

## Tema 4: Rendimiento del procesador.

### Objetivos:

- Introducir los conceptos y criterios que permitan medir de forma cuantitativa el rendimiento de los procesadores.
- Estudiar los diferentes patrones de medida (*benchmark*).
- Analizar de forma cualitativa y con datos reales la influencia que tienen las diferentes alternativas de diseño, estudiadas en temas anteriores, sobre el rendimiento.
- Sintetizar el resultado del anterior análisis en un conjunto de alternativas de diseño que son determinantes para aumentar el rendimiento del procesador y que se concretan en la alternativa RISC.
- Analizar la forma de explotar al máximo el rendimiento del procesador desde análisis del programa realizado por el compilador.

### Contenido:

1. Medidas del rendimiento de un computador
2. Patrones de medida (*Benchmarks*)
3. Influencia en el rendimiento de las alternativas de diseño
4. Influencia de los compiladores de lenguajes de alto nivel
5. Procesadores RISC y CISC

### 1. Medidas del rendimiento de un computador

Cuando se quieren comparar diferentes procesadores es necesario establecer el criterio de medida que permita cuantificar los resultados de la comparación. En este sentido existen dos conceptos que conviene aclarar previamente: la *unidad de medida* y el *patrón de medida*. El primero se refiere a la métrica utilizada para cuantificar la comparación. Y el segundo a la carga de trabajo respecto a la que se realiza la comparación.

**El tiempo es la unidad de medida** por excelencia cuando se comparan varios procesadores, aunque no siempre coincidan los puntos de vista de los diferentes observadores. Así, el usuario de un procesador puede decir que el procesador A es mejor que el procesador B cuando A ejecuta su programa en menor tiempo que B. En cambio el responsable de un centro de cálculo entenderá que A es mejor que B si es capaz de ejecutar mayor número de trabajos por unidad de tiempo. El primero estará interesado en el tiempo de respuesta (*response time*) del procesador mientras que el segundo lo estará en la productividad (*throughput*). Pero en ambos casos la clave es el tiempo: el procesador que realiza la misma cantidad de trabajo en el menor tiempo posible será el más rápido, la diferencia estriba en si medimos una tarea (tiempo de respuesta) o muchas (productividad).

**El patrón de medida más significativo es el conjunto de programas reales** que se ejecutan en los procesadores. Sin embargo aquí surge de nuevo y con más intensidad la diversidad de puntos de vista. En efecto, el usuario de un editor de texto querrá medir el rendimiento de un procesador respecto a la eficiencia para ejecutar su programa, que posiblemente no coincida con el

punto de vista del usuario de un programa de diseño gráfico. Fijar de la forma más objetiva posible los patrones o programas respecto a los cuales se mida el rendimiento de un procesador será pues una tarea polémica y siempre cuestionada por la comunidad de interesados en los resultados de la medida.

### 1.1. Tiempo de ejecución

El tiempo que tarda un programa en ser ejecutado por un computador puede ser difícil de medir, debido a los Sistemas Operativos Multitarea y a los dispositivos de E/S, que tienen tiempos de respuesta que son independientes de la frecuencia de reloj del ordenador. Por ello es necesario diferenciar entre el tiempo que tarda una CPU en ejecutar el código de un programa, el tiempo que utiliza el S.O. para realizar sus tareas, y el tiempo necesario para acceder a los dispositivos de E/S.

El tiempo de ejecución de un programa lo dividiremos en las siguientes componentes:

- *Tiempo de respuesta*
- *Tiempo de CPU*

A su vez, el tiempo de CPU lo dividimos en:

- *Tiempo de CPU utilizado por el usuario.*
- *Tiempo de CPU utilizado por el S.O.*

- ❑ **Tiempo de respuesta** Es el tiempo necesario para completar una tarea, incluyendo los accesos al disco, a la memoria, las actividades de E/S y los gastos del S.O. Es el tiempo que percibe el usuario.
- ❑ **Tiempo de CPU** Es el tiempo que tarda en ejecutarse un programa, sin tener en cuenta el tiempo de espera debido a la E/S o el tiempo utilizado para ejecutar otros programas. Se divide en:
  - ❑ **Tiempo de CPU utilizado por el usuario.** Es el tiempo que la CPU utiliza para ejecutar el programa del usuario. No se tiene en cuenta el tiempo de espera debido a la E/S o el tiempo utilizado para ejecutar otros programas
  - ❑ **Tiempo de CPU utilizado por el S.O.** Es el tiempo que el S.O. emplea para realizar su gestión interna.

La función *time* de Unix produce una salida de la forma: 90.7u 12.9s 2:39 65%, donde:

- Tiempo de CPU del usuario = 90.7 segundos
- Tiempo de CPU utilizado por el sistema = 12.9 segundos
- Tiempo de CPU = 90.7 seg.+ 12.9seg = 103.6
- Tiempo de respuesta = 2 minutos 39 segundos = 159 segundos
- Tiempo de CPU = 65% del tiempo de respuesta = 159 segundos\*0.65 = 103.6
- Tiempo esperando operaciones de E/S y/o el tiempo ejecutando otras tareas 35% del tiempo de respuesta = 159 segundos\*0.35 = 55.6 segundos

El *tiempo de respuesta* se utiliza como medida del rendimiento del sistema (con el sistema no cargado), mientras que el rendimiento de la CPU normalmente hace referencia al *tiempo de CPU del usuario* sobre un sistema no cargado.

El tiempo de CPU de un programa podemos expresarlo como:

$$\text{Tiempo\_de\_CPU} = \text{Número\_de\_ciclos\_de\_reloj\_de\_la\_CPU} \cdot T_c$$

donde  $T_c = \text{Duración\_del\_ciclo\_de\_reloj}$

$$\text{Tiempo\_de\_CPU} = \frac{\text{Número\_de\_ciclos\_de\_reloj\_de\_la\_CPU}}{F_c}$$

**donde**  $F_c = \text{Frecuencia\_de\_reloj} = 1/T_c$

Además del número de ciclos de reloj que necesita la ejecución de un programa, también se puede medir el número de instrucciones  $NI$  ejecutadas, y podremos calcular el número medio de Ciclos de reloj Por Instrucción o CPI como:

$$CPI = \frac{\text{Número\_de\_ciclos\_de\_reloj\_de\_la\_CPU}}{NI}$$

por lo que el tiempo de CPU podemos expresarlo como:

$$\text{Tiempo\_de\_CPU} = NI \cdot CPI \cdot T_c.$$

o bien:

$$\text{Tiempo\_de\_CPU} = \frac{NI \cdot CPI}{F_c}$$

- $NI$ : depende del compilador y la arquitectura utilizada, y se expresa en *instrucciones/programa*
- $CPI$ : depende de la arquitectura y estructura (organización) de la máquina, y se expresa en *ciclos de reloj/instrucción*
- $T_c$ : Depende de la estructura y la tecnología de la máquina, y se expresa en *segundos/ciclo de reloj*

En determinadas situaciones de diseño puede ser útil calcular el número total de ciclos de reloj de la CPU como:

$$\text{Número\_de\_ciclos\_de\_reloj\_de\_la\_CPU} = \sum_{i=1}^n CPI_i \cdot NI_i$$

donde  $NI_i$  representa el número de veces que el grupo de instrucciones  $i$  es ejecutado en un programa, y  $CPI_i$  representa el número medio de ciclos de reloj para el conjunto de instrucciones  $i$ . El tiempo de CPU se evalúa en este caso mediante una de las 2 siguientes expresiones:

$$\text{Tiempo\_de\_CPU} = \left( \sum_{i=1}^n CPI_i \cdot NI_i \right) \cdot T_c$$

$$\text{Tiempo\_de\_CPU} = \frac{\left( \sum_{i=1}^n CPI_i \cdot NI_i \right)}{F_c}$$

El número medio de ciclos por instrucción vendrá dado por:

$$CPI = \frac{\left( \sum_{i=1}^n CPI_i \cdot NI_i \right)}{NI} = \sum_{i=1}^n \left( CPI_i \cdot \frac{NI_i}{NI} \right)$$

Esta última expresión calcula el *CPI* multiplicando cada *CPI<sub>i</sub>* individual por la fracción de ocurrencias de las instrucciones *i* en el programa. *CPI<sub>i</sub>* debe ser medido, y no calculado a partir de la tabla del manual de referencia, ya que se deben incluir los fallos de caché y otras ineficiencias del sistema de memoria.

### Ejemplo

Se consideran 2 alternativas para implementar el salto condicional en un procesador:

**CPU A:** Una instrucción de comparación genera el código de condición, y una de salto bifurca en función del valor de código generado.

**CPU B:** Una única instrucción de salto que incluye la comparación.

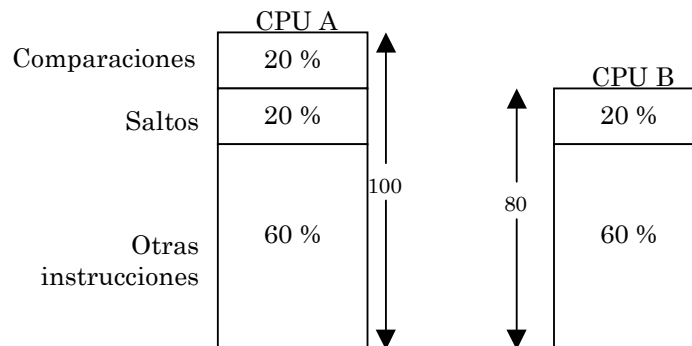
En ambos casos la instrucción de salto consume 2 ciclos de reloj, las demás instrucciones consumen 1 ciclo. Se ignoran las pérdidas debidas al sistema de memoria.

En la CPU A, el 20% de las instrucciones ejecutadas son saltos condicionales, y como en esta CPU cada salto es precedido por una comparación, otro 20% de las instrucciones ejecutadas son comparaciones.

Debido a que la CPU A no incluye la comparación en el salto, su ciclo de reloj es un 25% más rápido que el de la CPU B.

Bajo estas condiciones ¿Qué CPU es más rápida?

### Solución



$$Tc\_B / Tc\_A = 1.25 \implies Tc\_B = 1.25 \cdot Tc\_A$$

$$NI\_B = 0.8 \cdot NI\_A$$

$$Tiempo\ de\ CPU\_A = NI\_A \cdot CPI\_A \cdot Tc\_A$$

$$Tiempo\ de\ CPU\_B = NI\_B \cdot CPI\_B \cdot Tc\_B$$

$$Tiempo\ de\ CPU\_B = (0.8 \cdot NI\_A) \cdot CPI\_B \cdot (1.25 \cdot Tc\_A)$$

$$Tiempo\ de\ CPU\_B = (0.8 \cdot 1.25) \cdot NI\_A \cdot CPI\_B \cdot Tc\_A$$

$$CPI_A = (NI_{salto\_A} / NI_A) \cdot CPI_{salto} + (NI_{resto\_A} / NI_A) \cdot CPI_{resto} = (20/100) \cdot 2 + (80/100) \cdot 1 = 0.2 \cdot 2 + 0.8 \cdot 1 = 1.2$$

$$CPI_B = (NI_{salto\_B} / NI_B) \cdot CPI_{salto} + (NI_{resto\_B} / NI_B) \cdot CPI_{resto} = (20/80) \cdot 2 + (60/80) \cdot 1 = 0.25 \cdot 2 + 0.75 \cdot 1 = 1.25$$

Luego la alternativa de diseño correspondiente a la CPU A sería más rápida. La ganancia de velocidad de la alternativa A sobre la B valdrá:

$$Tiempo\ de\ CPU\_A = 1.2 \cdot NI_A \cdot Tc_A.$$

$$Tiempo\ de\ CPU\_B = 1.25 \cdot NI_A \cdot Tc_A$$

La ganancia de velocidad valdrá:

$$\frac{Tiempo\ de\ CPU\_B}{Tiempo\ de\ CPU\_A} = \frac{1.25}{1.2} = 1.0416$$

## 1.2. Otros parámetros de rendimiento

- MIPS (Millones de Instrucciones Por Segundo)

$$MIPS = \frac{NI}{Tiempo\_de\_ejecución \cdot 10^6} = \frac{1}{CPI \cdot 10^6 \cdot Tc} = \frac{Fc}{CPI \cdot 10^6}$$

$$Tiempo\_de\_ejecución = \frac{NI}{MIPS \cdot 10^6}$$

- Los MIPS dependen del repertorio de instrucciones, por lo que resulta un parámetro difícil de utilizar para comparar máquinas con diferente repertorio
- Varían entre programas ejecutados en el mismo computador
- Pueden variar inversamente al rendimiento, como ocurre en máquinas con hardware especial para punto flotante, que emplean menor tiempo que las que no disponen de este hardware y por tanto tienen mejor rendimiento. Sin embargo presentan un menor valor de los MIPS porque ejecutan menor número de instrucciones

En definitiva, los MIPS pueden fallar al dar una medida del rendimiento, puesto que no reflejan el tiempo de ejecución. Por esta razón se han venido utilizando los MIPS relativos, que miden cuantas veces más tarda en ejecutarse un programa en la máquina a medir que en una de referencia, que por definición tiene un MIPS (VAX-11/780):

$$MIPS_{relativos} = \frac{Tiempo\ de\ ejecución\ en\ la\ máquina\ de\ referencia}{Tiempo\ de\ ejecución\ en\ la\ máquina\ a\ medir} \cdot MIPS_{referencia}$$

$$MIPS_{referencia} = 1 \text{ (MIPS del VAX-11/780)}$$

- MFLOPS (Millones de operaciones en punto FLOtante Por Segundo)

$$MFLOPS = \frac{Número\ de\ operaciones\ en\ coma\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecución \cdot 10^6}$$

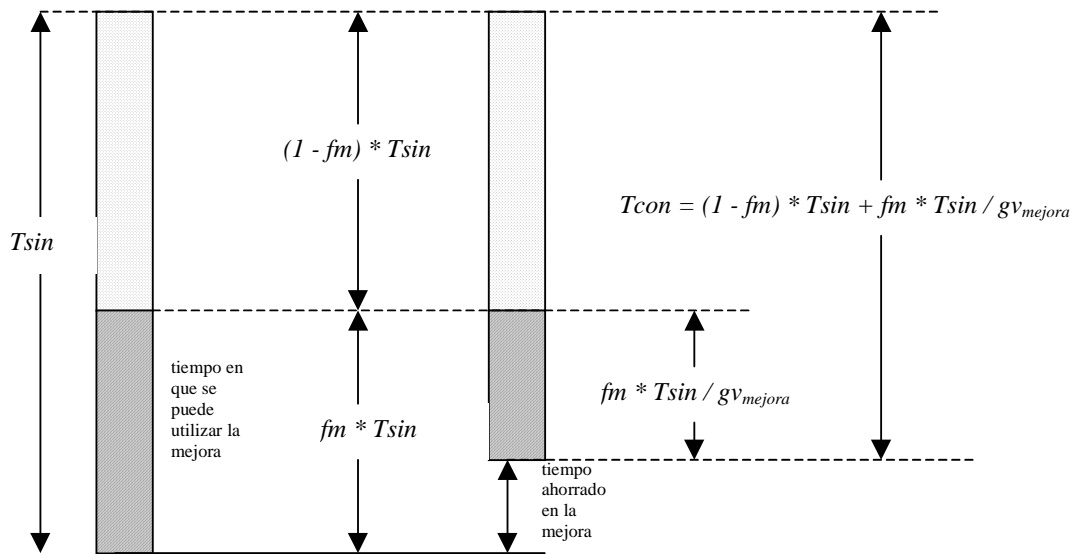
- Existen operaciones en coma flotante rápidas (como la suma) o lentas (como la división), por lo que puede resultar una medida poco significativa. Por ello se han utilizado los MFLOPS normalizados, que dan distinto peso a las diferentes operaciones. Por ejemplo: suma, resta, comparación y multiplicación se les da peso 1; división y raíz cuadrada peso 4; y exponenciación, trigonométricas, etc. peso 8.
- Productividad (*throughput*)  
Número de tareas ejecutadas en la unidad de tiempo
- Ganancia de velocidad (*speedup*): Ley de Amdahl

Para calcular el aumento de rendimiento que puede obtenerse al mejorar alguna parte de un computador se utiliza la Ley de Amdahl: *La mejora obtenida en el rendimiento global de un computador al utilizar algún modo de ejecución más rápido está limitada por la fracción de tiempo que se puede utilizar ese modo más rápido.*

La ganancia de velocidad global de un sistema se define por el siguiente cociente:

$$g^{v_{global}} = \frac{\text{Tiempo\_de\_ejecución\_sin\_mejora}(T_{sin})}{\text{Tiempo\_de\_ejecución\_con\_mejora}(T_{con})}$$

Si llamamos  $f_m$  a la fracción de tiempo que puede utilizarse el modo de ejecución más rápido, es decir, introduciendo una mejora, y  $g^{v_{mejora}}$  la ganancia de velocidad propia del elemento o modo mejorado, la ganancia de velocidad global del sistema vendrá dada por la siguiente expresión:



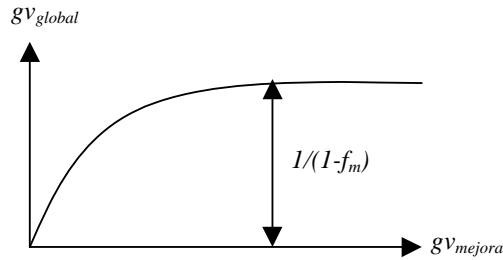
$$g^{v_{global}} = \frac{T_{sin}}{T_{con}} = \frac{T_{sin}}{(1 - f_m) * T_{sin} + \frac{f_m * T_{sin}}{g^{v_{mejora}}}} = \frac{1}{(1 - f_m) + \frac{f_m}{g^{v_{mejora}}}}$$

Esta expresión recoge la forma cuantitativa de la ley de Amdahl. Si obtenemos el límite cuando la ganancia de velocidad del modo mejorado tiende a infinito, es decir, suponemos que el modo mejora infinitamente, obtenemos lo siguiente:

$$\lim_{gV_{mejora} \rightarrow \infty} gV_{global} = \frac{1}{1 - f_m}$$

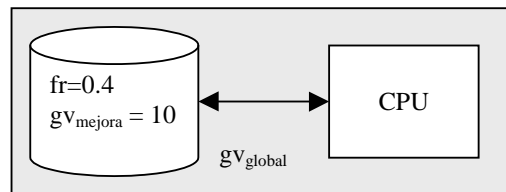
$$gV_{mejora} \rightarrow \infty$$

Es decir, la ganancia de velocidad global se mantiene limitada por una expresión de la fracción  $f_m$



### Ejemplo

Un disco magnético 10 veces más rápido que en la máquina original. El disco se utiliza sólo el 40% del tiempo de ejecución. ¿Cual es la ganancia de velocidad global?



$$gV_{mejora} = 10; f_m = 0.4 \implies gV_{global} = \frac{1}{(1-0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

$$\lim_{gV_{mejora} \rightarrow \infty} gV_{global} = \frac{1}{(1-0.4)} = 1.666 \text{ (máxima ganancia para velocidad del disco infinita)}$$

$$gV_{mejora} \rightarrow \infty$$

- Rendimiento medio armónico

Sean  $P_1, P_2, \dots, P_i, \dots, P_m$  un conjunto de programas que representan modos de funcionamiento de una carga de trabajo. Por ejemplo,  $P_1$ , puede representar el modo de funcionamiento entero,  $P_2$  el modo real, etc.

Sean  $r_1, r_2, \dots, r_i, \dots, r_m$  las velocidades de ejecución en *instrucciones/segundo* de los programas anteriores, y  $t_1, t_2, \dots, t_i, \dots, t_m$  los tiempos de ejecución medios por instrucción en segundos/instrucción:  $t_i = 1/r_i$

Se definen los siguientes parámetros:

#### Tiempo de ejecución medio aritmético

$$T_a = \frac{1}{m} \sum_{i=1}^m t_i = \frac{1}{m} \sum_{i=1}^m \frac{1}{r_i}$$

Velocidad de ejecución media armónica

$$R_h = \frac{1}{T_a} = \frac{m}{\sum_{i=1}^m \frac{1}{r_i}}$$

Tiempo de ejecución medio ponderado

$$T_a^* = \sum_{i=1}^m \frac{f_i}{r_i} \quad \text{con } f_1, f_2, \dots, f_i, \dots, f_m \text{ los pesos de los programas } P_1, P_2, \dots, P_i, \dots, P_m;$$

$$\sum_{i=1}^m f_i = 1$$

Velocidad de ejecución media armónica ponderada

$$R_h^* = \frac{1}{T_a^*} = \frac{1}{\sum_{i=1}^m \frac{f_i}{r_i}}$$

Velocidad de ejecución media aritmética

$$R_a = \frac{1}{m} \sum_{i=1}^m r_i$$

Velocidad de ejecución media aritmética ponderada

$$R_a = \sum_{i=1}^m f_i * r_i$$

Problema cuando se promedian velocidades en lugar de tiempos cuando se quiere caracterizar el comportamiento de un computador que funciona con 2 o más modos:

P<sub>1</sub> ejecuta 10.000 instrucciones en 5 segundos => r<sub>1</sub> = 2.000 instrucciones/segundo

P<sub>2</sub> ejecuta 10.000 instrucciones en 2 segundos => r<sub>2</sub> = 5.000 instrucciones/segundo



P <sub>1</sub>	P <sub>2</sub>
10.000 instrucciones	10.000 instrucciones
5 segundos	2 segundos

Velocidad de ejecución media aritmética  $R_a = (2.000 + 5.000)/2 = 3.500$  instrucciones/segundo

Si con esta velocidad media calculamos las instrucciones ejecutadas en 7 segundos =

$3.500 * 7 = 24.500$  instrucciones  $\neq 20.000$

$$T_a = 1/2(5/10.000 + 2/10.000) = 7/20.000 \Rightarrow R_h = 1/T_a = 20.000/7 = 2.857,14$$

Si con esta velocidad media armónica calculamos las instrucciones ejecutadas en 7 segundos =

$$2.857,14 * 7 = 20.000$$

**Ejemplo**

Se ejecutan 4 programas P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub> (que representan 4 modos de ejecución de un programa general) sobre 3 computadores A, B y C. Por cada programa se ejecutan 10<sup>8</sup> instrucciones, resultando los siguientes tiempos en segundos:

Tiempo de ejecución (seg)	Computador		
Programa	A	B	C
P <sub>1</sub>	1	10	20
P <sub>2</sub>	10.000	100	20
P <sub>3</sub>	500	1000	50
P <sub>4</sub>	100	800	100

¿Cuál es el computador más rápido?

Los MIPS da cada procesador para cada programa valdrán:

MIPS	Computador		
Programa	A	B	C
P <sub>1</sub>	100	10	5
P <sub>2</sub>	0,1	1	5
P <sub>3</sub>	0,2	0,1	2
P <sub>4</sub>	1	0,125	1

$$T_a = \frac{1}{m} \sum_{i=1}^m t_i = \frac{1}{m} \sum_{i=1}^m \frac{1}{r_i}$$

Tiempos de ejecución medios aritméticos

$$T_a(A) = 1/4*(1/100+1/0,1+1/0,2+1) = 16,01/4 = 4,002$$

$$T_a(B) = 1/4*(1/10+1+1/0,1+1/0,125) = 19,1/4 = 4,775$$

$$T_a(C) = 1/4*(1/5+1/5+1/2+1) = 1,9/4 = 0,475$$

Velocidades de ejecución medias armónicas

$$R_h(A) = 1/ T_a (A) = 0,25$$

$$R_h(B) = 1 / T_a(B) = 0,21$$

$$R_h(C) = 1 / T_a(C) = 2,12$$

Luego el mejor computador (más rápido) es C, seguido de A, y seguido de B

Velocidad de ejecución media geométrica

$$R_g = \prod_{i=1}^m R_i^{1/m}$$

Velocidad de ejecución media geométrica ponderada

$$R_g^* = \prod_{i=1}^m R_i^{f_i}$$

Se utiliza con medidas de rendimiento normalizadas con respecto a una máquina de referencia.

## 2. Patrones de medida (*Benchmarks*)

Para evaluar el rendimiento de un computador podemos utilizar diferentes técnicas:

- Modelos analíticos (matemáticos) de la máquina
- Modelos de simulación (algorítmicos) de la máquina
- La máquina real

Las dos primeras alternativas se utilizan cuando la máquina no está disponible. Los modelos analíticos tienen limitado su ámbito de utilización por la dificultad de expresar en forma de ecuaciones matemáticas el comportamiento detallado de la máquina y su carga de trabajo. Se utilizan en fases muy tempranas de diseño para realizar estimaciones generales del rendimiento.

Los modelos de simulación pueden construirse con mayor precisión, recogiendo especificaciones detalladas de diseño. Sin embargo, estos modelos requieren una gran capacidad computacional cuando se incorporan todos los componentes básicos de la máquina.

En la tercera alternativa es la máquina o máquinas reales las que se evalúan con el fin de disponer de algún criterio de elección. En este caso (y posiblemente en algunos modelos de simulación) será necesario disponer de un conjunto de programas representativos de la carga real de trabajo que vaya a tener la máquina, y con respecto a los cuales se realicen las medidas. Estos programas patrones se denominan *benchmarks*, y serán el objeto de estudio de este apartado.

Podemos utilizar diferentes criterios no excluyentes a la hora de clasificar los *benchmarks* que se utilizan en la actualidad para evaluar los computadores.

Comenzaremos con una clasificación general en función del ámbito de aplicación que representan, es decir, el tipo de recursos computacionales que mayoritariamente intervienen en la evaluación. En este sentido los clasificaremos en:

- **Enteros:** aplicaciones en las que domina la aritmética entera, incluyendo procedimientos de búsqueda, operaciones lógicas, etc. Por ejemplo, *SPECint2000*.
- **Punto flotante:** aplicaciones intensivas en cálculo numérico con reales. Por ejemplo, *SPECfp2000* y *LINPACK*.
- **Transacciones:** aplicaciones en las que dominan las transacciones *on-line* y *off-line* sobre bases de datos. Por ejemplo, *TPC-C*.

En segundo lugar los agruparemos por la naturaleza del programa que implementan:

- **Programas reales:** Compiladores, procesadores de texto, etc. Permiten diferentes opciones de ejecución. Con ellos se obtienen las medidas más precisas
- **Núcleos (Kernels):** Trozos de programas reales. Adecuados para analizar rendimientos

- específicos de las características de una determinada máquina: *Linpack*, *Livermore Loops*
- **Patrones conjunto** (*benchmarks suits*) Conjunto de programas que miden los diferentes modos de funcionamiento de una máquina: *SPEC* y *TPC*.
  - **Patrones reducidos** (*toy benchmarks*): Programas reducidos (10-100 líneas de código) y de resultado conocido. Son fáciles de introducir y ejecutar en cualquier máquina (*Quicksort*,...)
  - **Patrones sintéticos** (*synthetic benchmarks*): Código artificial no perteneciente a ningún programa de usuario y que se utiliza para determinar perfiles de ejecución. (*Whetstone*, *Dhrystone*)

A continuación estudiaremos con algún detalle tres de los *benchmarks* con mayor difusión.

## 2.1. LINPACK

Es una colección de subrutinas Fortran que analizan y resuelven ecuaciones lineales y problemas de mínimos cuadrados. Los sistemas de ecuaciones lineales que contempla utilizan diferentes formas de las matrices: generales, en banda, simétricas, triangulares etc. Se diseñó para medir el rendimiento en supercomputadores a finales de los 70 y principio de los 80 (Dongarra). Lo componen las siguientes subrutinas:

- matrices generales y en banda
- matrices definidas positivas
- matrices en banda definidas positivas
- matrices indefinidas simétricas
- matrices triangulares y tridiagonales
- descomposición de *Cholesky*
- descomposición QR
- actualización de las descomposiciones QR y *Cholesky*
- descomposición valores singulares

La tabla siguiente muestra los resultados de este *benchmark* para algunos computadores corriendo bajo un sistema operativo y utilizando un compilador concreto. La primera columna numérica presenta el resultado en *Mflops/segundo*, es decir, en millones de operaciones en punto flotante por segundo para una matriz de orden 100. La segunda para una matriz de orden 1000, y la tercera presenta el rendimiento de pico de la máquina. Para resolver un sistema de  $n$  ecuaciones se realizan  $2/3n^3 + 2n^2$ .

Computador	OS/Compilador	N=100 Mflops/s	N=1000 Mflops/s	Peak Mflops/s
Cray T916 (8 proc. 2.2 ns)	cf77 (6.0) -Zp -Wd-68		10800	14400
Cray T916 (4 proc. 2.2 ns)	cf77 (6.0) -Zp -Wd-68		5711	7200
Cray T916 (2 proc. 2.2 ns)	cf77 (6.0) -Zp -Wd-68		2943	3600
Cray T916 (1 proc. 2.2 ns)	cf77 (6.0) -Zp -Wd-68	522	1576	1800
Cray C90 (16 proc. 4.2 ns)	CF77 5.0 -Zp -Wd-e68	479	10780	15238
Cray C90 (8 proc. 4.2 ns)	CF77 5.0 -Zp -Wd-e68	468	6175	7619
Cray 3-128 (4 proc. 2.11 ns)	CSOS 1.0 level 129	421	2862	3792
Hitachi S-3800/480(4 proc 2 ns)			20640	32000
Hitachi S-3800/380(3 proc 2 ns)			16880	24000
Hitachi S-3800/280(2 proc 2			12190	16000
Hitachi S-3800/180(1 proc 2 ns)	OSF/1 MJ FORTRAN:V03-00	408	6431	8000
Cray 3-128 (2 proc. 2.11 ns)	CSOS 1.0 level 129	393	1622	1896
Cray C90 (4 proc. 4.2 ns)	CF77 5.0 -Zp -Wd-e68	388	3275	3810
Cray C90 (2 proc. 4.2 ns)	CF77 5.0 -Zp -Wd-e68	387	1703	1905
Cray C90 (1 proc. 4.2 ns)	CF77 5.0 -Zp -Wd-e68	387	902	952
NEC SX-3/44R (4 proc. 2.5 ns)			15120	25600
NEC SX-3/42R (4 proc. 2.5 ns)			8950	12800
NEC SX-3/41R (4 proc. 2.5 ns)			4815	6400
NEC SX-3/34R (3 proc. 2.5 ns)			12730	19200
NEC SX-3/32R (3 proc. 2.5 ns)			6718	9600

NEC SX-3/31R (3 proc. 2.5 ns)			3638	4800
NEC SX-3/24R (2 proc. 2.5 ns)			9454	12800
NEC SX-3/22R (2 proc. 2.5 ns)			5116	6400
NEC SX-3/21R (2 proc. 2.5 ns)			2627	3200
NEC SX-3/14R (1 proc. 2.5 ns)	f77sx 040 R2.2 -pi**:	368	5199	6400
NEC SX-3/12R (1 proc. 2.5 ns)	f77sx 040 R2.2 -pi**:	368	2757	3200
Cray 3-128 (1 proc. 2.11 ns)	CSOS 1.0 level 129	327	876	948
NEC SX-3/44 (4 proc. 2.9 ns)			13420	22000
NEC SX-3/24 (2 proc. 2.9 ns)			8149	11000
NEC SX-3/42 (4 proc. 2.9 ns)			7752	11000
NEC SX-3/22 (2 proc. 2.9 ns)			4404	5500
NEC SX-3/14 (1 proc. 2.9 ns)	f77sx 020 R1.13 -pi**:	314	4511	5500
NEC SX-3/12 (1 proc. 2.9 ns):	f77sx 020 R1.13 -pi**:	313	2283	2750
Cray Y-MP/832 (8 proc. 6 ns)	CF77 4.0 -Zp -Wd-e68	275	2144	2667
Fujitsu VP2600/10 (3.2 ns)	FORTRAN77 EX/VP V11L10	249	4009	5000
Cray Y-MP/832 (4 proc. 6 ns)	CF77 4.0 -Zp -Wd-e68	226	1159	1333
Fujitsu VPP500/1(1 proc. 10 ns)	FORTRAN77EX/VP V12L20	206	1490	1600
Cray Y-MP M98 (8 proc. 6 ns)	CF77 5.0 -Zp -Wd-e68	204	1733	2666
Fujitsu VP2200/10 (3.2 ns)	FORTRAN77 EX/VP V12L10	203	1048	1250
Cray 2S/4-128 (4 proc. 4.1 ns)	CSOS 1.0 level 129	202	1406	1951

## 2.2. SPEC: (System Performance and Evaluation Cooperative)

SPEC es una sociedad sin ánimo de lucro cuya misión es establecer, mantener y distribuir un conjunto estandarizado de *benchmarks* que se pueden aplicar a las últimas generaciones de procesadores. Se han sucedido ya 5 generaciones de *benchmarks*: SPEC89, SPEC92, SPEC95, SPEC2000 y SPEC2006. En lo que sigue nos limitaremos a la SPEC2000 y la última en vigor.

- SPEC2000 (2006)

### 2.2.1.1. SPEC CPU2000 (2006)

Los diseña el *OSG (Open Systema Group)* de *SPEC* que es el encargado de los *benchmarks* de componentes y sistemas. En este apartado se encuentran los *SPECint2000 (2006)* y *SPECfp2000 (2006)*, representativos de las aplicaciones enteras y reales (punto flotante). Dentro de *SPEC* existen otros dos grupos *HPG (High Performance Group)* y *GPCG (Graphics Performance Characterization Group)*.

La máquina de referencia de los *SPEC CPU2000* y *SPEC CPU2006* es la *UltraSPARC10* con un procesador *UltraSPARC Iii*, a 300 MHz y 256 MB de memoria. Esto significa que todos los resultados se calculan como *ratios* frente a la máquina de referencia, que por definición tiene *SPECint2000* = 100, y *SPECfp2000* = 100.

### 2.2.1.2. SPECint2000

Lo integran los siguientes programas:

- **gzip**: programa de compresión de datos que utiliza el algoritmo de *Lempel-Ziv (LZ77)*.
- **vpr**(*versatil place and route*): implementa los procesos de ubicación (*placement*) y conexionado (*routing*) de un circuito digital sobre bloques lógicos de *FPGAs*. Pertenece a la clase de programas de optimización combinatoria. Utiliza *simulated annealing* como procedimiento de optimización.
- **gcc**: compilador de C que genera código para el procesador Motorola 88100.
- **mcf**: Programa de planificación temporal (*scheduling*) de transporte público que utiliza el código *MFC* del método *simplex* de red (versión especializada del algoritmo *simplex* de programación lineal).
- **crafty**: programa de juego de ajedrez por computador con un número significativo de operaciones enteras y lógicas tales como *AND*, *OR*, *XOR* y desplazamientos.
- **parser**: analizador sintáctico de inglés basado en gramáticas de enlace (*link grammar*) con un diccionario de mas de 60.000 palabras.
- **eon**: trazador de rayos (*ray tracer*) que transmite un número de líneas 3D (rayos) sobre

un modelo poligonal 3D, calcula la intersección entre las líneas y los polígonos, y genera nuevas líneas para calcular la luz incidente en los puntos de intersección. El resultado final es una imagen vista por cámara.

- ❑ **perlbnk**: versión del lenguaje de *script Perl* en el que se han eliminado las características específicas del SO.
- ❑ **gap**: implementa un lenguaje diseñado para computar en grupos (*GAP: Groups, Algorithm and Programming*).
- ❑ **vortex**: procede de un *OODBMS* (sistema de gestión de bases de datos orientado a objetos) que se ha adaptado para conformarlo a las exigencias de SPEC2000.
- ❑ **bzip2**: basado en la versión 0.1 de bzipb.
- ❑ **twolf**: paquete de ubicación (*placement*) y conexionado (*routing*) en el proceso de diseño de circuitos integrados basados en celdas standard. Utiliza el algoritmo de *simulated annealing*.

<b>CINT2000 Result</b>					
Copyright © 1999-2002 Standard Performance Evaluation Corporation					
	<b>Intel Corporation</b>	=		SPECint2000	<b>35</b>
	Intel D850MD motherboard ( <b>2.0A GHz, Pentium 4 processor</b> )			SPECint_base 2000 =	<b>22</b>
SPEC license # 13	Tested by: Intel Corporation	Test date: Nov-2001	Hardware Jan-2002	Avail: Software Avail: Oct-2001	

Benchmark	Reference Time	Base Runtime	Base Ratio	Runtime	Ratio	0 185 370 555 740 925 1110 1295
164.gzip	1400	184	762	182	767	
175.vpr	1400	316	443	315	444	
176.gcc	1100	128	862	128	861	
181.mcf	1800	326	552	326	553	
186.crafty	1000	147	681	146	686	
197.parser	1800	249	723	250	721	
252.eon	1300	142	917	122	1066	
253.perlbnk	1800	209	860	209	860	
254.gap	1100	116	944	116	945	

255.vortex	1900	162	1171	159	1192	
256.bzip2	1500	259	580	254	591	
300.twolf	3000	596	503	594	505	
SPECint_base2000			722			
			SPECint2000	735		

<h3>CINT2000 Result</h3> <p>Copyright © 1999-2000 Standard Performance Evaluation Corporation</p>
---

<b>Intel Corporation</b> <b>Intel VC820 (1.0 GHz Pentium III)</b>	SPECint2000 = <b>10</b> SPECint_base 2000 = <b>07</b>
--	--

SPEC license # 13	Tested by: Intel Corporation	Test date: Mar-2000	Hardware Avail: Mar-2000	Software Avail:
-------------------	------------------------------	---------------------	--------------------------	-----------------

Benchmark	Reference Time	Base Runtime	Base Ratio	Runtime	Ratio	
164.gzip	1400	288	486	287	488	
175.vpr	1400	426	329	426	329	
176.gcc	1100	297	371	295	373	
181.mcf	1800	710	254	710	254	
186.crafty	1000	200	499	200	499	
197.parser	1800	523	344	523	344	
252.eon	1300	320	406	320	406	
253.perlbnk	1800	312	576	313	576	
254.gap	1100	290	379	289	381	
255.vortex	1900	272	697	266	714	

256.bzip2	1500	413	363	398	377	
300.twolf	3000	844	355	820	366	
SPECint_base2000			407			
SPECint2000				410		

2.2.1.3. SPECint2006

400.perlbench	C	PERL Programming Language
401.bzip2	C	Compression
403.gcc	C	C Compiler
429.mcf	C	Combinatorial Optimization
445.gobmk	C	Artificial Intelligence: go
456.hmmmer	C	Search Gene Sequence
458.sjeng	C	Artificial Intelligence: chess
462.libquantum	C	Physics: Quantum Computing
464.h264ref	C	Video Compression
471.omnetpp	C++	Discrete Event Simulation
473.astar	C++	Path-finding Algorithms
483.xalancbmk	C++	XML Processing

$$SPECint2006 = \sqrt[12]{\prod_{i=400.perlbench}^{483.xalancbmk} ratio_i} \quad ratio_i = \frac{time_{reference}}{time_{procesador}} \times 100$$

$time_{reference}$  = tiempo de ejecución del benchmark en la Máquina de referencia

$time_{procesador}$  = tiempo de ejecución del benchmark en el procesador a medir

4 Versiones	peak	base
speed	<i>SPECint2006</i>	<i>SPECint_base2006</i>
throughput	<i>SPECint_rate2006</i>	<i>SPECint_rate_base2006</i>

2.2.1.4. SPECfp2000

Lo integran los siguientes programas:

- ❑ **wupwise**: cromodinámica cuántica
- ❑ **swim**: modelado del agua superficial
- ❑ **mgrid**: resolutor multi-malla: campo potencial 3D
- ❑ **applu**: ecuaciones diferenciales parciales elípticas/parabólicas

- ❑ **mesa**: librería gráfica 3D
- ❑ **galgel**: dinámica de fluidos
- ❑ **art**: reconocimiento de imágenes/redes neuronales
- ❑ **equake**: simulación de propagación de ondas sísmicas
- ❑ **facerec**: procesamiento de imagen: reconocimiento del rostro
- ❑ **ammp**: química computacional
- ❑ **lucas**: teoría de números/prueba de números primos
- ❑ **fma3d**: simulación de choques por elementos finitos
- ❑ **sixtrack**: diseño de aceleradores de física nuclear de alta energía
- ❑ **apsi**: meteorología: distribución de contaminantes

*Tiempo en una  
UltraSPARC10 con procesador  
UltraSPARCIII, 300 MHz y 256  
MB de memoria*

<b>CFP2000 Result</b>				
Copyright © 1999-2002 Standard Performance Evaluation Corporation				
Intel Corporation Intel D850EMV2 motherboard (2.0A GHz, Pentium 4 Processor)			SPECfp2000 = <b>73</b> SPECfp_base2000 = <b>764</b>	
SPEC license # 13	Tested by: Intel Corporation	Test date: May-2002	Hardware Avail: May-2002	Software Avail: Apr-2002

Benchmark	Reference Time	Base Runtime	Base Ratio	Runtime	Ratio	
168.wupwise	1600	168	952	167	957	
171.swim	3100	235	1317	233	1333	
172.mgrid	1800	246	730	245	736	
173.applu	2100	263	798	258	815	
177.mesa	1400	183	767	182	769	
178.galgel	2900	268	1084	265	1094	
179.art	2600	489	532	484	538	
183.equake	1300	144	905	137	950	
187.facerec	1900	201	947	200	951	
188.ammp	2200	434	507	431	511	
189.lucas	2000	189	1057	189	1057	



191.fma3d	2100	282	746	281	746	████████████████████
200.sixtrack	1100	302	365	293	376	██████████
301.apsi	2600	454	572	454	573	████████████████████
SPECfp_base2000			764			
			SPECfp2000	773		

2.2.1.5. SPECfp2006

410.bwaves	Fortran	Fluid Dynamics
416.gamess	Fortran	Quantum Chemistry
433.milc	C	Physics: Quantum Chromodynamics
434.zeusmp	Fortran	Physics/CFD
435.gromacs	C/Fortran	Biochemistry/Molecular Dynamics
436.cactusADM	C/Fortran	Physics/General Relativity
437.leslie3d	Fortran	Fluid Dynamics
444.namd	C++	Biology/Molecular Dynamics
447.dealII	C++	Finite Element Analysis
450.soplex	C++	Linear Programming, Optimization
453.povray	C++	Image Ray-tracing
454.calculix	C/Fortran	Structural Mechanics
459.GemsFDTD	Fortran	Computational Electromagnetics
465.tonto	Fortran	Quantum Chemistry
470.lbm	C	Fluid Dynamics
481.wrf	C/Fortran	Weather Prediction
482.sphinx3	C	Speech recognition

$$SPECfp2006 = \sqrt[17]{\prod_{i=410.bwaves}^{482.sphinx} ratio_i} \quad ratio_i = \frac{time_{reference}}{time_{procesador}} \times 100$$

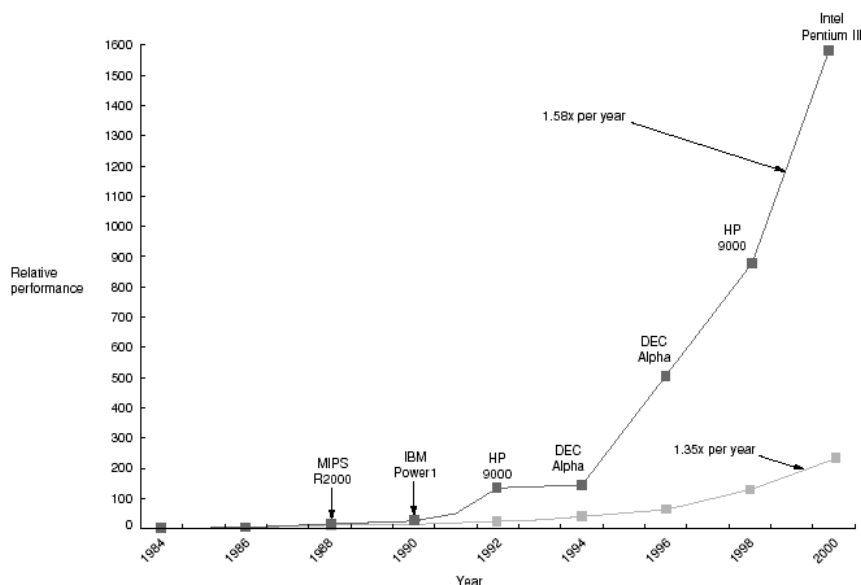
$time_{reference}$  = tiempo de ejecución del benchmark en la Máquina de referencia

$time_{procesador}$  = tiempo de ejecución del benchmark en el procesador a **medir**

4 Versiones	peak	base
speed	<i>SPECfp2006</i>	<i>SPECfp_base2006</i>

throughput	<i>SPECfp_rate2006</i>	<i>SPECfp_rate_base2006</i>
------------	------------------------	-----------------------------

La siguiente gráfica muestra la evolución de los procesadores en los últimos años caracterizados por su valor de SPECint con referencia en el VAX 11/780



### 2.3. TPC (Transaction Processing Performance Council)

TPC es un consorcio de 47 fabricantes de software y hardware entre los que se encuentran *Compaq, Digital, IBM, NCR, Sun, Informix, Intel, Microsoft y Oracle* dedicado al diseño de benchmarks para la medida de rendimiento de los sistemas informáticos. Estos *benchmarks* analizan la mayoría de los aspectos del sistema en relación con el procesamiento de transacciones, es decir, accesos de consulta y actualización a bases de datos. Tiene el inconveniente de la cantidad de tiempo que requieren las pruebas (meses) haciendo que su coste sea elevado (miles de euros). Los *benchmarks* TPC-A, TPC-B y TPC-C ya están en desuso. TPC-D fue sustituido en abril de 1999 por TPC-H y TPC-R.

- ❑ **TPC-H**: es un *benchmark* de soporte a la decisión. Consta de un conjunto preguntas (*queries*) específicas, orientadas a la actividad comercial.
- ❑ **TPC-R**: es similar a *TPC-H* pero permite optimizaciones adicionales basadas en el conocimiento de las preguntas.

Además, TPC ha introducido un *benchmark* para medir las transacciones en la WEB:

- ❑ **TPC-W**: La carga de trabajo se configura en un entorno de comercio que simula las actividades de un servidor de *WEB* orientado a las transacciones comerciales.

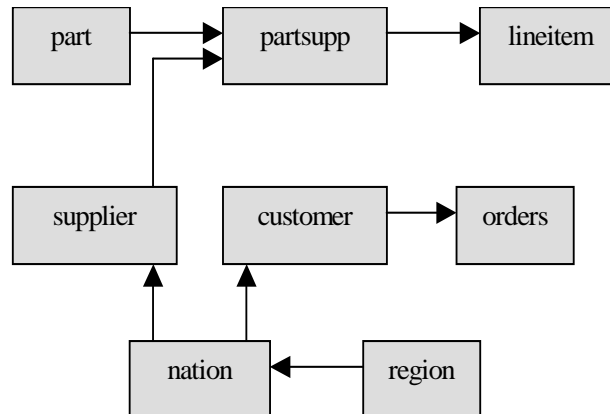
Estudiaremos con más detalle en los siguientes párrafos los actuales TPC-R y TPC-H.

#### TPC-R y TPC-H

Los benchmarks TPC-R y TPC-H fueron introducidos para sustituir al TPC-D como estándar industrial para aplicaciones comerciales. Los dos son muy similares ya que modelan la misma aplicación comercial (base de datos, preguntas y funciones de refresco). Se diferencian en las reglas de implementación, ya que mientras TPC-H limita el uso de índices y esquemas de particiones, TPC-R no limita el uso de estas estructuras en la base de datos.

- Base de Datos

La base de datos de TPC-H y TPC-R utiliza un esquema en la 3ª forma normal y responde al diagrama relación-entidad de la siguiente figura (8 tablas):



TPC dispone de un generador de datos para todas las tablas de la base dependiendo del factor de escala SF. El factor de escala determina el tamaño de los datos de la base, por ejemplo, SF=100 significa que la suma de todas las tablas de la base es igual a 100 GB. Las dos tablas mayores son *lineitem* y *orders* que contienen el 83% de los datos. El tamaño de todas las tablas excepto *nation* y *region* tienen un tamaño proporcional al factor de escala.

- Carga de Trabajo

Las cargas de trabajo para ambos *benchmarks* constan de los siguientes componentes:

- Construcción de la base de datos
- Ejecución de 22 preguntas de solo-lectura en modo monousuario y multiusuario
- Ejecución de 2 funciones de refresco.

La construcción de la base de datos incluye todos los tiempos de respuesta para crear las tablas, cargar los datos, crear índices, definir y validar restricciones, etc.

Las 22 preguntas están definidas en SQL-92 como plantillas (templates), por lo que antes de ser ejecutada sobre la base de datos se tiene que realizar la sustitución de parámetros. Se han elegido para mostrar la capacidad del sistema utilizando todos los recursos en contextos monousuario y multiusuario.

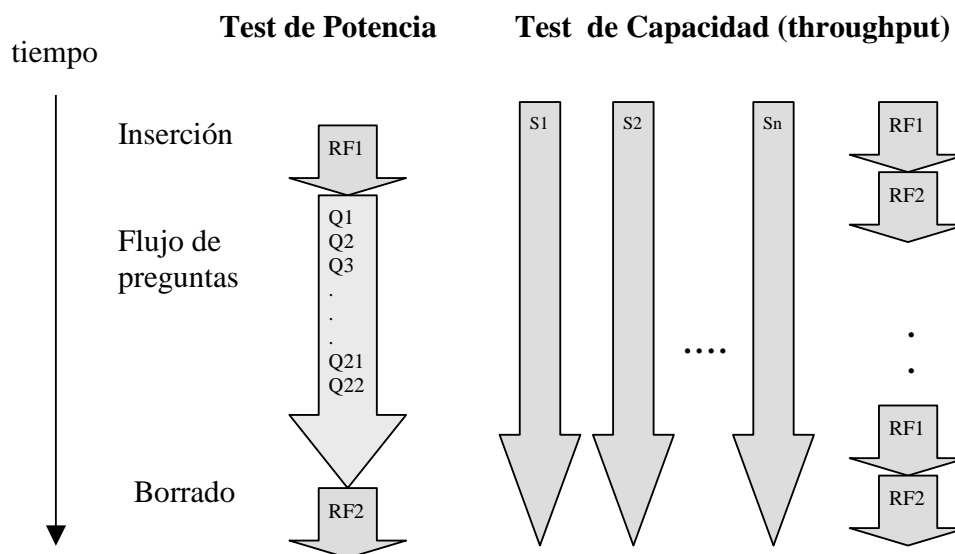
Las dos funciones de refresco (RF1 y RF2) modelan la carga de nueva información (RF1) y la eliminación de información obsoleta (RF2). RF1 inserta nuevas filas en la tabla *lineitem* y *orders*, mientras que RF2 elimina el mismo número de filas de las mismas tablas.

- Métrica de rendimiento

La principal métrica de rendimiento de TPC es la métrica de rendimiento compuesta (QphH/QphR). Para calcular la métrica compuesta de un sistema para un factor de escala dado, hay que ejecutar un *test* de potencia seguido por un *test* de capacidad de procesamiento (*throughput*). Después se combinan los resultados de ambos *tests* para calcular la métrica compuesta.

En un *test* de potencia se ejecuta la función de refresco RF1, seguida por la ejecución de las 22 preguntas en modo de único usuario, y por la ejecución de la función de refresco RF2.

En el *test* de capacidad se ejecutan las 22 preguntas (flujos de preguntas Si) en orden predefinido en múltiples sesiones concurrentes. Además en una sesión separada se ejecuta secuencialmente un par de actualizaciones (RF1, RF2) por cada flujo de preguntas.



El número mínimo de flujos depende del factor de escala, según la tabla siguiente:

Factor de Escala (SF)	Flujos (S)
1	2
10	3
100	4
300	6
1000	7
3000	8
10000	9

Los resultados de los test de potencia y capacidad se utilizan para calcular la potencia de procesamiento ([Power@Size](#)) y la capacidad de procesamiento ([Throughput@Size](#))

La **potencia de procesamiento** se calcula como la media geométrica de los tiempos de respuesta correspondientes a las preguntas y funciones de refresco. La unidad es preguntas/hora. La media geométrica reduce los efectos de las preguntas con tiempos de respuesta muy cortos y muy largos. Para un factor de escala SF dado, la potencia de procesamiento se calcula como:

$$Power @ Size = \frac{3600 * SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0) * \prod_{j=1}^{j=2} RI(j,0)}}$$

- $i=1,2,...,22;j=1,2$
- $QI(i,0)$ : tiempo de respuesta en segundos de la query  $Q_i$  del test de potencia (power)
- $RI(j,0)$ : tiempo de respuesta en segundos de la función de refresco  $RF_j$
- 3600: segundos (= 1 hora)

La **capacidad de procesamiento** se calcula como la razón del número total de preguntas ejecutadas a lo largo del intervalo de ejecución. La unidad es preguntas/hora. Para un factor de escala SF viene dada por:


$$\text{Throughput @ Size} = \frac{S * 22 * 3600}{T_s} * SF$$


- $T_s$ : tiempo de respuesta de la ejecución multi flujo
- $S$ : número de flujos
- 3600: segundos (= 1 hora)
- 22: número de preguntas por flujo

La métrica de rendimiento compuesta (QphH para TPC-H y QphR para TPC-R) se calcula como:


$$QphH @ Size, QphR @ Size = \sqrt{\text{Power @ Size} * \text{Throughput @ Size}}$$







TPC-H y TPC-R también definen una métrica precio/rendimiento como la división del precio total del sistema por el resultado de la métrica compuesta.

100 GB Results									
Rank	Company	System	QphH	Price/QphH	System Availability	Database	Operating System	Date Submitted	Cluster
1		<a href="#">HP AlphaServer ES45 Model 68/1000</a>	5,578	404 US \$	07/15/02	Oracle 9iR2 w/Real Application Cluste	HP Tru64 Unix V5.1A/IPK	07/15/02	\
2		<a href="#">IBM eServer x350 with DB2 UDB</a>	2,960	336 US \$	06/20/02	IBM DB2 UDB 7.2	Turbolinux 7 Servers	02/01/02	\
3		<a href="#">SGI 1450 Server with DB2 UDB EEE v7.2</a>	2,733	347 US \$	10/31/01	IBM DB2 UDB EEE 7.2	Linux 2.4.3	05/11/01	\
4		<a href="#">HP Proliant DL760 X900</a>	1,933	89 US \$	12/31/02	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows .NET Enterprise Server	07/31/02	↑
5		<a href="#">ProLiant 8000-X700-8P</a>	1,699	161 US \$	08/01/00	Microsoft SQL Server 2000	Microsoft Windows 2000	07/21/00	↑
6		<a href="#">HP Proliant DL580 G2</a>	1,695	82 US \$	06/26/02	Microsoft SQL Server 2000 Enterprise Edition	Microsoft Windows 2000 Advanced Server	06/26/02	↑
7		<a href="#">e-@ction Enterprise Server ES5085R</a>	1,669	169 US \$	01/31/01	Microsoft SQL Server 2000	Microsoft Windows 2000	12/22/00	↑
8		<a href="#">Netfinity 8500R</a>	1,454	200 US \$	08/01/00	Microsoft SQL Server 2000	Microsoft Windows 2000	05/23/00	↑
9		<a href="#">ProLiant 8000-8P</a>	1,308	174 US \$	08/01/00	Microsoft SQL Server 2000	Microsoft Windows 2000	04/05/00	↑






10		<a href="#">NetServer LXr8500</a>	1,291	195 US \$	08/18/00	Microsoft SQL Server 2000	Microsoft Windows 2000	08/18/00	
----	---	-----------------------------------	-------	-----------	----------	---------------------------	------------------------	----------	--


300 GB Results									
Rank	Company	System	QphH	Price/QphH	System Availability	Database	Operating System	Date Submitted	Cluster
		<a href="#">Compaq ProLiant DL760 x900-64P</a>	12,995	199 US \$	06/20/02	IBM DB2 UDB 7.2	Microsoft Windows 2000 Advanced Server	04/09/02	
		<a href="#">NUMA-Q 2000</a>	7,334	616 US \$	08/15/00	IBM DB2 UDB 7.1	DYNIX/ptx 4.5.1	05/03/00	
		<a href="#">Compaq AlphaServer ES45 Model 68/1000</a>	5,976	453 US \$	06/01/02	Oracle 9i R2 Enterprise Edition	Compaq Tru64 Unix V5.1A/IPK	05/05/02	
		<a href="#">NUMA-Q 2000</a>	5,923	653 US \$	09/05/00	IBM DB2 UDB 7.1	DYNIX/ptx 4.5.1	09/05/00	
		<a href="#">Unisys ES7000 Orion 130 Enterprise Server</a>	4,774	219 US \$	03/31/03	Microsoft SQL Server 2000 Enterprise Ed. 64-bit	Microsoft Windows .NET Server 2003 Datacenter Edt.	10/17/02	
		<a href="#">NUMA-Q 2000</a>	4,027	652 US \$	09/05/00	IBM DB2 UDB 7.1	DYNIX/ptx 4.5.1	09/05/00	
		<a href="#">AlphaServer ES40 Model 6/667</a>	2,832	1,058 US \$	02/14/01	Informix XPS 8.31 FD1	Compaq Tru64 UNIX V5.1	12/19/00	
		<a href="#">ProLiant 8000-8P</a>	1,506	280 US \$	11/17/00	Microsoft SQL Server 2000	Microsoft Windows 2000	11/17/00	
		<a href="#">NetServer LXr8500</a>	1,402	207 US \$	08/18/00	Microsoft SQL Server 2000	Microsoft Windows 2000	08/18/00	


1,000 GB Results									
Rank	Company	System	QphH	Price/QphH	System Availability	Database	Operating System	Date Submitted	Cluster
1		<a href="#">HP 9000 Superdome Enterprise</a>	25,805	231 US \$	10/30/02	Oracle Database Enterprise Edition v9.2.0.2.0	9i HP UX 11.i 64-bit	06/24/02	



		<a href="#">Server</a>							
2		<a href="#">Compaq ProLiant DL760 X900-128P</a>	22,361	255 US \$	06/20/02	IBM DB2 UDB 7.2	Microsoft Windows 2000 Advanced Server	02/06/02	\
3		<a href="#">Sun Fire[TM] 15K server</a>	18,802	256 US \$	07/17/02	Oracle Database Enterprise Edition	9i Sun Solaris 8	01/17/02	f
4		<a href="#">WorldMark 5250</a>	18,542	638 US \$	07/27/01	Teradata V2R4.1	MP-RAS 3.02.00	10/09/01	\
5		<a href="#">HP 9000 Superdome Enterprise Server</a>	13,160	564 US \$	09/05/01	Oracle 9i.9.0.1 Enterprise Edi	HP UX 11.i 64-bit	08/06/01	f
6		<a href="#">IBM RS/6000 SP 550</a>	12,866	649 US \$	08/15/00	IBM DB2 UDB 7.1	IBM AIX 4.3.3	06/15/01	\
7		<a href="#">HP 9000 Superdome Enterprise Server</a>	9,754	814 US \$	02/13/01	Informix Extended Parallel Ser 8.31FD1	HP UX 11.i 64-bit	02/13/01	f
8		<a href="#">Sun Fire 6800</a>	4,735	581 US \$	10/31/01	IBM DB2 UDB EEE 7.2	Sun Solaris 8	06/11/01	f

### 3000 GB Results

Rank	Company	System	QphH	Price/QphH	System Availability	Database	Operating System	Date Submitted	Cluster
1		<a href="#">HP 9000 Superdome Enterprise Server</a>	27,094	240 US \$	10/30/02	Oracle Database Enterprise Edition v9.2.0.2.0	9i HP UX 11.i 64-bit	08/26/02	N
2		<a href="#">Sun Fire[TM] 15K Server with Oracle9i R2</a>	23,813	237 US \$	10/30/02	Oracle 9i R2 Enterprise Edition	Sun Solaris 9	06/26/02	N
3		<a href="#">Compaq ProLiant DL760 X900-128P</a>	21,053	291 US \$	06/20/02	IBM DB2 UDB 7.2	Microsoft Windows 2000 Advanced Server	02/06/02	Y
4		<a href="#">WorldMark 5250</a>	18,803	989 US \$	07/27/01	Teradata V2R4.1	MP-RAS 3.02.00	10/09/01	Y
5		<a href="#">HP 9000 Superdome Enterprise</a>	17,908	569 US \$	05/15/02	Oracle Database Enterprise	9i HP UX 11.i 64-bit	12/17/01	N

		<a href="#">Server</a>				Edition			
6		<a href="#">Sun Starfire Enterprise 10000 with Oracle9i</a>	10,764	1,250 US \$	06/19/01	Oracle9i Database Ent. Edition 9.0.1	Sun Solaris 8	04/13/01	Y

100 GB Results							
^ Company	System	QphR	Price/QphR	System Availability	Database	Date Submitted	
	<a href="#">PowerEdge 6600/1.6 GHz Xeon MP</a>	4,452	41 US \$	04/04/03	Oracle 9i R2 Enterprise Edition	10/11/02	

1,000 GB Results							
^ Company	System	QphR	Price/QphR	System Availability	Database	Date Submitted	
	<a href="#">WorldMark 5200</a>	17,529	737 US \$	02/15/00	NCR Teradata V2R3.0	08/23/99	
	<a href="#">WorldMark 5200</a>	21,254	607 US \$	08/31/00	NCR Teradata V2R4.0	03/06/00	

#### 2.4. Otros benchmarks

- Whetstone:

*Benchmark* sintético representativo de la programación numérica escalar (punto flotante). Está formado por un conjunto de bucles con una alta localidad de referencia, lo que favorece a las cachés pequeñas. Utiliza muchas llamadas a librería, lo que posibilita mejorar los resultados optimizando estas librerías. El resultado se expresa en *whetstone/segundo*, es decir, instrucciones del *benchmark/segundo*. Se basa en una estadística recogida por Brian Wichmann del National Physical Laboratory de Inglaterra, utilizando un compilador Algo 60 que traducía Algo a instrucciones de la máquina imaginaria *Whetstone*, una pequeña ciudad a las afueras de Leicester.

- Dhystone:

*Benchmark* sintético que utiliza un juego de palabras relativas a su antecesor *whetstone* y es representativo de programas generales (enteros) escritos originariamente en Fortran. Reúne operaciones aritméticas de enteros, tratamiento de caracteres y punteros. Está constituido por un bucle de gran tamaño. El resultado se expresa en *dhystone/segundo*, es decir, número de ejecuciones del bucle por segundo. En la actualidad está también disponible en ADA, Pascal y C.

### 3. Influencia en el rendimiento de las alternativas de diseño

Revisaremos en este apartado las alternativas de diseño de los repertorios de instrucciones (ISA) estudiadas en los temas anteriores con el objeto de seleccionar aquellas que aportan mayor rendimiento al procesador. En muchos casos las alternativas se contemplan junto con datos reales de su presencia en el código máquina que generan los compiladores para programas reales. El resultado será la caracterización de un repertorio (ISA) que definirá las propiedades generales de los procesadores de tipo RISC. Nosotros proyectaremos estas propiedades sobre un procesador



hipotético denominado DLX, que es un compendio de las principales características de los actuales procesadores RISC: MIPS, Power PC, Precision Architecture, SPARC y Alpha.

### 3.1. 2.1 Tipo de elementos de memoria en la CPU

Tres alternativas:

Tipo de máquina	Ventajas	Inconvenientes
Pila		
Acumulador	Instrucciones cortas	Elevado tráfico con Memoria
Registros	Mayor flexibilidad para los compiladores Más velocidad (sin acceso a Memoria)	Instrucciones más largas

El tráfico con memoria es uno de los cuellos de botella más importantes para el funcionamiento del procesador. Se disminuye este tráfico con instrucciones que operen sobre registros, ya que el acceso es más rápido y las referencias a los registros se codifican con menor número de bits que las referencias a memoria (instrucciones más cortas).

#### Conclusión

- Se opta por el tipo de máquina con registros de propósito general (RPG)

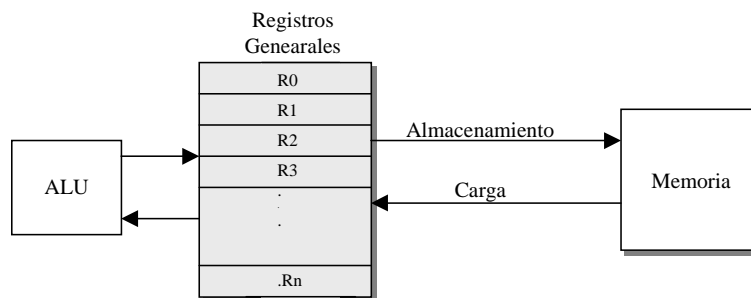
### 3.2. Referencias a memoria en instrucciones ALU

Tres alternativas:

Tipo de máquina	Ventajas	Inconvenientes
<b>Registro-Registro</b>	<ul style="list-style-type: none"> <li>• Ninguna referencia a Memoria</li> <li>• Codificación fija =&gt; formato simple</li> <li>• Generación de código simple</li> </ul>	<ul style="list-style-type: none"> <li>• Mayor número de instrucciones por programa</li> </ul>
<b>Registro-Memoria</b>	<ul style="list-style-type: none"> <li>• Menor número de instrucciones</li> </ul>	<ul style="list-style-type: none"> <li>• Mayor tráfico con Memoria</li> <li>• Formato más complejo</li> </ul>
<b>Memoria-Memoria</b>	<ul style="list-style-type: none"> <li>• Muchos tipos de direccionamiento</li> <li>• Número mínimo de instrucciones por programa</li> </ul>	<ul style="list-style-type: none"> <li>• Mucho acceso a memoria</li> <li>• Formato complejo</li> </ul>

#### Conclusión

- Dentro de las máquinas con registros de propósito general, las que operan en las instrucciones ALU de registro - registro (RR) son las que optimizan el uso de registros, quedando el acceso a memoria limitado a las instrucciones de carga y almacenamiento.



### 3.3. Orden de ubicación de los datos

Dos alternativas:

big-endian	Selección basada en motivos de compatibilidad
litle-endian	

#### Conclusión

- ❑ Es indiferente desde el punto de vista del rendimiento. Serán motivos de compatibilidad con otros procesadores los que determinen una elección

### 3.4. Alineamiento de datos

Dos alternativas:

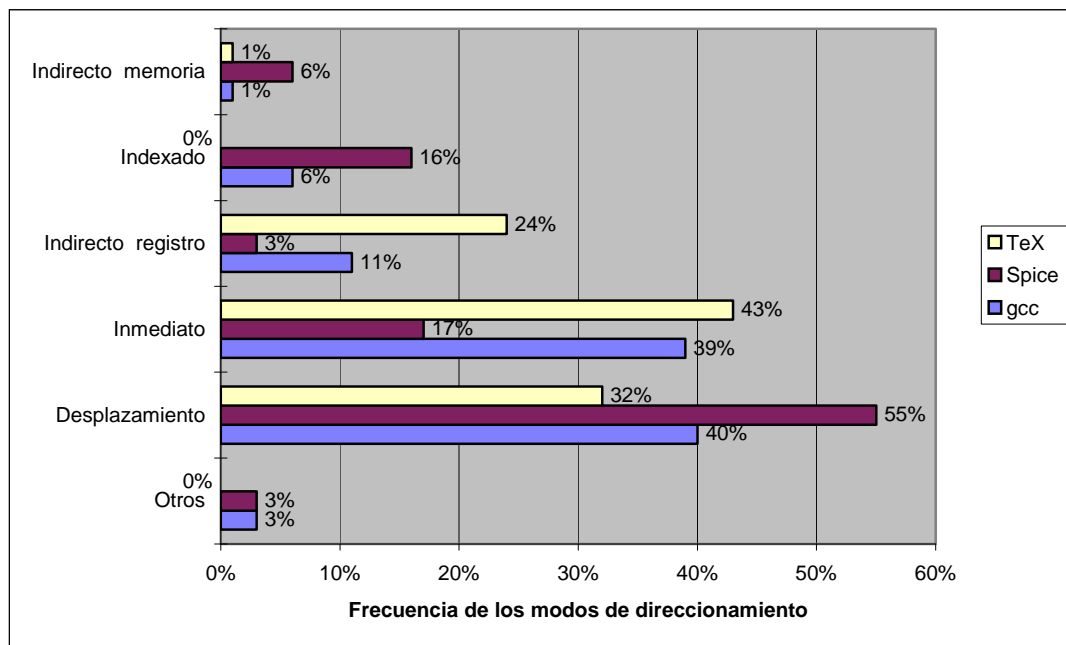
acceso alineado	<ul style="list-style-type: none"> <li>• más rápido</li> </ul>
acceso no alineado	<ul style="list-style-type: none"> <li>• más lento en general</li> <li>• mayor flexibilidad</li> </ul>

#### Conclusión

- ❑ Datos alineados, y si el procesador permite los no alineados, será el compilador quien genere siempre datos alineados.

### 3.5. Direccionamientos

Los modos de direccionamiento pueden reducir significativamente el número de instrucciones de un programa. Sin embargo, añaden complejidad al repertorio aumentando con ello el *CPI* (número medio de ciclos por instrucción). En la gráfica siguiente aparecen los resultados de medir los modos de direccionamiento que utilizan 3 programas del *SPEC89* sobre la arquitectura VAX (una de las que más modos de direccionamiento dispone): *TeX*, *Spice* y *gcc*. Como puede observarse en la gráfica, los direccionamientos inmediato y con desplazamiento dominan con diferencia sobre los demás. No se ha incluido el direccionamiento relativo que se utiliza casi exclusivamente en las instrucciones de bifurcación.



El direccionamiento registro + desplazamiento alcanza más del 75%. El tamaño de los desplazamientos no aparece en la gráfica pero varía de 12 a 16 bits en un porcentaje que va del 75% al 99%. El tamaño del campo inmediato varía de 8 a 16 bits en un porcentaje que va del 50% al 80%.

### Conclusiones

Una máquina eficiente, que favorezca los casos frecuentes (que como hemos visto por la ley de Amdahl son los que más aportan a la ganancia de velocidad global de la máquina) debería soportar:

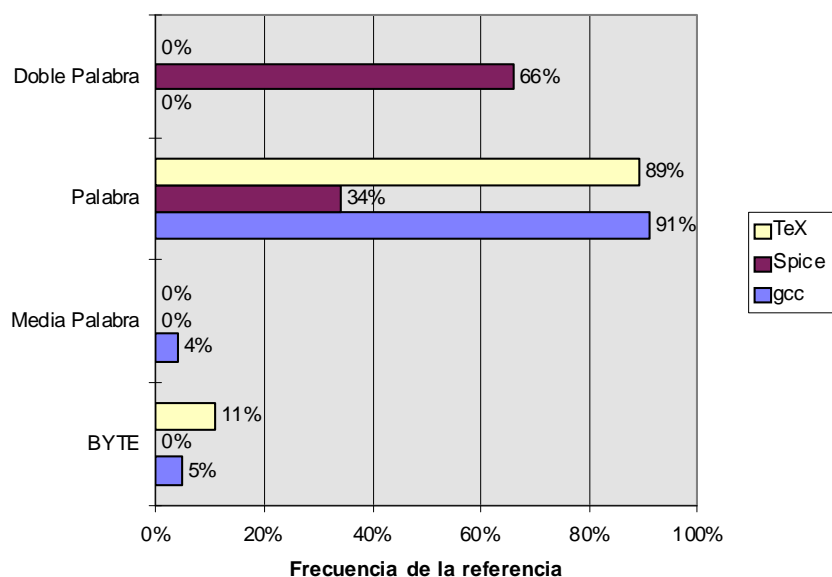
- ❑ Direccionamientos registro + desplazamiento y el inmediato
- ❑ Tamaños de los desplazamientos de 12 a 16 bits
- ❑ Tamaño del dato inmediato de 8 a 16 bits
- ❑ La supresión de los modos complejos no afectan decididamente al rendimiento

### 3.6. Datos operando

La siguiente figura muestra el porcentaje de referencias en los *benchmarks* anteriores a los objetos de datos mas usuales: byte, media palabra, palabra y doble palabra. Se desprende que los accesos a datos de longitud palabra o doble palabra dominan sobre los demás. Si a ello añadimos la necesidad de acceder al elemento mínimo que define la resolución del direccionamiento, así como el soporte del tipo carácter, es decir, el byte; y la existencia en la mayoría de los procesadores de operaciones hardware en punto flotante, llegamos a las siguientes

### Conclusiones

- ❑ enteros de 16 y 32 bits
- ❑ flotantes de 64 bits
- ❑ caracteres de 8 bits



### 3.7. Operaciones

Se cumple en la práctica que las operaciones más simples son las que más se repiten en los programas, concretamente las operaciones de carga, salto condicional, comparación, almacenamiento, suma, and, resta, transferencia entre registros y saltos-retornos de subrutina se

llevan el 96% de las ocurrencias, para el caso de programas enteros ejecutándose sobre la familia x86, tal como muestra la siguiente tabla:

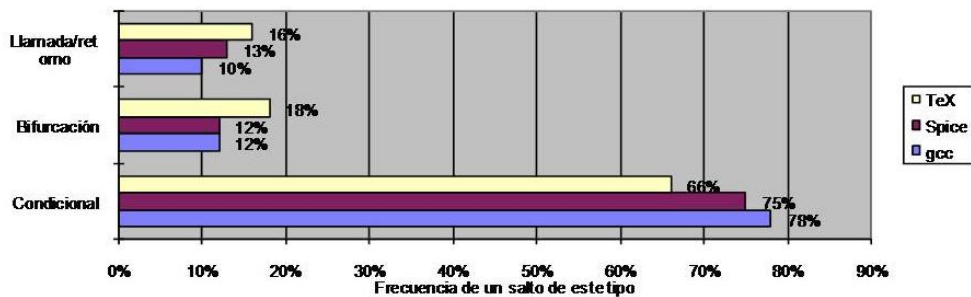
Ordenación	instrucción x86	% total ejecutadas
1	Carga	22%
2	Salto condicional	20%
3	Comparación	16%
4	Almacenamiento	12%
5	Suma	8%
6	And	6%
7	Resta	5%
8	Transferencia RR	4%
9	Salto a subrutina	1%
10	Retorno de subrutina	1%
	<b>TOTAL</b>	<b>96%</b>

### Conclusiones

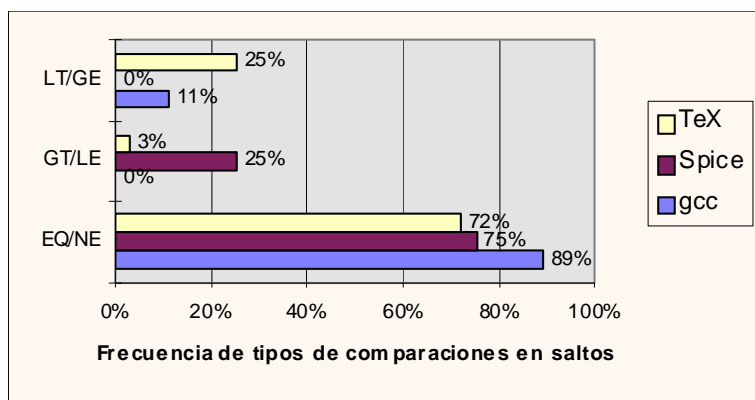
- El repertorio ISA de un procesador eficiente no deberá incluir muchas más operaciones que las aparecidas en la tabla anterior.

### 3.8. Sentencias de salto condicional

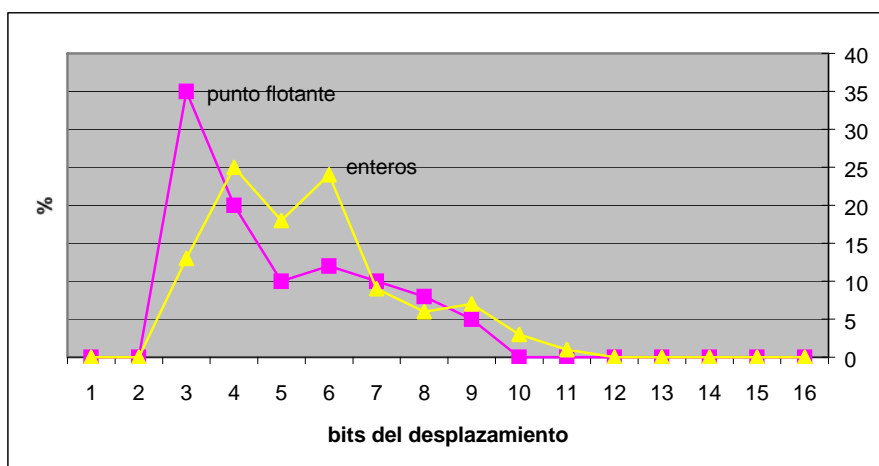
En la siguiente figura tenemos el porcentaje de aparición de los tres tipos de sentencias de ruptura de secuencia. Como se puede observar las bifurcaciones condicionales ocupan el primer lugar (un 75%), por lo que resulta importante diseñar de forma eficiente el mecanismo de generación de condiciones y salto sobre el valor de las mismas.



Si analizamos la distribución de los saltos condicionales observamos (figura siguiente) que más de un 70% son saltos sobre igual o diferente, y un gran número son comparaciones con cero. Esto justifica que algunos procesadores incorporen un registro cuyo contenido es siempre cero.



La gráfica siguiente muestra la distribución del desplazamiento (número de bits) relativa al PC:



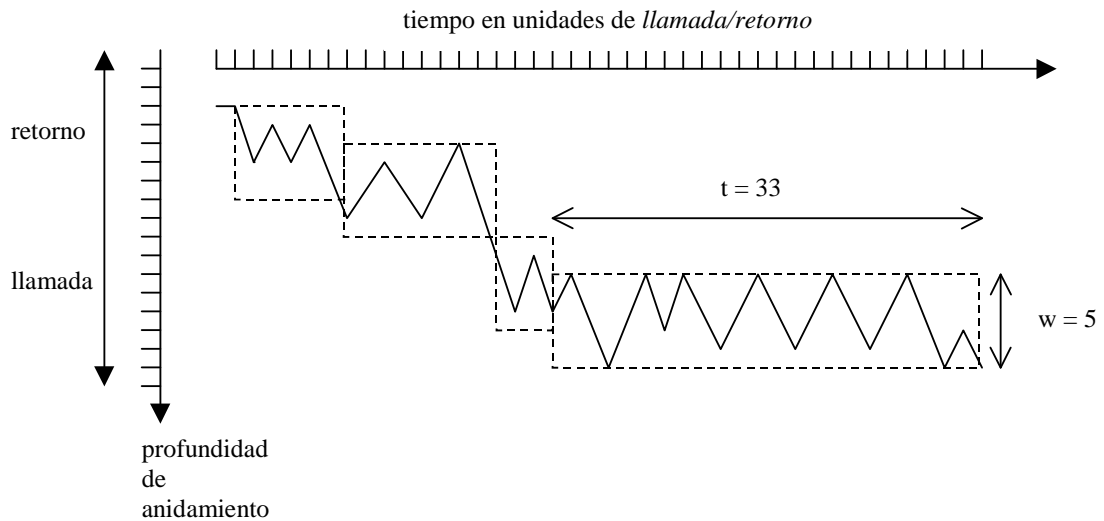
### Conclusiones

- ❑ Instrucciones que integren el test sobre la condición y el correspondiente salto
- ❑ Registro cuyo contenido es inalterable igual a cero.
- ❑ Desplazamiento de 8 bits

### 3.9. Llamadas a procedimientos (subrutinas)

Aunque las instrucciones de llamada a procedimientos no son tan frecuentes como las de bifurcación condicional, su consumo de tiempo en los LAN es elevado debido a la gestión de los registros de activación (RA) que deben de realizar. Resulta, pues, conveniente optimizar la ejecución de estas instrucciones. Del análisis de los datos se deducen los siguientes hechos:

- Más del 98% de las llamadas utilizan menos de 6 parámetros
- Más del 92% de las llamadas utilizan menos de 6 variables locales
- La variación del nivel de anidamiento en la que se mueve un programa se mantiene relativamente pequeña a lo largo de una secuencia de llamadas/retorno, como se muestra en la siguiente figura:



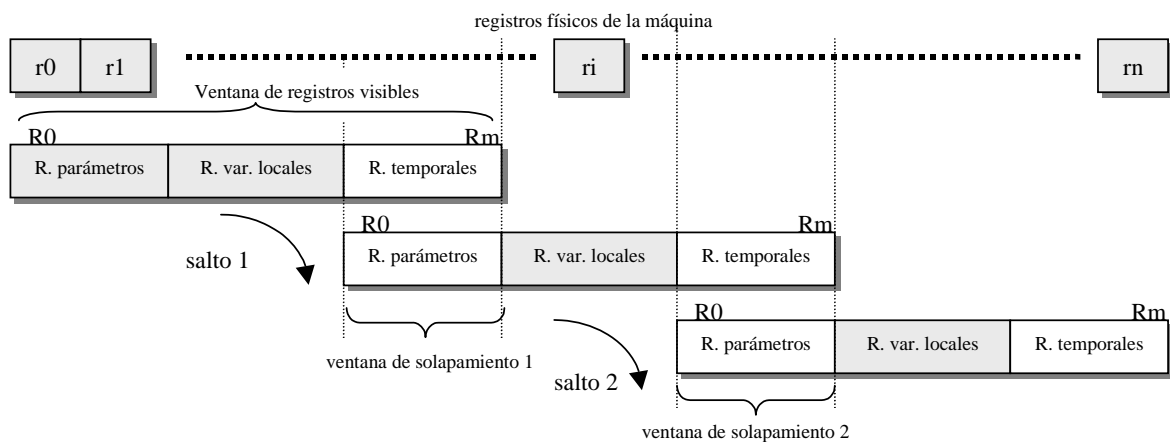
Puede observarse que la variación del nivel de anidamiento se ha mantenido dentro de un valor  $w = 5$  durante una secuencia de  $t = 33$  llamadas/retornos de procedimientos.

### Conclusiones

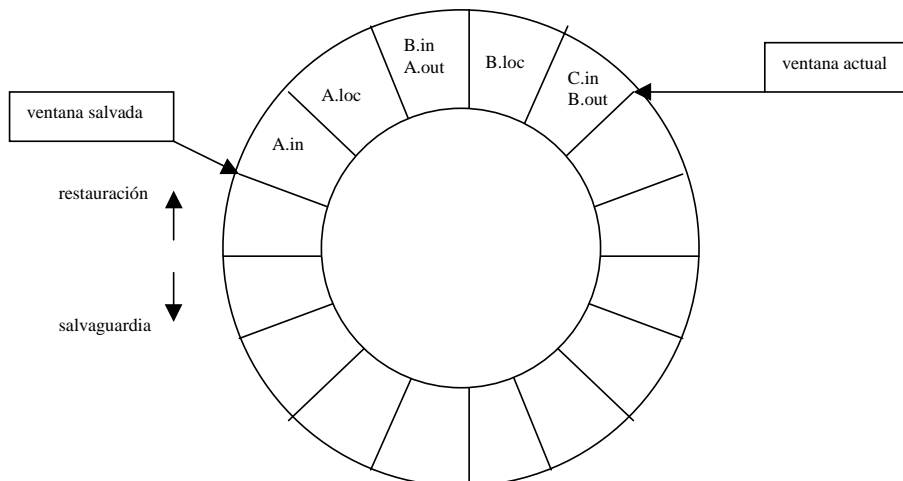
- Este comportamiento de los programas en lo relativo a las llamadas a procedimientos se ha explotado en algunos procesadores (por ejemplo, SPARC) utilizando una ventana de registros para soportar los entornos, marcos o registros de activación (RA). Este mecanismo lo analizamos en el punto siguiente.

### 3.10.Registros de propósito general: ventanas para soportar llamadas a procedimientos

El procesador dispone de  $n$  registros físicos  $r_0, r_1, \dots, r_n$  de los que en cada momento sólo son visibles para el programador (ventana)  $m$  de ellos:  $R_0, R_1, \dots, R_m$  ( $m < n$ ). Cuando todavía no se ha realizado ningún salto a subrutina, los registros visibles coinciden con los  $m$  primeros registros físicos. Cuando se ejecuta una instrucción de salto, los registros visibles se desplazan sobre los físicos de tal manera que se da un solapamiento entre los últimos antes del salto y los primeros después del salto. Es en los registros de solapamiento donde se realiza el paso de parámetros a la rutina y la devolución de resultados.

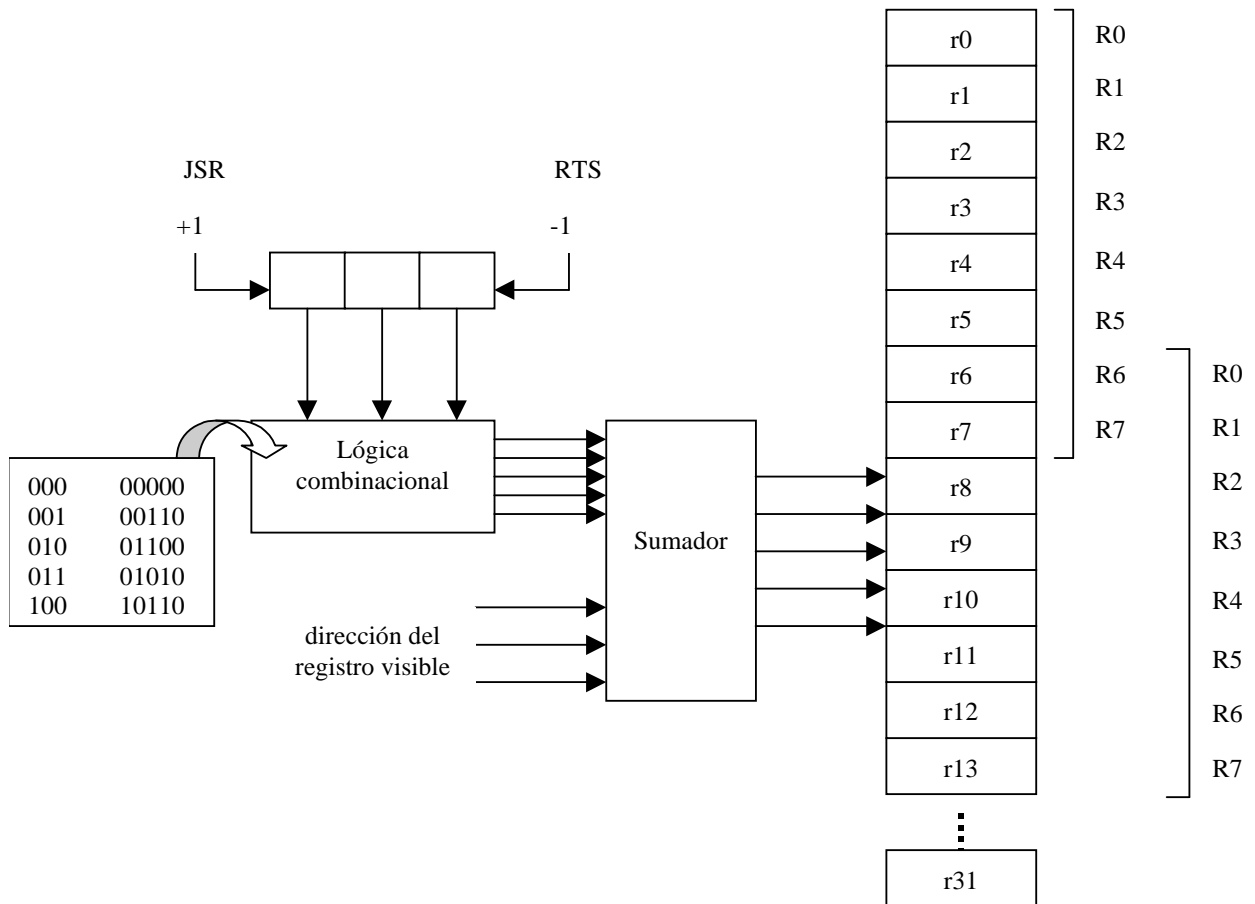


El conjunto de registros tiene una estructura de buffer circular, de tal manera que cuando se agota su capacidad se lleva a memoria el conjunto de registros del entorno (registro de activación) que primero se introdujo (primera llamada). El tamaño del buffer se elige de manera tal que permita soportar un nivel de anidamiento activo (en registros) de acuerdo a los datos empíricos que muestran los programas. En nuestro caso vimos que una variación de nivel igual a 5 se mantenía durante 33 llamadas/retorno, lo que significa que durante esas 33 llamadas todos los registros de activación se hubiesen soportado en un sistema con 5 ventanas .



### 3.11.Ejemplo

**3.12.**En la siguiente figura se presenta el diseño de un sistema de registros con ventana de solapamiento que dispone de 32 registros físicos,  $r_0, r_1, \dots, r_{31}$ , y una ventana de 8 registros:  $R_0, R_1, \dots, R_7$ . El solapamiento es de 2 registros.



#### 4. Influencia de los compiladores de lenguajes de alto nivel

Las prestaciones de los actuales computadores al ejecutar un programa dependen significativamente del compilador que se utilice. Cuando se diseña el repertorio de instrucciones se tiene en mente en todo momento la tecnología de diseño de compiladores. Las decisiones arquitectónicas afectan directamente a la complejidad de diseño de un buen compilador

Un compilador consta de una serie de etapas, las cuales transforman representaciones abstractas de alto nivel, en representaciones de más bajo nivel, hasta llegar al repertorio de instrucciones. Un compilador busca en primer lugar la **exactitud de la traducción** (mantenimiento de la semántica del programa en la transformación), en segundo lugar la **velocidad del código** generado, en tercer lugar el **tamaño del código**, en cuarto lugar la **velocidad del compilador**, y en quinto lugar el **soporte a la depuración**. Desde el punto de vista del rendimiento que ahora nos ocupa, la velocidad del código generado es el factor a optimizar

Las optimizaciones realizadas por los compiladores modernos podemos resumirlas en las siguientes:

- ❑ **Optimización de alto nivel:** realizadas en el código fuente
  - Integración de procedimientos: sustituye la llamada a un procedimiento por el cuerpo de éste (expansión de macros).
- ❑ **Optimizaciones locales:** afectan a fragmentos de código lineal (sin bifurcaciones)
  - Eliminación de sub-expresiones comunes
  - Propagación de constantes



- Reducción del tamaño de la pila en la evaluación de expresiones reorganizando su estructura
- **Optimizaciones globales:** afectan al programa completo, son más complejas de implementar.
  - Optimización de bucles
  - Eliminación de sub-expresiones comunes de alcance global (incluyendo saltos)
  - Propagación de copias: sustituye todas las instancias de una variable asignada.
- **Optimización del uso de registros:** es una de las que reporta mayor incremento de rendimiento. La estudiaremos con más detalle a continuación.
- **Optimizaciones dependientes de la máquina:** Aprovechan el conocimiento de las arquitecturas específicas.
  - Multiplicación por una constante y sustitución por sumas y desplazamientos
  - Elección del desplazamiento más corto en los saltos

Por último conviene destacar que las dos ayudas más importantes que la arquitectura de un procesador puede prestar al diseño del compilador son:

- Regularidad del repertorio, es decir, ortogonalidad de sus elementos para simplificar la generación de código
- Proporcionar funciones primitivas, no soluciones codificadas en las instrucciones, pues estas resultan difícil de utilizar cuando el caso se aparta ligeramente del que originó su diseño.

#### 4.1. Optimización de registros

El uso optimizado de los registros de una máquina es responsabilidad del compilador cuya misión es mantener en registros (y no en memoria) los operandos necesarios para el mayor número posible de cálculos, minimizando las operaciones de carga/almacenamiento de registros que requieren el acceso a memoria

El proceso de asignación óptima de registros se realiza en dos fase:

1) Diseño del grafo de interferencias entre las variables del programa:

nodos: las variables

arcos: entre variables que están activas simultáneamente en el programa

2) Coloreado del grafo

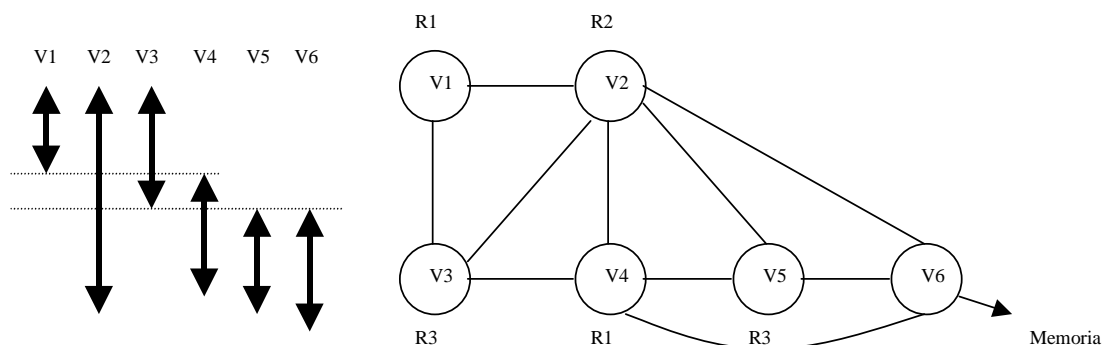
Se intenta colorear el grafo con  $n$  colores, siendo  $n$  el nº de registros disponibles en la máquina. A los nodos no coloreados se le asignan posiciones de memoria y se utilizan instrucciones de carga/almacenamiento.

Como se sabe este es un problema NP-duro que requiere el uso de heurísticas muy elaboradas para acortar el tiempo de proceso.

#### Ejemplo

Asignación de las 6 variables (V1,...V6) que aparecen en el segmento de programa de la siguiente figura, en la que se indica de forma gráfica sus interferencias. Se disponen de 3 registros en la máquina: R1, R2 y R3.

- 1) Se construye el grafo de interferencias tal como se muestra en la siguiente figura.
- 2) Se colorea el grafo asignando a los nodos los tres registros de manera que no se asigne el mismo registro a dos nodos conectados por un arco en el grafo.
- 3) Los nodos que no pueden ser asignados a registros se asignan a memoria (V7)



## 5. Procesadores RISC y CISC

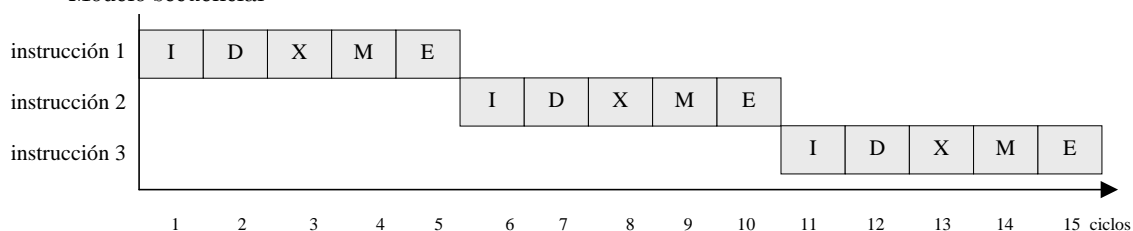
### 5.1. Características comparativas

CISC(Complex Instruction Set Computers)	RISC( Reduced Instruction Set Computers)
• Arquitectura RM y MM	• Arquitectura RR (carga/almacenamiento)
• Diseñadas sin tener en cuenta la verdadera demanda de los programas	• Diseñadas a partir de las mediciones practicadas en los programas a partir de los 80
• <b>Objetivo:</b> dar soporte arquitectónico a las funciones requeridas por los LANs	• <b>Objetivo:</b> dar soporte eficiente a los casos frecuentes
• Muchas operaciones básicas y tipos de direccionamiento complejos	• Pocas operaciones básicas y tipos de direccionamiento simples
• Instrucciones largas y complejas con formatos muy diversos ==> decodificación compleja (lenta)	• Instrucciones de formato simple (tamaño fijo) ==> decodificación simple (rápida)
• Pocas instrucciones por programa ==> elevado número de ciclos por instrucción (CPI)	• Muchas instrucciones por programa ==> reducido número de ciclos por instrucción (CPI)
• Muchos tipos de datos ==> interfaz con memoria compleja	• Sólo los tipos de datos básicos ==> interfaz con memoria sencilla
• Número limitado de registros de propósito general ==> mucho almacenamiento temporal en memoria	• Número elevado de registros ==> uso eficiente por el compilador del almacenamiento temporal
• Baja ortogonalidad en las instrucciones ==> muchas excepciones para el compilador	• Alto grado de ortogonalidad en las instrucciones ==> regularidad en el compilador

### 5.2. Segmentación

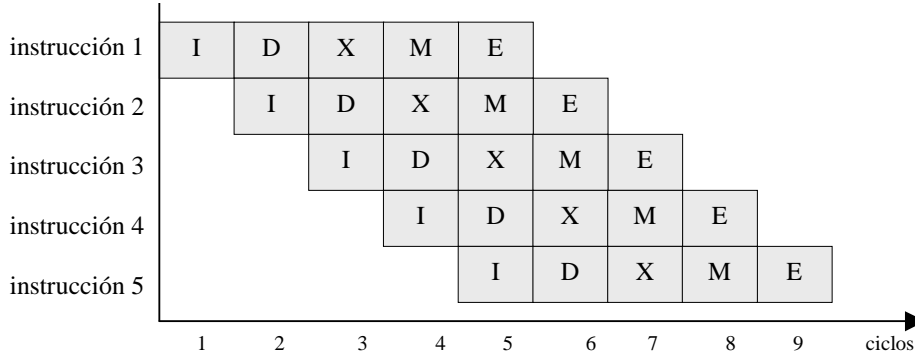
Una de las ventajas de los procesadores RISC es la facilidad que presentan sus instrucciones para ser ejecutadas de forma segmentada, es decir, solapando dos o más fases de ejecución. La tarea de cada instrucción se divide en etapas, de tal forma que en cada ciclo se ejecuta una etapa de una instrucción, simultaneándose la ejecución de etapas de diferentes instrucciones.

- Modelo secuencial



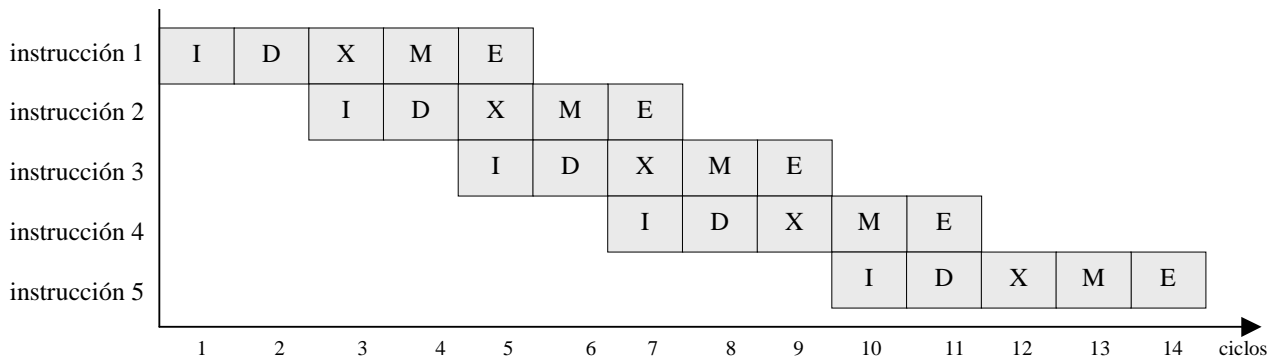
- Modelo segmentado lineal

Los primeros RISC tenían un modelo de ejecución *segmentado lineal* en el que todas las instrucciones pasan por las mismas etapas y en cada ciclo entra una nueva instrucción a ejecutar.



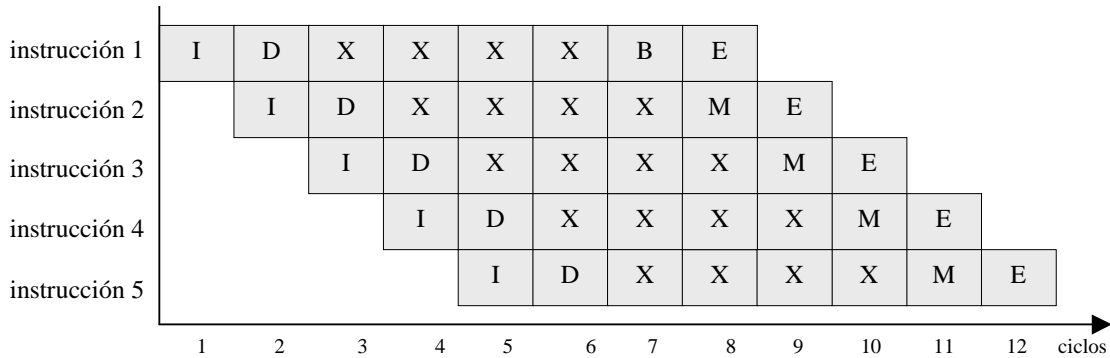
- Modelo infrasegmentado

En cada  $n$  ciclos se lanza una nueva instrucción (Stanford MIPS). Esto está motivado fundamentalmente porque el tiempo de ciclo no permite acceder a algunos recursos hardware como la memoria, por lo que se necesitan dos (o más) ciclos para acceder a la misma.



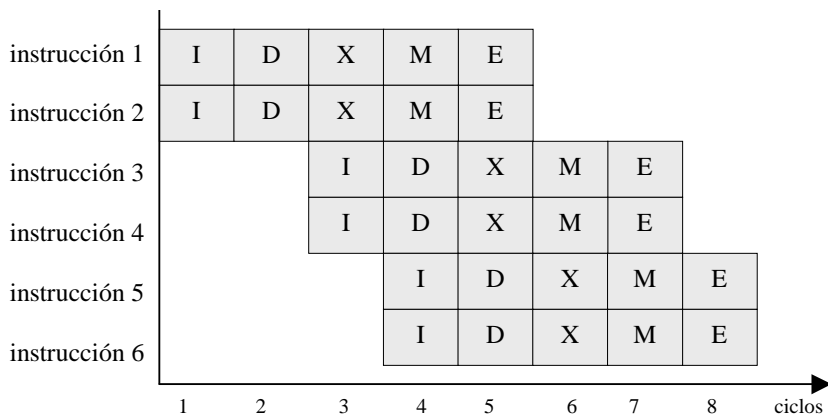
- Modelo supersegmentado

Aumentan el rendimiento disminuyendo el tiempo de ciclo. Esto lo consiguen aumentando considerablemente el número de etapas del procesador (o dividiendo la etapa en subciclos). Con ello se consigue que el rendimiento máximo del procesador aumente; pero tiene un inconveniente que va a afectar considerablemente al rendimiento real: el aumento de las latencias de las instrucciones inducido por el aumento del número de conflictos entre las instrucciones que tengan algún tipo de dependencia a la hora de la ejecución (dependencias de datos, estructurales o de control), con la consiguiente pérdida de rendimiento.



- Modelo superescalar

Para aumentar el rendimiento el modelo *superescalar* lanza a ejecutar en cada ciclo  $n$  instrucciones a la vez, siendo  $n$  el *orden o grado* del superescalar. Sin embargo, para que se puedan lanzar a ejecutar a la vez  $n$  instrucciones no debe haber conflictos entre ellas. Éste es un factor que va a limitar, de forma significativa, el rendimiento, y es una medida del paralelismo implícito que presentan las aplicaciones y que pueden extraer los compiladores.



### 5.3. Dependencias

Son las causantes de la disminución de rendimiento en la ejecución segmentada y superescalar. Son de tres tipos:

- Dependencias de datos

Producidas por la referencia a un mismo dato por más de una instrucción. Hay tres tipos:

Dependencia de flujo (verdadera dependencia) *RAW (Read After Write)*

S1 → S2

S1        X := ...

.

.

.

S2        ...:=...X...

Ejemplo:    Load r1, a  
              Add r2, r1, r1

Antidependencia *WAR (Write After Read)*

S1 --/→ S2

S1        ...:=...X...

.

.

.

S2        X :=...

Ejemplo:    Mul r1, r2, r3  
              Add r2, r4, r5

Dependencia de salida WAW (Write After Write)

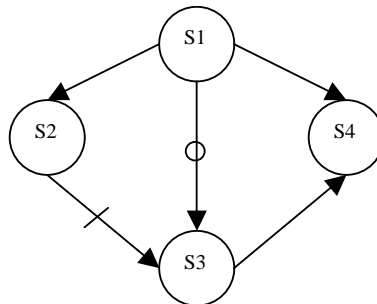
S1 → S2

```
S1    X := ...
.
.
.
S2    X := ...
```

Ejemplo:      Mul r1, r2, r3  
                  Add r1, r4, r5

• Grafo de dependencias

```
S1:    Load  R1, A            /R1 ← M(A)/
S2:    Add    R2, R1          /R2 ← <R1>+<R2>/
S3:    Move   R1, R3          /R1 ← <R3>/
S4    Store  B, R1            /M(B) ← <R1>/
```



□ Dependencias de control

Debido a las instrucciones de salto condicional.

Ejemplo:            Mul r1, r2, r3  
                       Jz zproc  
                       Sub r4, r1, r1  
                       .  
                       .  
                       .  
                       zproc    Load r1, x

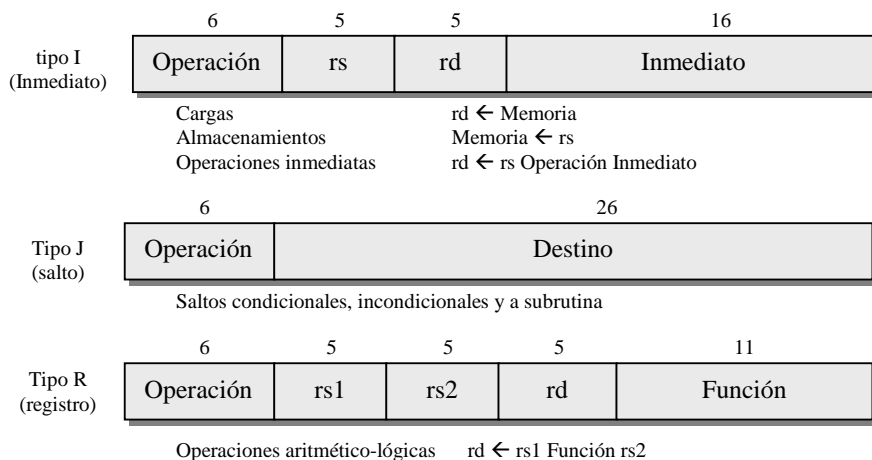
□ Dependencias de recursos(estructurales)

Producidas al tratar de usar recursos compartidos

**5.4. Ejemplo: DLX**

- Arquitectura de 32 bits
- 32 registros de uso general (GPR) de 32 bits. R0 siempre contiene 0
- Memoria direccionable por byte en modo big-endian
- Instrucciones de longitud fija de 32 bits y alineadas
- Un solo modo de direccionamiento: registro + desplazamiento (16)
- Datos 8, 16 y 32 bits (enteros)

- Formatos de las instrucciones



- Repertorio de instrucciones

Veamos algunas de las instrucciones de los diferentes grupos:

INSTRUCCIONES ARITMETICO/LOGICAS		
Instrucción	Ejemplo	Semántica
ADD (Sumar)	$R1 \leftarrow R2 + R3$	ADD R1, R2, R3
ADDI (Sumar inmediato)	$R1 \leftarrow R2 + 3$	ADDI R1, R2, #3
SLLI (Desplazamiento lógico izquierda)	$R1 \leftarrow R2 \ll 5$	SLL R1, R2, #5
SLT (Activa menor que)	IF (R2 < R3) R1 ← 1 ELSE R1 ← 0	SLT R1, R2, R3

INSTRUCCIONES DE CONTROL DE FLUJO		
Instrucción	Semántica	Ejemplo
J (Salto)	PC ← nombre	J nombre
JAL (Salto y enlace)	$R31 \leftarrow PC + 4$ ; PC ← nombre	JAL nombre
JR (Salto registro)	PC ← R3	JR R3
BEQZ (Bifurcación si igual cero)	IF (R4 == 0) PC ← nombre	BEQZ R4, nombre
BNEZ (Bifurcación si no igual cero)	IF (R4 != 0) PC ← nombre	BNEZ R4, nombre

INSTRUCCIONES DE CARGA/ALMACENAMIENTO		
Instrucción	Semántica	Ejemplo
LW (Cargar palabra)	$R1 \leftarrow_{-32} M[30 + R2]$	LW R1, 30 (R2)
	$R1 \leftarrow_{-32} M[1000 + 0]$	LW R1, 1000 (R0)
LB (Cargar byte)	$R1 \leftarrow_{-32} (M[40 + R3]_0)^{24} \## M[40+R3]$	LB R1, 40 (R3)
LBU (Cargar byte sin signo)	$R1 \leftarrow_{-32} 0^{24} \## M[40+R3]$	LBU R1, 40 (R3)
SW (Almacenar palabra)	$M[500+R4] \leftarrow_{-32} R3$	SW 500(R4), R3

SH (Almacenar media palabra)	$M[502+R2] \leftarrow_{16} R3_{16..31}$	SH 502(R2), R3
SB (Almacenar byte)	$M[41+R3] \leftarrow_8 R2_{24..31}$	SB 41(R3), R2

Síntesis de otros modos de direccionamiento y operaciones	
Instrucción aparente	Instrucción real
LW R2, <despl. de 16 bits> Dir. Directo (16 bits)	LW R2, <despl. de 16 bits> (R0)
LW R2, <despl. de 32 bits> Dir. Directo (32 bits)	LW R1, <16 bits altos del despl.> (R0) LW R2, <16 bits bajos del despl.> (R1)
LW R2, <despl. de 32 bits> (R4) Dir. Base+Desplazamiento (32 bits)	LW R1, <16 bits altos del despl.> (R0) ADDU R1, R1, R4 LW R2, <16 bits bajos del despl.> (R1)
Carga de un registro con datos inmediatos	ADDI R1, R0, #8                      R1 ← 8
Transferencia entre registros	ADD R1,R0,R2                      R1 ← R2

- Eficiencia de los procesadores RISC

Comparación entre el VAX y el DLX utilizando 5 programas del SPEC92 (*compress, eqntott, espresso, gcc, li*)

Se representan las siguientes relaciones:

- relación (ratio) del número de instrucciones ejecutadas
- relación (ratio) de CPIs
- relación (ratio) de rendimiento medido en ciclos de reloj

Discusión:

DLX ejecuta aproximadamente el doble de instrucciones que el VAX

El CPI del VAX es aproximadamente 6 veces que el del DLX

Conclusión:

DLX tiene aproximadamente el triple de rendimiento que el VAX

