Arquitectura de computadores

TEMA 1 RENDIMIENTO DE LOS COMPUTADORES

Tema 1
Rendimiento de los computadores

1. NECESIDAD DE EVALUAR EL RENDIMIENTO

Hay distintos tipos de personas que ven necesario evaluar el rendimiento de un computador:

- **Ingenieros**: Se necesita alguna herramienta o medida que permita decidirnos por múltiples alternativos de diseña, eligiendo la más adecuada en cada momento.
- **Fabricantes y vendedores**: Deben poder comparar y diferenciar su producto del de la competencia.
- **Compradores**: Necesitan algún mecanismo que ayude a decidirse entre los distintos computadores en el mercado.

Hay, no obstante, que hacer dos reflexiones:

- Cuando compramos un ordenador, nos llevamos el sistema completo así que debemos evaluar no solo el procesador, sino también la memoria, sistema operativo, compiladores…
- Los resultados de una evaluación tienen que ser reproducibles y presentar claramente los datos obtenidos para poder volver a llevar a cabo esa evaluación en ese u otro ordenador y así poder compararlos entre sí.

2. FIGURAS DE MÉRITO

Las medidas que nos ayudan a evaluar el rendimiento de manera cuantitativa son las llamadas figuras de mérito, y son tres principalmente: el tiempo, los MIPS y los MFLOPS.

Formulaciones

$T_{ciclo} = 1/frecuencia$

$T_{UCP} = ciclos_{UCP} \cdot T_{ciclo}$

$$T_{UCP} = \frac{ciclos_{UCP}}{frecuencia}$$

$$CPI = \frac{ciclos_{UCP}}{N}$$

$$IPC = \frac{1}{CPI} = \frac{N}{ciclos_{UCP}}$$

El **tiempo** es la mejor medida que se puede utilizar, dado que es consistente y real para comparar el rendimiento de dos ordenadores, tanto si medimos el tiempo de respuesta (el tiempo que tarda una aplicación desde que empieza hasta que acaba) como si es la productividad (la cantidad de tareas ejecutadas en un espacio de tiempo determinado).

Habitualmente trabajaremos con el tiempo de UCP (T_{UCP}) que incluye el tiempo que tarda la máquina en ejecutar código de usuario más el tiempo en ejecutar código del sistema operativo, ya que la frontera entre ambos es muy diluida. Por otro lado este tiempo debe poder relacionarse con algún parámetro físico del computador, y este parámetro es la frecuencia. Así que un ordenador tenga una frecuencia de 200 MHz, significa que tiene un reloj interno de 200 millones de ciclos por segundo, como el tiempo de ciclo es inverso a la frecuencia, el tiempo de ciclo es de 5 ns (es decir $5 \cdot 10^{-9}$ segundos).

Es habitual presentar el tiempo de UCP en función de los ciclos de reloj, y si además fuéramos capaces de calcular el número de instrucciones ejecutadas (N), podríamos obtener el número medio de ciclos por instrucción (CPI); también se puede utilizar justamente el inverso del CPI, es decir el IPC que es el número de instrucciones ejecutadas por ciclo; al final podemos reescribir la fórmula inicial como sigue a continuación:

$$T_{UCP} = ciclos_{UCP} \cdot T_{CICLO} = N \cdot \frac{ciclosUCP}{N} \cdot T_{CICLO} = N \cdot CPI \cdot T_{CICLO}$$

Donde N es el número de instrucciones a ejecutar, CPI es el número medio de ciclos por instrucción y T_{CICLO} es la inversa de la frecuencia del ordenador.
De esta ecuación se desprendes 3 posibles opciones para optimizar un procesador:

- Reducir (N) el número de instrucciones del programa, lo que se relaciona con la arquitectura del nivel de lenguaje máquina y la compilación.
- Reducir (CPI) el número medio de ciclos por instrucción, que depende de la arquitectura del nivel de lenguaje máquina y de la organización del procesador.
- Reducir (T_{CICLO}) el tiempo del ciclo del procesador, asociado este cambio a la tecnología de hardware y la organización del procesador.

Otra medida son los **MIPS**, o millones de instrucciones de lenguaje máquina ejecutados por segundo. Como observamos en las fórmulas laterales, equivale al número de instrucciones ejecutadas dividido por el tiempo empleado en realizarlas y dividido por 1 millón. También equivale a la frecuencia dividido por el número medio de ciclos por instrucción.

MIPS

$$MIPS = \frac{N}{T_{UCP} \cdot 10^6}$$

$$MIPS = \frac{frecuencia}{CPI \cdot 10^6}$$

La ventaja de esta medida es que es sencilla de asimilar, pero depende del juego de instrucciones, lo que dificulta comparar máquinas diferentes; varía en función de los programas que se ejecutan.

Hace tiempo los fabricantes y vendedores cuando utilizaban esta figura de mérito realmente utilizaban los MIPS de pico, es decir, una velocidad que seguro nunca alcanzarán los procesadores en una aplicación real pues consideraban los recursos ideales: memoria ideal (siempre disponible), instrucciones distribuidas homogéneamente en todas las unidades funcionales trabajando siempre al máximo de rendimiento, etc.

Los **MFLOPS** son los millones de operaciones de coma flotante ejecutadas por segundo; la definición es prácticamente igual que los MIPS pero utilizando únicamente las operaciones en coma flotante; esto hace que un programa que no tenga ninguna operación en coma flotante tenga un MFLOPS de 0, que dependa mucho del programa y del juego de instrucciones. En cambio los MFLOPS permiten comparar máquinas diferentes porque el número de operaciones de coma flotante es constante y viene determinado por el programa que quiera resolverse, no varía según la máquina en la que se ejecute el programa.

MFLOPS

$$MFLOPS = \frac{N_{COMAflo\,tante}}{T_{UCP} \cdot 10^6}$$

También se ha utilizado los MFLOPS de pico que corresponde a la suma de los MFLOPS de pico que puede obtener cada una de las unidades funcionales del procesador trabajando al máximo rendimiento posible, circunstancia que es imposible que se de nunca en el trabajo habitual del procesador.

3. PROGRAMAS PARA EVALUAR EL RENDIMIENTO

Para evaluar el rendimiento de un computador se utilizan una serie de programas estándar (benchmarks) que pretenden simular situaciones de uso cotidiano.

Los **programas sintéticos** son programas de tamaño moderado que intentan simular un entorno de trabajo con una carga de trabajo también determinada. El problema es que al tratarse de pequeños programas son fáciles de manipular, así un fabricante dedicará esfuerzo y dinero a hacer que sus máquinas ejecuten estos programas sintéticos de manera muy efectiva utilizando diferentes estrategias de optimización que, en cualquier caso, no se podrán aplicar después a los programas reales que son los que en realidad nos interesan.

Los **programas de juguete** son programas pequeños de entre 10 y 100 líneas de código y de los cuales se conoce el resultado a priori. Hoy día son muy poco utilizados.

Los **núcleos** son pequeños fragmentos de código extraídos de aplicaciones reales que sólo se ejecutan para evaluar el rendimiento de un computador.

Las **aplicaciones reales** son la solución más utilizada hoy en día, y son aplicaciones que pueden ejecutarse en cualquier plataforma y sistema operativo sin hacer modificaciones y siendo por tanto, más objetivo su resultado. El problema se centra en decidir qué aplicaciones se utilizan y para ello se creó en el año º988 un el SPEC (Standar Performance Evaluation Corporation) que define un conjunto de programas de test realista y estandarizado. Actualmente se utiliza el SPEC95 que, debido a la velocidad de los microprocesadores actuales, debe ser pronto cambiada. Lo interesante del SPEC95 es que evalúa no solo el procesador, sino el computador completo y obliga a incluir en los resultados las características siguientes con respecto al hardware y software: tipo de procesador y frecuencia, tipo de coprocesador, características de las memorias caché, tamaño de la memoria principal, tipos de dispositivos de entrada/salida utilizados, tipo y versión del sistema operativo, tipo de compilador y versión, tipo de sistema de ficheros y las opciones de compilación utilizadas.

TEMA 2 INTRODUCCIÓN A LA SEGMENTACIÓN

1. MEJORAS EN EL RENDIMIENTO DE LOS MICROPROCESADORES

Ya hemos visto en el tema anterior, que el tiempo de ejecución de una aplicación depende de la siguiente expresión: $T_{UCP} = N \cdot CPI \cdot T_{CICLO}$, por tanto, depende del número de instrucciones a ejecutar (N), el número medio de ciclos por instrucción (CPI) y del tiempo de ciclo del reloj (T_{CICLO}).

Si conseguimos reducir cualquier término o combinación de términos de la anterior ecuación, el tiempo de ejecución se reducirá, y se puede llevar a cabo mediante:

- **El algoritmo**: Hay que resolver el problema de forma más rápida, por tanto mejorando el compilador para generar un código más eficiente, reduciremos el número de instrucciones (N) y/o el CPI, de forma general reduciremos el producto N·CPI.
- **La arquitectura**: Se pueden hacer cambios en el juego de instrucciones del procesador (arquitectura) que genera mejoras en N, o cambios en cómo se ejecutan las instrucciones de lenguaje máquina (microarquitectura) que genera mejoras en CPI y T_{CICLO}.
- **La tecnología**: Las nuevas técnicas de fabricación de semiconductores se traduce directamente en reducciones del T_{CICLO} e indirectamente del CPI.
- **Diseño de máquinas específicas**: Es una mejora no contemplada hoy en día y consiste en fabricar ordenadores para cumplir trabajos específicos. El coste de investigación más desarrollo se tiene que repartir entre pocas máquinas por lo que se eleva excesivamente.

Obviamente, la mayor mejora sucede cuando interaccionan la tecnología y la arquitectura, especialmente la **disminución en la escala de integración**, que implica que los dispositivos sean más rápidos al tener que recorrer los electrones menor distancia, y que puedan integrarse más dispositivos en una misma superficie.

2. PARALELISMO A NIVEL DE INSTRUCCIÓN

Cuando varias instrucciones de un mismo código, pueden ejecutarse en cualquier orden sin que se modifique el resultado final, varias de ellas podrían hacerse al mismo tiempo, y existe lo que llamamos **paralelismo a nivel de instrucción**. Para aprovechar este paralelismo existen dos métodos: La segmentación y la multiplicación.

La **segmentación** es la misma idea que encierra una cadena de montaje en una fábrica. Para montar un coche se descompone su montaje en etapas independientes, hasta pasar por la última en la que se obtiene el vehículo montado. Realmente un coche debe pasar por todas las etapas lo que supone el mismo tiempo en montarlo, pero cuando el coche ya está montado hay varios más ya con estas etapas iniciadas, vamos, tantos como etapas hayamos dividido, por lo que finalmente el proceso es más rápido; claro, más rápido siempre y cuando queremos montar más de un vehículo. Así es posible aplicar la segmentación de dos formas distintas:

- **Segmentación en la función de un microprocesador**: Se descompone una función en otras más sencillas llamadas etapas; se separan estas etapas en registros para evitar que el resultado de una etapa afecte a las demás; se determina el orden en que deben recorrerse estas etapas; se conecta la salida de una etapa con la entrada de la siguiente, y se añade una señal de reloj que indica en qué momento el resultado de todas las etapas se copia a las siguientes.

Segmentación de una función

$T_s = (n+B) \cdot (R + T_s)$
n = número de operaciones
B = Bits
R = Retardo en unidades de t.
T_s = Retraso introducido

Segmentación del proceso de ejecución de las instrucc.

$T_{CICLO} = Max(T_r) + T_r$
$T_S = (n + K) \cdot T_{CICLO}$

Para que esta función de segmentación sea realmente útil, la función a segmentar tiene que ser muy utilizada, ya que la segmentación tiene un coste que se suma al resultado final; y además la función tiene que tener evaluaciones independientes, como la segmentación inicia la ejecución de instrucciones antes de que otras hayan terminado, el resultado de las anteriores no debe afectar a las siguientes. El tiempo que se tarda en realizar una operación con segmentación lo podemos ver en la figura lateral.

- **Segmentación del proceso de ejecución de las instrucciones**: Las instrucciones segmentadas se van realizando en línea paralelamente, para ejecutar más instrucciones por unidad de tiempo; así aunque el resultado final de procesar una instrucción tiene el mismo coste, al haber efectuado finalmente más instrucciones, el coste por instrucción disminuye.

  Así, en esta ocasión, el tiempo de ciclo del procesador, es igual al tiempo de cálculo de la etapa más lenta, más el tiempo de retardo introducido por un registro. Y por ello el tiempo para hacer n operacione segmentadas será el número de operaciones (n) más el número de etapas (K) por el tiempo de ciclo.

La **multiplicación** consiste en aprovechar que hay paralelismo a nivel de instrucción para iniciar la ejecución de más de una instrucción en un mismo ciclo. El objetivo es conseguir un CPI<1 sin aumentar el tiempo de ciclo del procesador.

Para aplicar esta técnica solo hay que multiplicar los recursos (unidades funcionales, procesador, etc). Así para optimizar por ejemplo la suma de naturales, si queremos ir 4 veces más rápido, sumaremos así 4 procesadores. Aunque cada una de las sumas tarda el mismo tiempo en hacerse que sin la multiplicación, puede haber 4 sumas haciéndose al mismo tiempo. Para que esta técnica sea efectiva debe llevarse a cabo en operaciones que se hagan con mucha frecuencia (de lo contrario no tiene sentido) y además deben ser independientes entre ellas.

Por tanto, las condiciones son iguales que en la segmentación, pero la multiplicación es mucho más cara de aplicar. Así para segmentar el sumador en cuatro etapas, solo hace falta añadir 4 registros de segmentación, mientras que para hacer lo mismo con la multiplicación, hay que añadir 4 sumadores completos.

Las dos técnicas pueden combinarse, así que el máximo rendimiento se obtiene aplicando al mismo tiempo la segmentación y la multiplicación.

3. EVOLUCIÓN CONJUNTA DE LA TECNOLOGÍA Y LA ARQUITECTURA

Cronológicamente a lo largo del tiempo, la tecnología y la arquitectura de los procesadores han ido evolucionando paralelamente en la siguiente forma de ejecutar las instrucciones:

- **Secuencial**: Una instrucción, un ciclo de procesador; y para garantizar que todas las instrucciones se ejecutan de manera correcta, el tiempo de ciclo tiene que ser mayor o igual al tiempo necesario para ejecutar la más lenta de todas las instrucciones.
- **Secuencial multiciclo**: Cada instrucción solo para por las fases que necesita, y el tiempo de ciclo es el de la fase de instrucciones que tarda más en ejecutarse. Aun así, cada instrucción tarda más de un ciclo en ejecutarse.
- **Segmentada**: Se aplica la técnica de segmentación al proceso de ejecución de instrucciones; cada una de las fases de ejecución está aislada de las vecinas mediante registros pasa a ser una etapa del procesador. En el cado ideal, se desarrollará una instrucción por ciclo.
- **Segmentada limitada por la búsqueda de instrucciones y datos**: Como la tecnología en la fabricación de procesadores avanza, el cuello de botella se encuentra en el tiempo que tarda la memoria en la búsqueda de datos. Como el tiempo de ciclo es muy grande, se desarrollaron arquitecturas CISC, así como ir a la memoria a buscar datos es muy lento, se procura que cada instrucción haga mucho trabajo para mantener ocupado el procesador durante la fase de

ejecución. Como las instrucciones tienen mucha semántica, se necesitan menos para hacer el mismo trabajo.

- **Segmentada RISC**: La mejora de la tecnología permitió añadir una memoria caché en el microprocesador y la búsqueda de instrucciones pasa a ser muy rápida en comparación con el coste de ejecutar una instrucción muy complicada, por lo que se vuelve al modelo de procesador segmentado con instrucciones sencillas y de rápida ejecución. Se puede continuar aumentando el rendimiento de la máquina aumentando el número de etapas de la segmentación para reducir el tiempo de ciclo. Aumentar las etapas para reducir el tiempo de ciclo, trae consigo aumentar la CPI, pero aún así, el producto CPI por Tiempo de ciclo disminuye, así que ganaremos rendimiento.
- **Superescalar**: Ya solo podemos aumentar el rendimiento, aumentando el hardware, ejecutando más de una instrucción por ciclo y así conseguir que CPI<1.

TEMA 3 PROCESADORES SEGMENTADOS

Tema 3
Procesadores segmentados

1. PROCESADORES SEGMENTADOS LINEALES

Los procesadores segmentados lineales son el tipo de procesador segmentado más sencillo posible, Para que se considere lineal tiene que cumplirse que todas las instrucciones pasen por las mismas etapas en el mismo orden y sin reutilizar ninguna etapa. Esto garantiza que la búsqueda de las instrucciones, el inicio de la ejecución y la finalización de las instrucciones se lleva a cabo en el mismo orden que se encuentran en el código secuencial.

2. ARQUITECTURA DE EJEMPLO

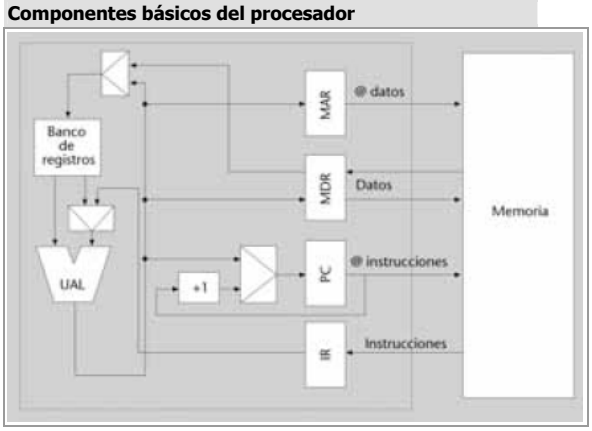
En los lenguajes máquina con operaciones con registros hay tres tipos básicos de instrucciones:

- **Operaciones de cálculo**: Incluyen las operaciones aritméticas y lógicas y tanto pueden ser valores enteros como en coma flotante. La sintaxis es:
  ```
  OP Rd, Rf1, Rf2; Rd = Rf1 OP Rf2
  ```
  Donde OP puede ser cualquier operación (suma, resta y multiplicación).

- **Operaciones de acceso a la memoria**: Hay dos instrucciones básicas: LOAD para ir a buscar un valor a la memoria y STORE para dejarlo:
  ```
  LOAD Rd, Rf1, Rf2; Rd = Memoria[Rf1 + Rf2]
  LOAD Rd, Rf1,  #X; Rd = Memoria[Rf1 +  X ]
  STORE #X, Rf1, Rf2; Memoria[Rf1 + x] = Rf2
  ```

- **Operaciones de control de secuencia**: Se trata de operaciones como saltos incondicionales, condicionales, saltos calculados, llamadas a subrutinas…
  ```
  BEQZ Rd, #X, Si (Rf==0) entonces PC = PC + X
  ```

Componentes básicos del procesador



A partir de esta semántica definida, los componentes básicos del procesador serán los siguientes:

- Un banco de registros con dos puertos de lectura y uno de escritura.
- Una Unidad Aritmético Lógica UAL para las operaciones de cálculo.
- Un sumador para calcular el contador de la instrucción siguiente del programa.
- Varios registros: Un para mantener las direcciones en la memoria para los datos (MAR), uno para las direcciones de la memoria con el fin de ir a buscar las instrucciones (PC), uno para los datos que van a la memoria o vuelven (MDR) y uno para las instrucciones (IR).

3. ARQUITECTURA SEGMENTADA LINEAL

En un primer lugar vamos a identificar las distintas etapas de la segmentación que podemos encontrar para cada uno de los tipos de instrucción y, hecho esto, aplicaremos el mismo número de etapas a todos los tipos de instrucción para asegurarnos que se trata de una arquitectura segmentada lineal.

- **Instrucciones de cálculo**: Encontramos que primero hay que ir a buscar la instrucción a la memoria (B), luego descodificar y leer (DL) dichas instrucciones, aplicar la operación sobre los operandos (UAL) y por último escribir el resultado en el banco de registros (ESC). Por tanto, 4 etapas.

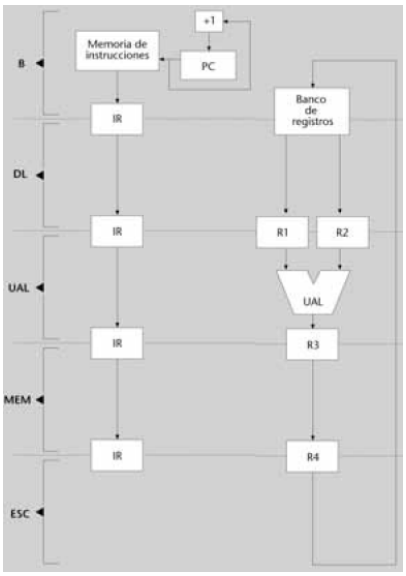
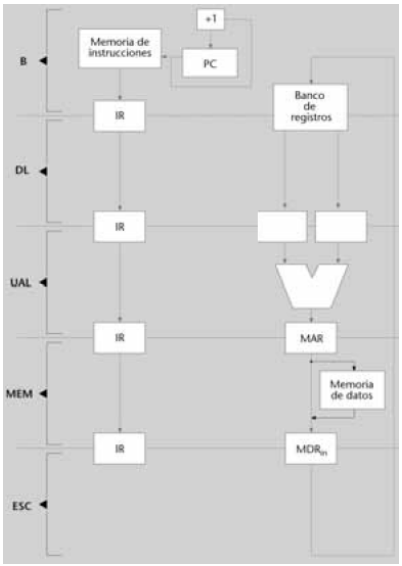
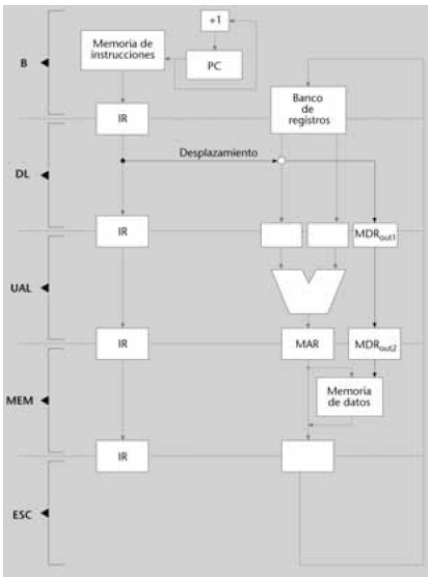
- **Instrucciones de acceso a la memoria**: Las dos primeras etapas, buscar instrucción a la memoria y descodificar y leer es igual que la anterior; después calculamos la dirección de la memoria donde tiene que accederse (ADR), accedemos a la memoria (MEM) y escribimos en el banco de registros si se trata de una instrucción LOAD (ESC) o no hacemos esta última etapa si es una instrucción STORE. Por tanto 4 ó 5 etapas dependiendo del tipo de instrucción.

Por lo tanto, necesitamos, 5 etapas en el camino de datos segmentado, que procedemos a explicar a continuación en la siguiente tabla para cada una de las instrucciones:

	INSTR. CÁLCULO	**ACCESO LOAD**	**ACCESO STORE**
ETAPA B Búsqueda de la instrucción a la memoria	Tenemos una memoria que contiene las instrucciones, un registro con la dirección de memoria y un incrementador para calcular el nuevo valor del contador del programa. El resultado de esta etapa es la instrucción que tiene que ejecutarse, la cual deja un registro (IR)	Igual que en las instrucciones de cálculo.	Igual que las anteriores.
ETAPA DL Descodificación y lectura	Se utilizan dos registros como fuentes de operando, por lo que necesitamos un banco de registros con dos puertos de lectura y queda almacenado el resultado de la lectura en dos registros de segmentación R1 y R2.	Hay que leer los operandos necesarios para calcular la dirección de la memoria. Se pueden utilizar dos registros (ya que disponemos de dos caminos de lectura al banco de registros de las instrucciones de cálculo) o un registro y un desplazamiento.	Hay que leer un registro y un desplazamiento para la dirección de la memoria. Además hay que leer un registro que contiene el valor que queremos llevar a la memoria MDR_{out}, por lo que hay que añadir una nueva conexión desde uno de los caminos del banco de registros a un nuevo registro de segmentación MDR_{out2}
ETAPA UAL Cálculo de la operación	La UAL aplica la operación en los dos operandos, y queda guardado el resultado en un nuevo registro R3.	Tiene que calcularse la dirección de la memoria. Puede aprovecharse que la UAL que hemos puesto para hacer las operaciones de instrucciones de cálculo. El resultado de la UAL se guarda en un registro que servirá de índice en la memoria de datos (MAR).	Hay que calcular la dirección a la memoria, al igual que en las instrucciones LOAD. Hay que recoger el dato que tiene que llevarse a la memoria y trasladarla hacia la etapa siguiente con un nuevo registro de segmentación.
ETAPA MEM Memoria	Las instrucciones no hacen nada, de manera que el resultado de la etapa de UAL se copia en un nuevo registro R4.	Hay que acceder a la memoria de datos en la dirección indicada por el registro anterior MAR. Tenemos que añadir una memoria para los datos o un nuevo camino que contenga las instrucciones y los datos para poder acceder a la memoria y buscar una nueva instrucción en un mismo ciclo. El resultado de esta etapa es el dato que quiere llevarse de la memoria al banco de registros, y se guardará en MDR_{in}	Se accede a la dirección de la memoria de datos indicada en el registro de segmentación MAR y se lleva a la memoria el valor que queremos guardar por un nuevo camino.
ETAPA ESC Escritura	Se escribe en el banco de registros el resultado que se obtiene de la operación que se ha hecho en la UAL y que se encontraba en R4.	Se escribe el dato que hemos traído de la memoria al banco de registros aprovechando el camino de escritura que ya tenemos para las instrucciones de cálculo.	No se utiliza ningún recurso.

El camino de datos tiene estas cinco etapas diferentes, el hardware que configura cada una de las etapas tiene que estar separado de la etapa anterior y de la siguiente mediante registros para poder aislarlas. Así, cada etapa puede operar sobre una instrucción diferente sin afectar a las otras etapas.

En la página siguiente, podemos observar en otra tabla, el camino de datos segmentado para cada uno de estos tipos de instrucciones, con los recursos que se consumen en cada una de las etapas.

INSTR. CÁLCULO	ACCESO LOAD	ACCESO STORE
		

4. LIMITACIONES DE LA SEGMENTACIÓN

Esta técnica de la segmentación tiene unos límites, y es cuando se producen conflictos. Los **conflictos de datos** se producen cuando el resultado de la ejecución segmentada es diferente del resultado de la ejecución secuencial como resultado del orden en el que se han accedido a los datos.

Así queda claro que si una operación posterior quiere acceder a la lectura de un dato antes de que se haya escrito ese dato en una operación anterior que ha sido segmentada, cambiará obviamente el resultado que se obtiene. Hay tres tipos de **dependencia de datos**:

- **Dependencias de lectura**: Se producen cuando dos instrucciones diferentes leen algunas variables comunes.

```
Si: ADD R1, R2, R3;  R1 = R2 + R3
                …
Sj: SUB R4, R3, R5; R4 = R3 – R5
```

  En este ejemplo queda claro que si Si y Sj están demasiado juntas se produciría una dependencia de lectura, ahora bien, las dependencias de lectura nunca pueden provocar un conflicto de datos, ya que las instrucciones que tienen la dependencia solo comparten variables de lectura, independientemente del orden en que se hagan las lecturas, el resultado será el mismo.

- **Antidependencias**: Aparecen cuando una instrucción utiliza una variable que se vuelve a definir más tarde.

```
Si: ADD R1, R2, R3;  R1 = R2 + R3
                …
Sj: SUB R3, R4, R5; R3 = R4 – R5
```

  Tampoco en este caso se pueden producir conflicto de datos, ya que todas las instrucciones pasan por las mismas etapas y en el mismo orden, y todas las instrucciones con registros escriben el resultado cuando pasan por la

última etapa, así pues cuando la instrucción Sj escribe, todas las anteriores ya han leído correctamente sus operandos.

- **Dependencias de escritura**: Se producen cuando dos instrucciones escriben en alguna variable común.

```
Si: ADD R1, R2, R3;  R1 = R2 + R3
                …
Sj: SUB R1, R4, R5; R1 = R4 – R5
```

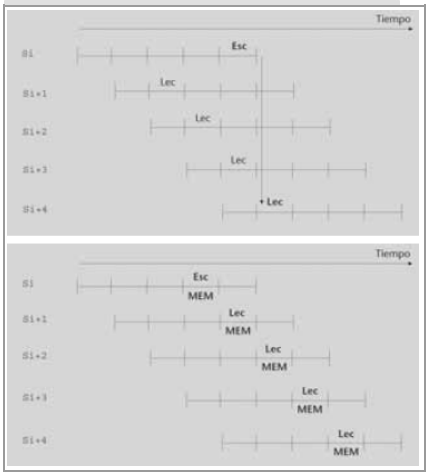
Tampoco en este caso se pueden producir conflicto de datos, ya que todas las instrucciones escriben en la misma etapa y cuando la instrucción Sj escribe, todas las anteriores ya han escrito.

- **Dependencias reales**: Tienen lugar cuando la instrucción Si genera una variable que es utilizada por la instrucción Sj.

```
Si: ADD R1, R2, R3;  R1 = R2 + R3
                …
Sj: SUB R4, R3, R1; R4 = R3 – R1
```

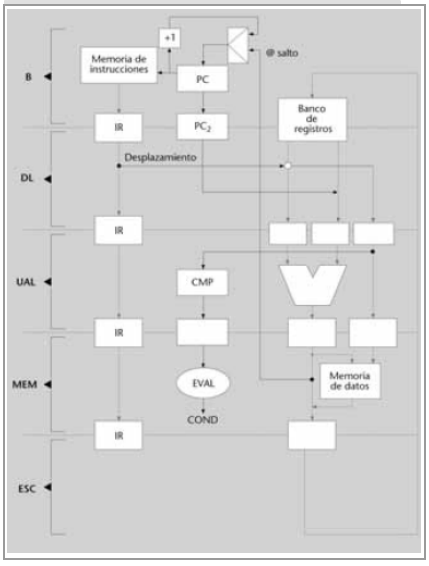
Éstas son las únicas dependencias de verdad que pueden acontecer en el procesador segmentado lineal. Hay diferencia si la variable que provoca la dependencia (en este caso R1) está en memoria o en un registro. Si está en un registro, la instrucción Sj debe encontrarse, al menos a una distancia de 4 instrucciones para que no provoque dependencia (ver figura adjunta). Si la variable problemática está en la memoria no puede haber conflicto de datos ya que la lectura y escritura siempre se produce en la misma etapa (la cuarta) del procesador segmentado lineal.

Dependencias reales
Variable en registro
Variable en memoria



Hasta ahora hemos considerado que las instrucciones se ejecutaban una detrás de otra, sin producirse saltos, pero como sabemos eso no siempre es así y esto nos introduce en un nuevo tipo de errores, los **conflictos de secuenciación**. Para añadir la secuenciación debemos hacer una serie de cambios en nuestro procesador segmentado lineal, así se podrá comparar el valor de un registro y en función del resultado saltar a una cierta distancia de la instrucción de salto:

Secuenciación



- La etapa de búsqueda de instrucciones y la de descodificación y lectura son las mismas que para las otras instrucciones.
- En la etapa UAL puede utilizarse la UAL del camino de datos para calcular el contador de programa de la instrucción siguiente en caso de que tenga que saltarse (PC = PC + X), también hay que comparar el registro leído con cero, y añadir un comparador en esta etapa del procesador.
- En el ciclo siguiente, en la etapa de memoria ya puede evaluarse la condición de salto y al final del ciclo todo está preparado (condición de salto y dirección de destino) para poder saltar.

Pero eso sí, si no se hace nada por evitarlo, durante 3 ciclos se habrá ido a buscar y se habrán ejecutado las tres instrucciones que siguen a la instrucción de salto; si finalmente se tiene que saltar porque se cumple la condición, habremos ejecutado erróneamente 3 instrucciones que con el método secuencial del código no se hubieran ejecutado y por tanto se habrá modificado el resultado del código original

Tanto para evitar este problema en los conflictos de secuenciación como en los conflictos de datos, se utiliza el **bloqueo del procesador**. Es una técnica que detiene la búsqueda de nuevas instrucciones y deja que las que están en las etapas

de UAL, memoria y escritura se continúen ejecutando. Los dos tipos de bloqueos de procesador que existen son:

- Para resolver los conflictos estructurales y de datos, además de detener la búsqueda de nuevas instrucciones, se detiene la etapa de descodificación y lectura de operandos; así la distancia entre las instrucciones que tienen dependencia o quieren usar un mismo recurso, aumenta hasta que desaparece el riesgo.
- Para resolver los conflictos de secuenciación no puede detenerse la descodificación y lectura de operandos, ya que tenemos que dejar que la instrucción de salto se ejecute para saber cuál es la instrucción siguiente que hay que buscar.

Durante estos ciclos que se bloquea el procesador, se generan instrucciones nulas, instrucciones NOP (no operación) que no modifican el estado del procesador.

5. REDUCCIÓN DE LOS CICLOS PERDIDOS POR CONFLICTOS DE DATOS

La solución del capítulo anterior es buena pero, por desgracia, nos hace perder 3 ciclos de proceso datos que es, en definitiva, para lo que segmentábamos el procesador, para poder ejecutar paralelamente más instrucciones por ciclo. Pero, como casi siempre, existen técnicas para reutilizar estos ciclos y que se sigan generando instrucciones, son dos principalmente, la planificación de instrucciones y los cortocirtuitos.

Planificación de instrucciones

1: ADD **R1**, R2, R3
2: SUB R4, R2, **R1**
3: ADD R5, R6, R2
4: MUL R7, R3, R6
5: ADD R2, R3, R2
6: SUB R3, R2, R4

1: ADD **R1**, R2, R3
3: ADD R5, R6, R2
4: MUL R7, R3, R6
2: SUB R4, R2, **R1**
5: ADD R2, R3, R2
6: SUB R3, R2, R4

La **planificación de instrucciones** es una técnica de software que se encarga de aplicar el compilador o el programador. Consiste en generar código teniendo en cuenta las dependencias de datos entre las instrucciones. Se basa en que cualquier par de instrucciones puede intercambiar el orden de ejecución si no tiene ningún tipo de dependencia entre ellas ni con las instrucciones que hay en medio. Pues bien, en los ciclos que se pierden por dependencia, en lugar de ejecutar instrucciones NOP, se ejecutan estas otras que realmente pertenecen al código y así no se pierden ciclos de procesador. En la figura lateral se observa un conjunto de instrucciones en las cuales, entre la primera y la segunda se perderían 3 ciclos de procesador, con una reorganización del mismo código, se obtiene el mismo resultado final y no se pierde ningún ciclo.

Renombramiento de registros

1: ADD R1, R2, R3
3: ADD R5, R6, R2
4: MUL R7, R3, R6
2: SUB R4, R2, R1
5: ADD **R2**, R3, R2
6: SUB R3, **R2**, R4

1: ADD **R1**, R2, R3
3: ADD R5, R6, R2
4: MUL R7, R3, R6
5: ADD **R20**, R3, R2
2: SUB R4, R2, **R1**
6: SUB R3, **R2**0, R4

En el ejemplo resuelto anteriormente, todavía se pierde un ciclo de procesador por culpa de la dependencia real entre las instrucciones 1 y 2. Todavía puede aplicarse otra técnica de software para intentar solucionarlo, el **renombramiento de registros**. Sólo con cambiar en nuestro ejemplo en la instrucción 5 y 6 el registro R2 por R20, podemos continuar reordenando instrucciones y separar aún más las que tienen dependencias reales.

Los **cortocircuitos** son una técnica de hardware que consiste en introducir cambios en el camino de datos y en el control del procesador segmentado que reducirán el número de ciclos perdidos a causa del conflicto de datos. Son conexiones especiales para los datos en el camino de datos que toman los mismos del punto donde se encuentran y las llevan donde se necesitan. Para ello debemos distinguir quienes son los productores de valores y quienes son los consumidores.

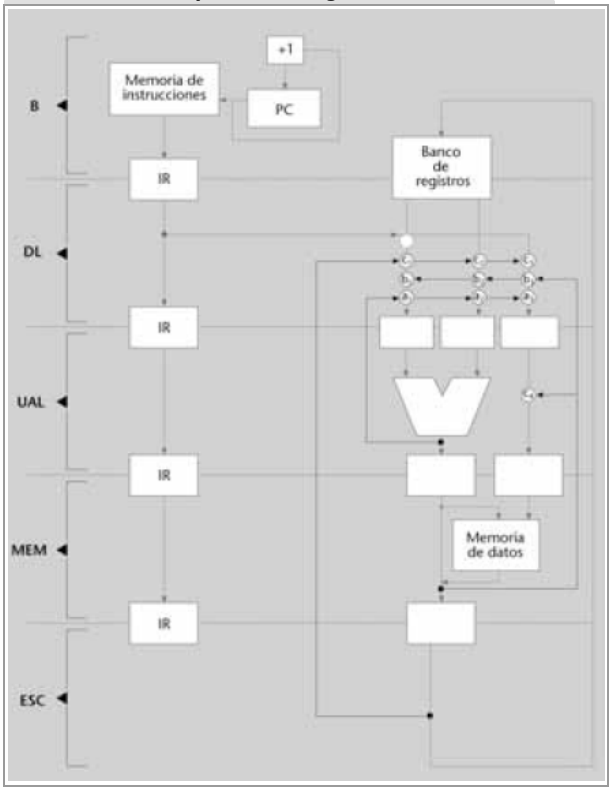
Los productores de valores se dan en la UAL (al ejecutar una instrucción de cálculo) y en la memoria de datos (al ejecutar una instrucción LOAD); estos valores pueden tomarse en cualquier punto entre el lugar que se calcula (UAL o memoria) y el lugar donde se escribirá (banco de registros), con lo que hay 3 lugares posibles donde pueden ir a buscarse valores: en la salida de la UAL (etapa UAL), en la salida de la memoria (etapa de memoria) o en la salida del registro de segmentación (etapa de escritura).

Los consumidores de valores son los que necesitan dichos valores para hacer un cálculo (UAL) o para almacenarlo (la memoria en la etapa STORE) y se pueden necesitar en los dos registros de entrada a la UAL y en el de salida de

valores hacia la memoria en la etapa de descodificación y lectura de operandos, o en el registro que propaga los valores que van a la memoria durante la etapa UAL.

Así, con todos estos aspectos, rediseñamos el procesador segmentado, añadiéndole nuevos caminos que son los cortocircuitos.

Cortocircuitos en el procesador segmentado



Aún así, en nuestro procesador segmentado sigue existiendo un caso de conflicto de datos que no puede resolverse con cortocircuitos y que nos hará bloquear el procesador y perder un ciclo de ejecución de instrucciones útiles; se trata de una instrucción LOAD seguida de cualquier instrucción a distancia 1 que utilice el registro que cargará la instrucción LOAD, que debe ser bloqueado para que dé un resultado correcto. Por ejemplo:

```
LOAD R1, R2, #13;    R1 = Memoria [R1 + R13]
ADD R4, R1, R5;        R4 = R1 + R5
SUB R7, R2, R1;         R7 = R2 - R1
```

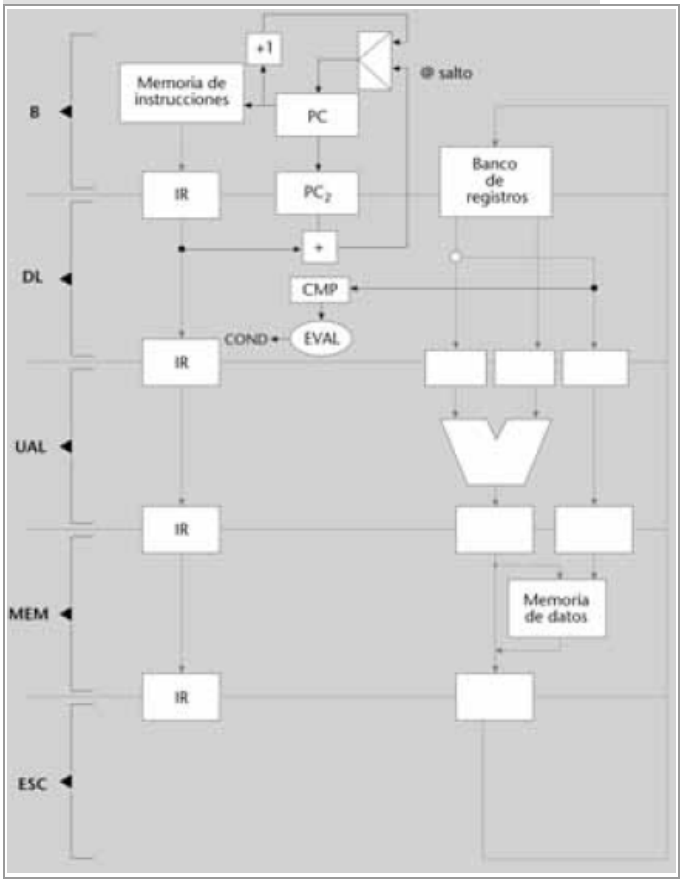
Esto sucede porque el valor que llega a la memoria está disponible al final de la cuarta etapa (MEM) de la ejecución de la instrucción LOAD, y la instrucción siguiente ADD necesita este valor al principio de la tercera etapa (UAL); se necesita pues antes de que el valor esté calculado y no hay ningún cortocircuito que pueda adelantar el valor del registro, perdemos por ello un ciclo de procesador.

6. REDUCCIÓN DE LOS CICLOS PERDIDOS POR CONFLICTOS DE SECUENCIACIÓN

Ya hemos visto la necesidad de bloquear el procesador con cada instrucción de salto para que el código se ejecute correctamente. Como por término medio una de cada 5 instrucciones son de salto y, en ese caso, perdemos 3 ciclos de procesador, los conflictos de secuenciación son realmente graves para el rendimiento del procesador.

El programador puede evitar esto introduciendo el menor número de instrucciones de salto posibles, pero por otro lado, existen otras 3 formas en el ámbito del hardware de mejorar este resultado.

Adelanto del cálculo de la condición y dirección de destino



- **Adelanto del cálculo de la condición y de la dirección de destino**: Se desplaza el hardware de la UAL necesario para que este cálculo tenga lugar en la etapa más temprana posible. Lo máximo que podemos hacer es efectuar el cálculo de la dirección de destino y evaluar la condición de salto en la etapa de descodificación y lectura. No podemos hacerlo antes porque entonces tendríamos que estar haciendo los cálculos sin saber todavía si se trata de una operación de salto o no. Destacamos en esta solución que hay que añadir un sumador extra al camino de datos del procesador segmentado para poder calcular la dirección de destino del salto y, que

en un ciclo es necesario tener tiempo para leer el banco de registros, comparar el registro y evaluar la condición; esto implica un tiempo de ciclo muy malo.

- **Predicción de saltos**: Consiste en empezar a ejecutar instrucciones (ya sea de las que siguen en secuencia a la instrucción de salto o de las instrucciones a partir de la dirección del salto) antes de acabar la ejecución de la instrucción de salto. Una vez que se ejecuta el salto, conviene saber si hemos tomado la decisión correcta respecto a las instrucciones a ejecutar (50% de probabilidades); para obtener la dirección de salto se utiliza una memoria caché de instrucciones de salto conocida como BTB, ahora bien, existen dos formas de predecir el salto:
    - **Estática**: La predicción para una instrucción de salto concreta es siempre la misma, bien porque todas las instrucciones de salto tienen la misma predicción (por ejemplo saltar siempre) o bien porque cada una de las instrucciones de salto tienen una predicción diferente.
    - **Dinámica**: La predicción puede variar a lo largo del tiempo en función de su comportamiento; así el Pentium III de Intel tiene u predictor de saltos dinámico que predice correctamente en torno al 90% de los saltos condicionales.

  Una vez producido o no el salto, si la predicción es correcta no se perderá ningún ciclo de procesador al ejecutar el salto, mientras que si es errónea se perderán tantos ciclos como en un procesador sin predicción. También hay que añadir que en el caso de un predictor estático en el que nunca se salte, no es necesaria siquiera memoria BTB, solo el fluir normal, sin bloquear, del procesador.
- **Saltos retardados**: Otra técnica consiste en hacer visible para el programador los ciclos que tarda en calcularse la dirección de destino y la condición de salto; así siempre se ejecutarán las n primeras instrucciones que hay después de una instrucción de salto, independientemente de si éste definitivamente se produce o no. Lo habitual es que tengan dos posiciones de retardo, por tanto se realizarán los dos ciclos siguientes del código.

7. ARQUITECTURA SEGMENTADA CON OPERACIONES MULTICICLO

El diseño del procesador que hasta ahora nos traíamos entre manos, complentaba una ejecución lineal con tiempo de ciclo constante que, para más INRI, tenía que ser el tiempo que tardase más. Si el objetivo es mejorar el rendimiento todo lo posible, interesa dividir la ejecución de las operaciones en las etapas que requiera cada tipo de operación. La segmentación ahora no será lineal y el diseño del procesador será más complejo.

Un ejemplo de esta necesidad son las operaciones en coma flotante, que son mucho más complejas que las operaciones con enteros. Así crearemos una máquina con dos procesadores, una será equivalente al procesador segmentado lineal que venimos conociendo en todo el tema, y la otra máquina ejecutará las nuevas instrucciones en coma flotante. En este nuevo esquema tendremos en cuenta que:

- Las operaciones con números enteros se efectúan de la misma forma que antes en el procesador segmentado lineal.
- Las operaciones en coma flotante se ejecutan en la nueva máquina con un banco de registros propios.
- Las operaciones de acceso a la memoria serán las únicas que relacionan ambas máquinas y se ejecutarán siempre en la parte de los enteros.
- El cálculo de las direcciones de la memoria siempre tiene lugar en la parte de los enteros y se requiere leer un registro del banco de registros enteros.

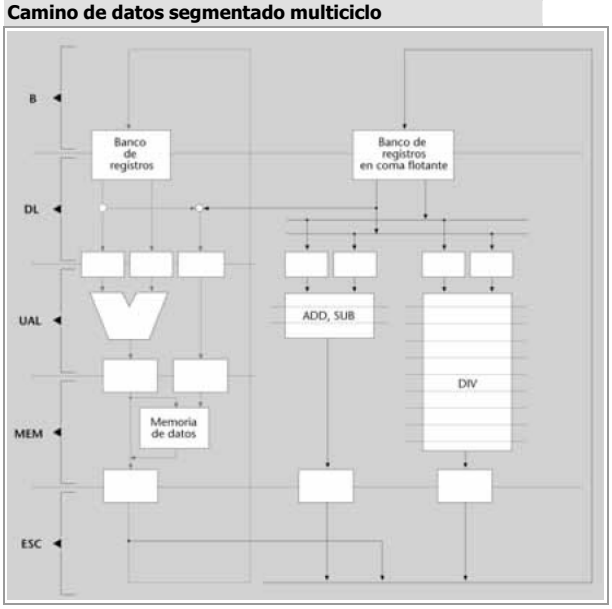
Obviamente estos cambios también influyen en las operaciones de cálculo apareciendo nuevas operaciones como FSUM para la suma en coma flotante. Las instrucciones además tienen un número de etapas que dependen de su dificultad,

así las operaciones de suma y resta se ejecutan en 3 ciclos, las de multiplicación en 5 y la de división en coma flotante, la más compleja, en 8 ciclos.

En este procesador segmentado con operaciones multiciclo ahora la nueva máquina puede tener también dependencias que estudiamos a continuación:

- Antidependencias: Todas las operaciones son idénticas hasta la etapa de lectura, como siempre se escribe después de leer (igual al procesador segmentado lineal) estas dependencias no pueden crear nunca un conflicto de datos.
- Dependencias reales: Una instrucción puede intentar leer un registro que ha calculado una instrucción anterior pero que todavía no se ha escrito, en este caso con bloqueos y pérdida de ciclos superamos el problema; reordenando las instrucciones y utilizando cortocircuitos evitamos este problema.
- Dependencias de salida: Como las operaciones pueden terminar en un orden diferente a aquél en el que se empezaron a ejecutar, es necesario en algunas ocasiones bloquear el procesador o anular la escritura más antigua.

Camino de datos segmentado multiciclo



8. TEMAS AVANZADOS

Dos técnicas que actualmente suelen aplicar todos los procesadores para mejorar su rendimiento:

- **Ordenación dinámica de las instrucciones**: Ya hemos comprobado anteriormente que cuando se produce un conflicto de datos que no puede resolverse, el procesador se bloquea y perdemos ciclos de trabajo. Pero si sabemos que las instrucciones que siguen al bloqueo se pueden ejecutar y son independientes y no dan lugar a ningún conflicto, entonces podremos ejecutarlas durante el tiempo de bloqueo. Lo que hacen los procesadores actuales es guardar dentro del procesador la instrucción bloqueante y se continúan buscando instrucciones, así que el procesador sólo se bloqueará cuando ya no pueda guardar más instrucciones en su interior. Lógicamente con esta nueva técnica pueden surgir nuevos conflictos, que se solucionan con un renombramiento de registros, así el procesador puede funcionar como si el compilador hubiera generado código para un procesador con infinitos registros. Lo negativo de esta técnica de ordenación dinámica de instrucciones con renombramiento de registros es que complica el diseño del control y empeora el tiempo de ciclo del procesador.
- **Ejecución superescalar de las instrucciones**: Se ejecutan más de una instrucción por ciclo de procesador, aprovechándonos de la técnica de multiplicación para explotar el paralelismo a nivel de instrucción. El grado de un procesador segmentado superescalar es el número de instrucciones que puede ejecutar por ciclo y que actualmente es de 2 a 4.

Ambos métodos anteriores pueden dar problemas en el **tratamiento de las excepciones**, dado que al no ejecutar el código línea a línea, cuando se presenta una excepción no puede detenerse para tratarla y luego continuar por el mismo punto en que se detuvo sin que ya se hayan ejecutado varias instrucciones intermedias.