fixed Point*. FP: *Floating Point*. L/S: (*memory*) *Load/Store*. B: *Branch*

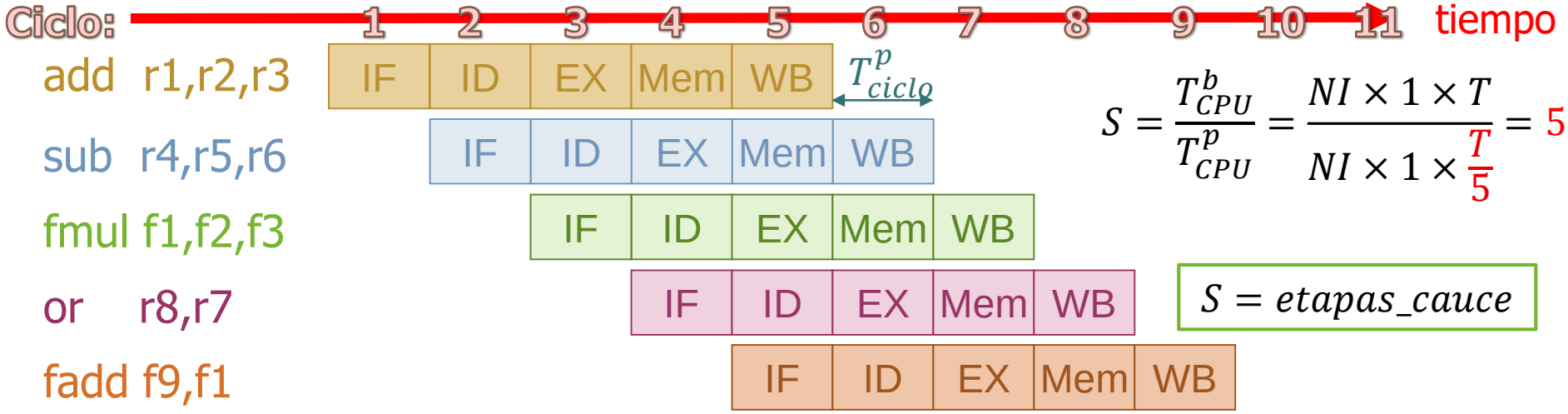
- Etapa de captación de instrucciones (***I**nstruction **F**etch*)
- Etapa de decodificación de instrucciones y emisión a unidades funcionales (***I**nstruction **D**ecode/Instruction **Iss**ue)* (incluye captura de "operandos" de los registros de la arquitectura)
- Etapas de ejecución (***Ex**ecution*). Etapa de acceso a memoria (***Mem**ory*)
- Etapa de almacenamiento de resultados en registros de la arquitectura (***W**rite-**B**ack*)

Mejora en un núcleo de procesamiento: segmentación



$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI^b \times CPI^b \times T^b_{ciclo}}{NI^p \times CPI^p \times T^p_{ciclo}}$$

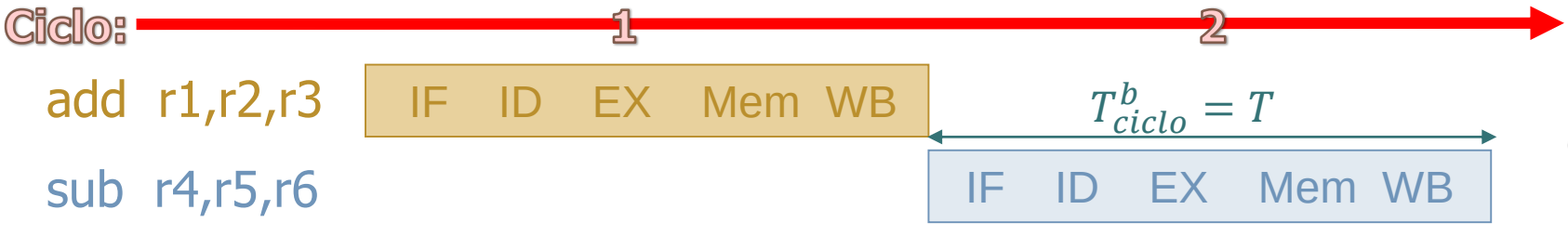
Núcleo **segmentado** en **5 etapas**:



$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI \times 1 \times T}{NI \times 1 \times \frac{T}{5}} = 5$$

$$S = etapas\_cauce$$

Mejora en un núcleo de procesamiento: operación superscalar



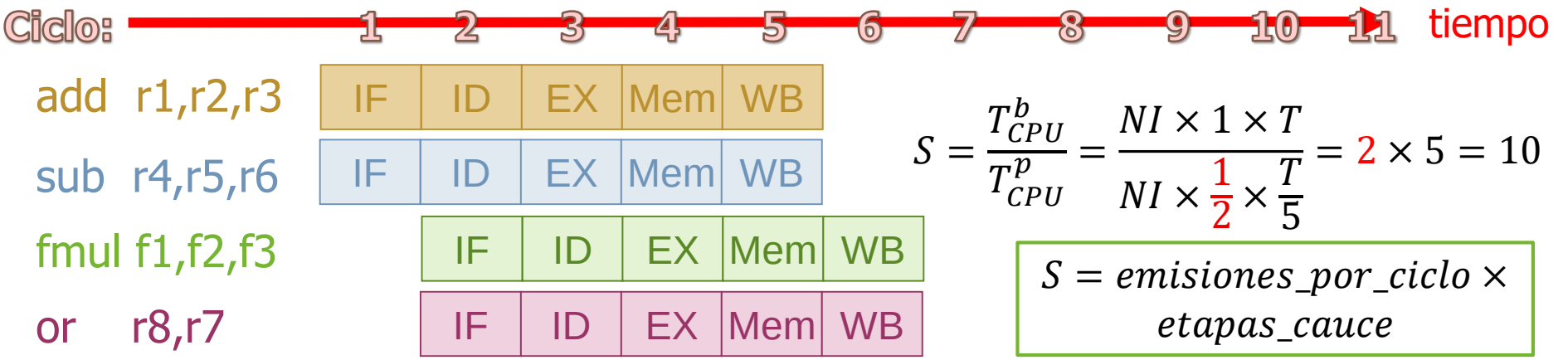
$NI^b = NI$

$CPI^b = 1$

$T^b_{ciclo} = T$

$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI^b \times CPI^b \times T^b_{ciclo}}{NI^p \times CPI^p \times T^p_{ciclo}}$$

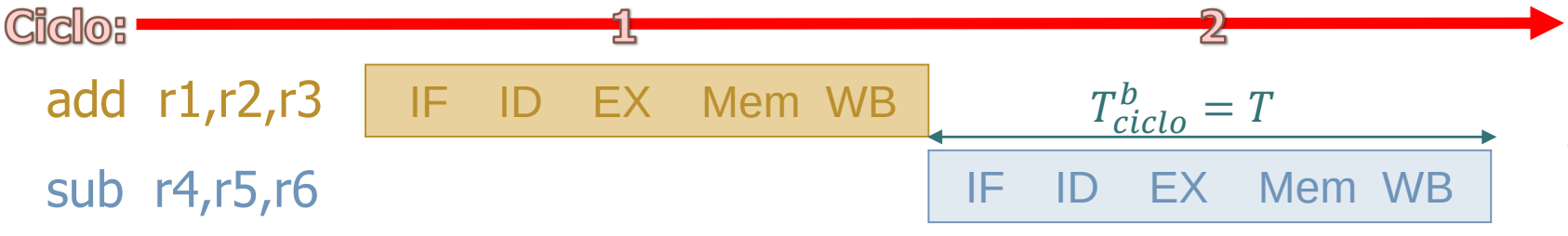
Núcleo **superescalar** con **2 emisiones por ciclo** y **5 etapas**:



$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI \times 1 \times T}{NI \times \frac{1}{2} \times \frac{T}{5}} = 2 \times 5 = 10$$

$$S = emisiones\_por\_ciclo \times etapas\_cauce$$

Mejora en un núcleo de procesamiento: unidades funcionales SIMD



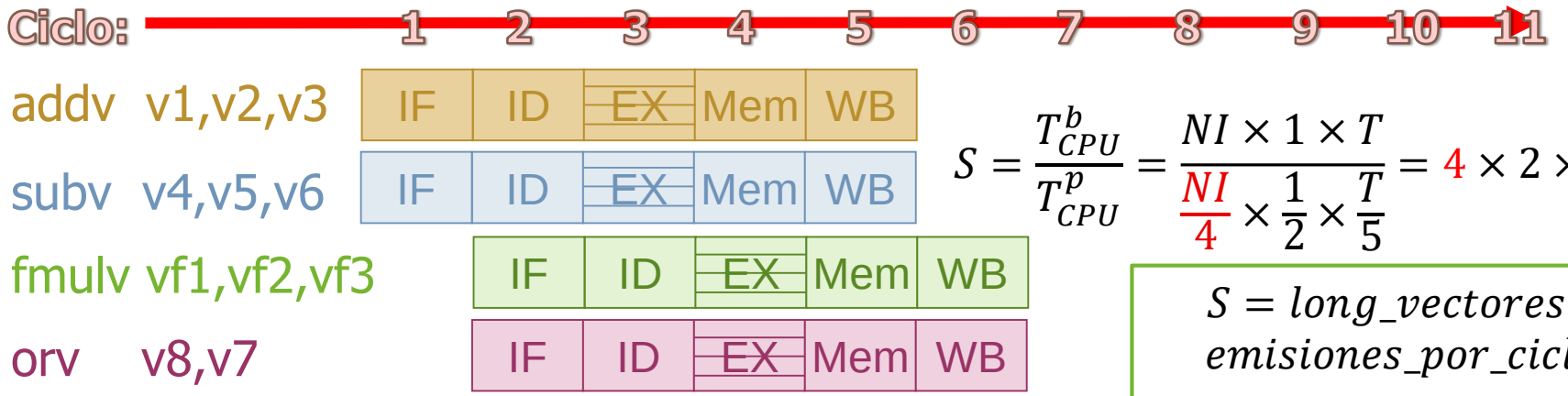
$NI^b = NI$

$CPI^b = 1$

$T^b_{ciclo} = T$

$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI^b \times CPI^b \times T^b_{ciclo}}{NI^p \times CPI^p \times T^p_{ciclo}}$$

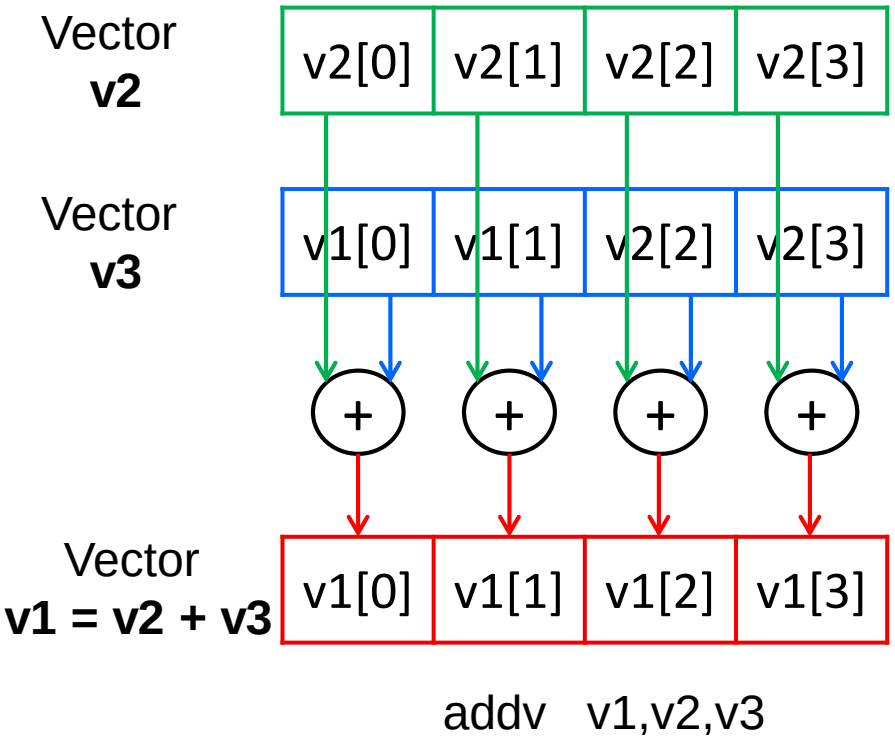
Núcleo **superescalar** con **2 emisiones por ciclo** y **5 etapas**, y
unidades funcionales SIMD (vectoriales) que procesan **vectores de 4 componentes**
(suponemos el mejor caso: el código genera sólo instrucciones que operan con vectores de 4 comp



$$S = \frac{T^b_{CPU}}{T^p_{CPU}} = \frac{NI \times 1 \times T}{\frac{NI}{4} \times \frac{1}{2} \times \frac{T}{5}} = 4 \times 2 \times 5 = 40$$

$S = long\_vectores \times emisiones\_por\_ciclo \times etapas\_cauce$

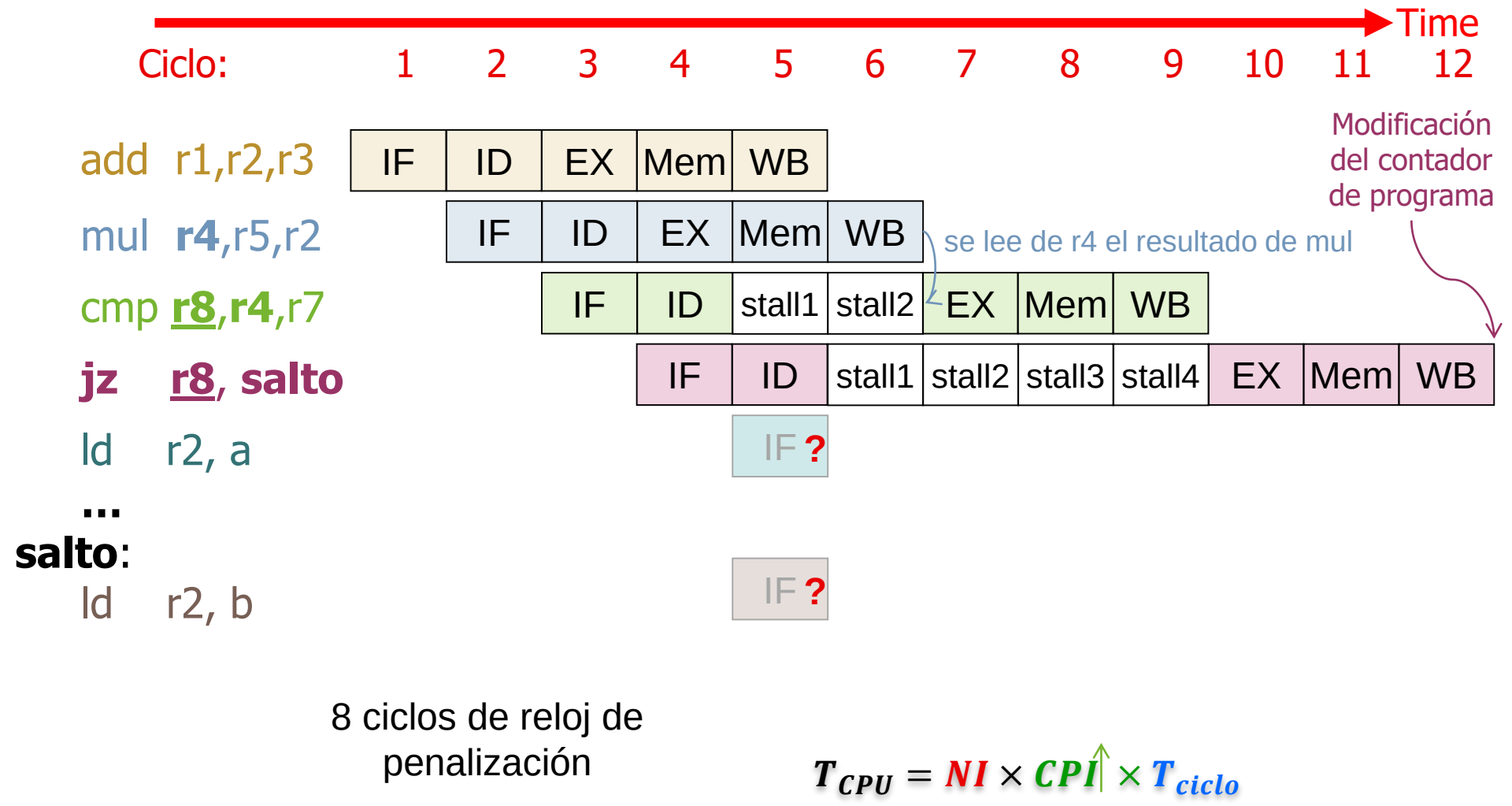
Paralelismo de datos. Ej: suma de dos vectores



¿Qué impide que se pueda obtener la ganancia en velocidad pico?

- Riesgos:
  - Datos
  - Control
  - Estructurales
- Accesos a memoria (debido a la jerarquía)

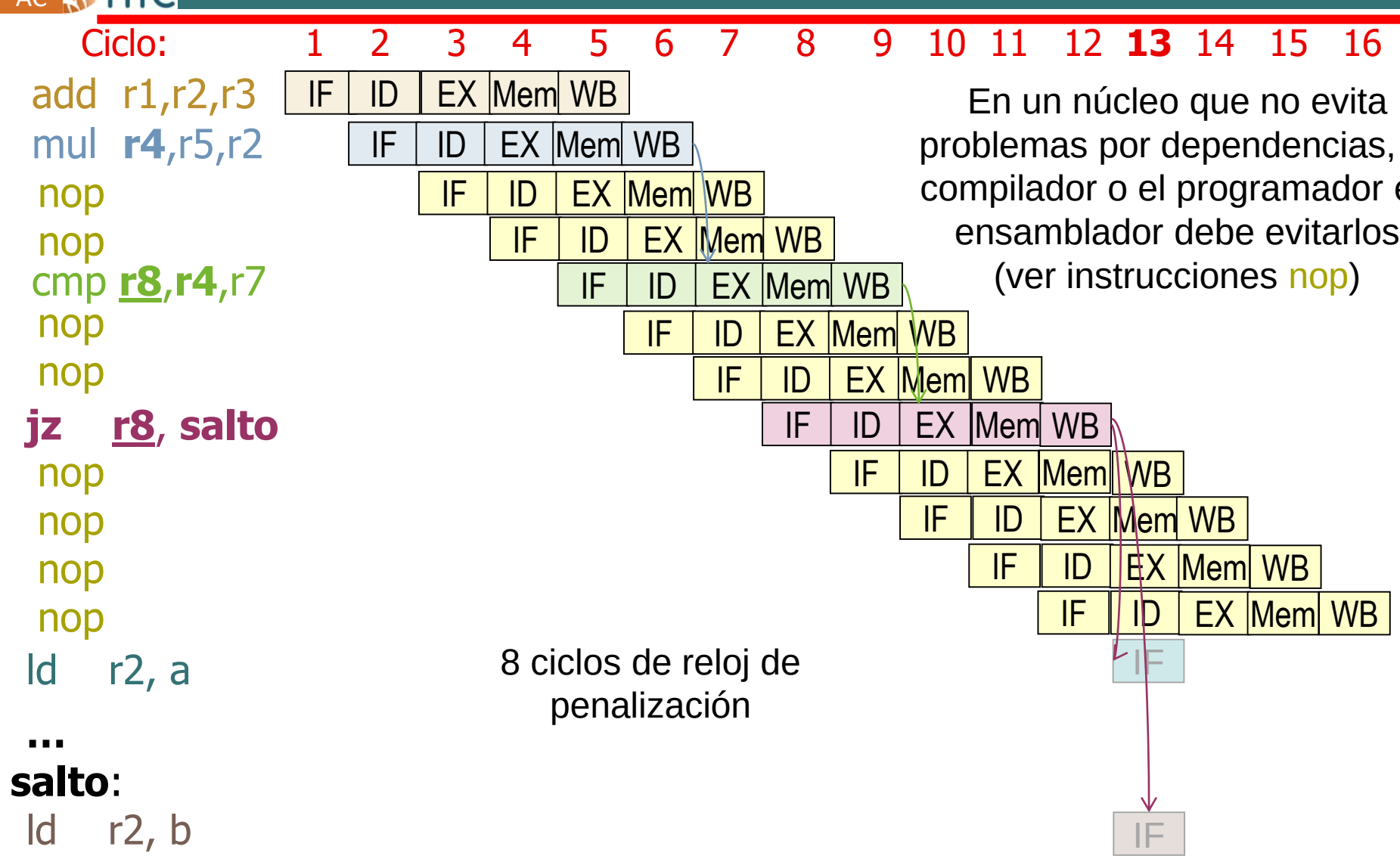
Riesgos (*hazards*): de datos, de control



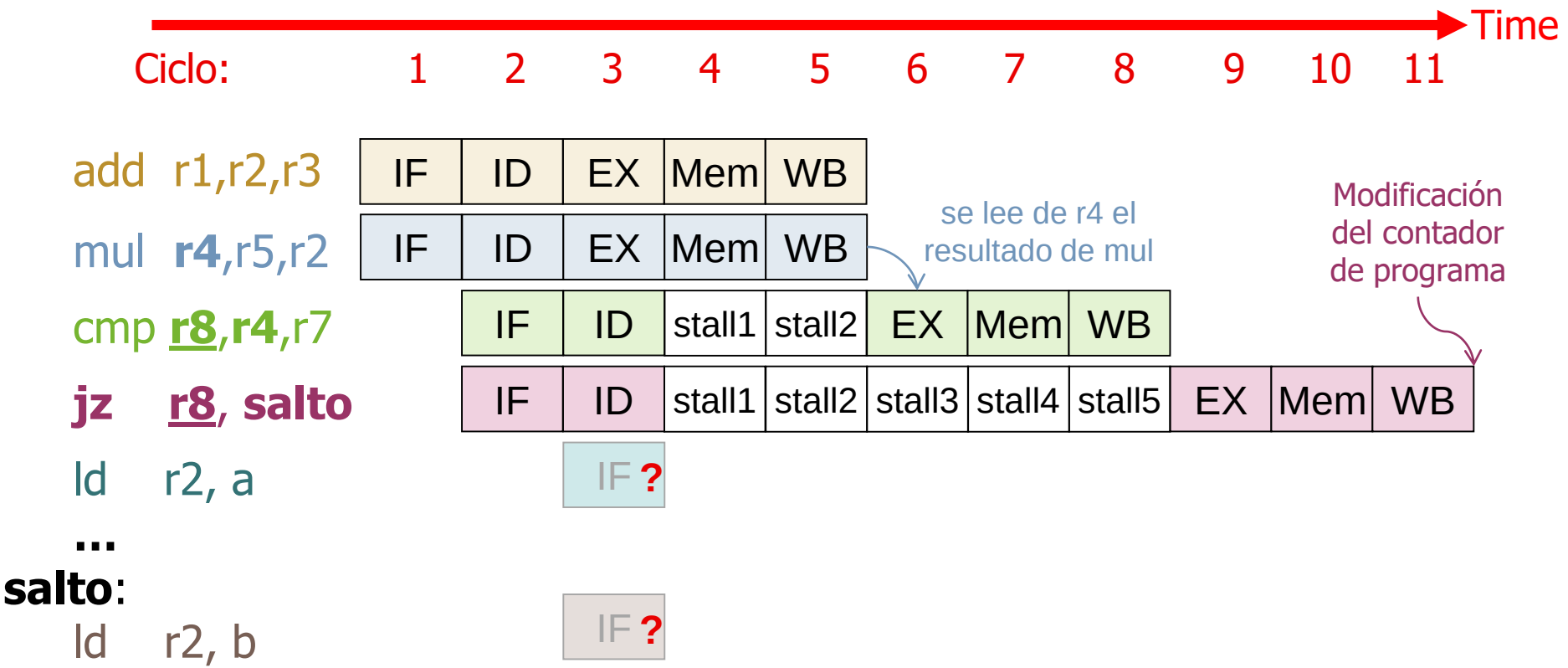
8 ciclos de reloj de penalización

$$T_{CPU} = NI \times CPI \times T_{ciclo}$$

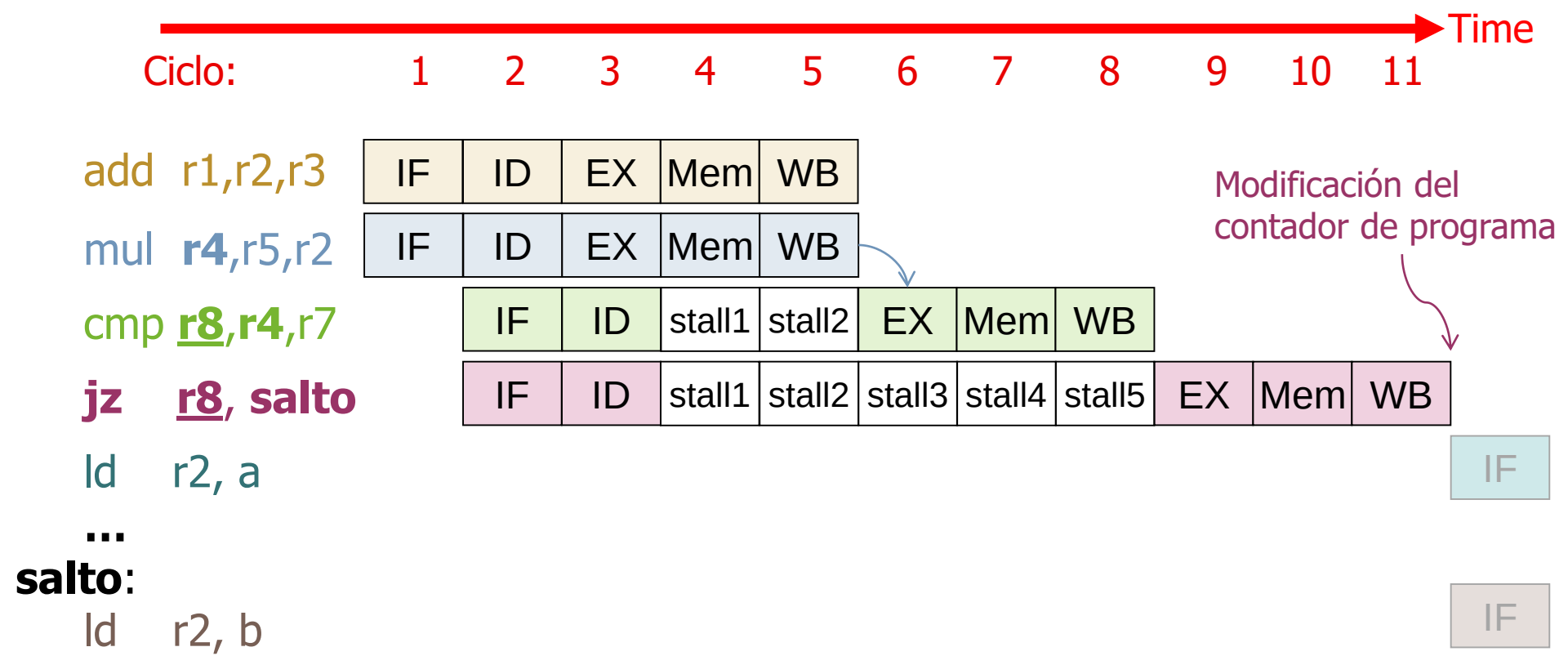
Riesgos (*hazards*): de datos, de control



Riesgos (*hazards*): de **datos**, de **control**

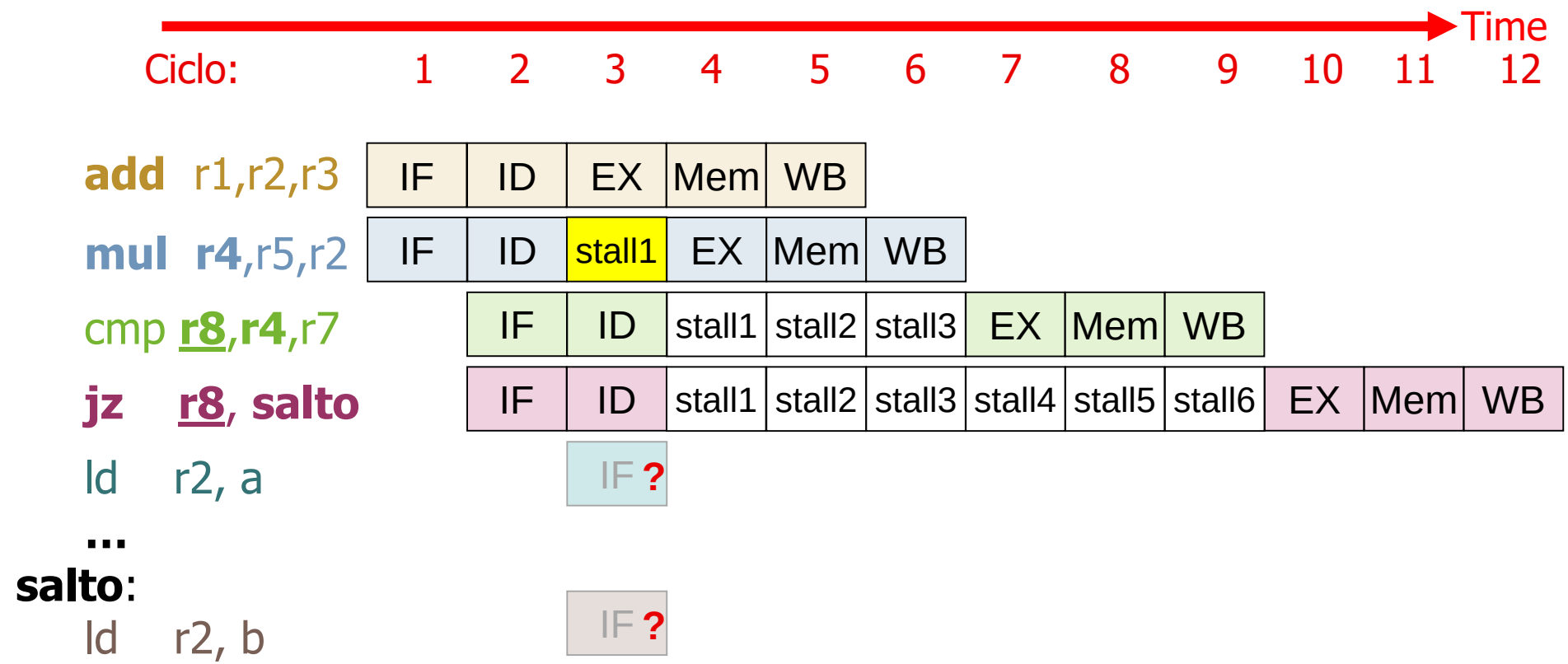


Riesgos (*hazards*): de **datos**, de **control**



9 ciclos de reloj de penalización

Riesgos (*hazards*): de datos, de control, **estructural**



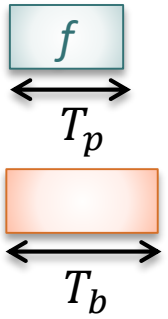
add y mul usan la misma unidad funcional

10 ciclos de reloj de penalización

Ley de Amdahl

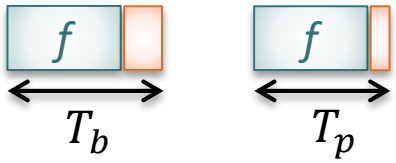
La mejora de velocidad, ***S***, que se puede obtener cuando se mejora un recurso de una máquina en un factor ***p*** está limitada por:

$$S = \frac{V_p}{V_b} = \frac{T_b}{T_p} \leq \frac{1}{f + \frac{1-f}{p}} = \frac{p}{1 + f(p-1)} \quad \begin{matrix} \xrightarrow[p\to\infty]{} \frac{1}{f} \\ \xrightarrow[f\to 0]{} p \end{matrix}$$



donde ***f*** es la fracción del tiempo de ejecución del sistema base (i.e. de la máquina sin la mejora) durante el que no se usa dicha mejora.

Ejemplo: Si un programa pasa un 25% de su tiempo de ejecución en una máquina realizando instrucciones de coma flotante, y se mejora la máquina haciendo que estas instrucciones se ejecuten en la mitad de tiempo, entonces ***p*=2, *f*=0.75 y**



$$S = \frac{T_b}{T_p} = \frac{1}{0.75 + \frac{0.25}{2}} \approx 1.14$$

Habría que mejorar el caso más frecuente (lo que más se usa)

Lecciones

- Lección 1. Clasificación del paralelismo implícito en una aplicación
- Lección 2. Clasificación de arquitecturas paralelas
- Lección 3. Evaluación de prestaciones de una arquitectura
  - Medidas usuales para evaluar prestaciones
  - Ganancia en prestaciones al realizar una mejora
  - **Conjunto de programas de prueba (*Benchmark*)**

Benchmarks

- Propiedades exigidas a medidas de prestaciones:
  - Fiabilidad => *Representativas, evaluar diferentes componentes del sistema* y *reproducibles*
  - Permitir comparar diferentes realizaciones de un sistema o diferentes sistemas => Aceptadas por todos los interesados (usuarios, fabricantes, vendedores)
- Interesados:
  - Vendedores y fabricantes de hardware o software.
  - Investigadores de hardware o software.
  - Compradores de hardware o software.

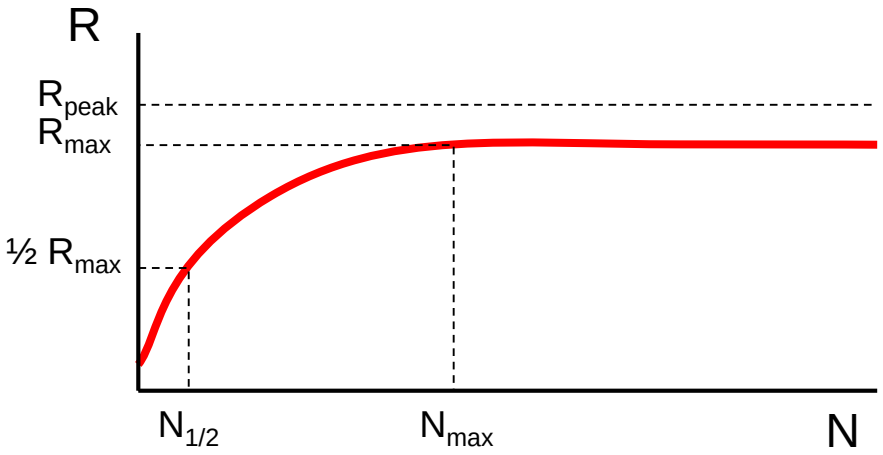
Tipos de *Benchmarks*

- Tipos de Benchmark:
  - De bajo nivel o microbenchmark
    - test ping-pong, evaluación de las operaciones con enteros o con flotantes
  - Kernels
    - resolución de sistemas de ecuaciones, multiplicación de matrices, FFT, descomposición LU
  - Sintéticos
    - Dhrystone, Whetstone
  - Programas reales
    - Ej. SPEC CPU2017 (https://www.spec.org/): enteros (gcc, perlbmk),
  - Aplicaciones diseñadas
    - Predicción de tiempo, dinámica de fluidos, animación etc. (p. ej. SPEC2017).

LINPACK

El núcleo de este programa es una rutina denominada DAXPY (***D**ouble precision- real **A**lpha **X** **P**lus **Y***) que multiplica un vector por una constante y los suma a otro vector. Las prestaciones obtenidas se escalan con el valor de **N** (tamaño del vector):

```
for (i=0 ; i<N ; i++)
 y[i] = alpha*x[i] + y[i];
```



https://www.top500.org/project/linpack/

TOP500

R_{max}: Número máximo de Gflops alcanzados

R_{peak}: Límite teórico del sistema (en Gflops)

Rank	Site Country/Year	Computer / Processors Manufacturer	R_{max} R_{peak}
1	DOE/NNSA/LLNL United States/2005	*BlueGene/L* **eServer Blue Gene Solution** / 65536 IBM	136800 183500
2	IBM Thomas J. Watson Research Center United States/2005	*BGW* **eServer Blue Gene Solution** / 40960 IBM	91290 114688
3	NASA/Ames Research Center/NAS United States/2004	*Columbia* **SGI Altix 1.5 GHz, Voltaire Infiniband** / 10160 SGI	51870 60960
4	The Earth Simulator Center Japan/2002	**Earth-Simulator** / 5120 NEC	35860 40960
5	Barcelona Supercomputer Center Spain/2005	*MareNostrum* **JS20 Cluster, PPC 970, 2.2 GHz, Myrinet** / 4800 IBM	27910 42144
6	ASTRON/University Groningen Netherlands/2005	**eServer Blue Gene Solution** / 12288 IBM	27450 34406.4

Para ampliar .....

- Páginas Web:
  - http://www.top500.org
  - http://en.wikipedia.org/wiki/LINPACK
- Artículos de Revistas:
  - Henning, J.L.: “SPEC CPU2000: Measuring CPU Performance in the New Millenium”. IEEE Computer. Julio, 2000.
  - O’Neal, D.: “On Microprocessors, Memory Hierarchies, and Amdahl’s Law”.