2

PROCESADORES SEGMENTADOS

2.1. Introducción y definiciones

La segmentación (*pipelining*) es una técnica de implementación de procesadores que desarrolla el paralelismo a nivel **intrainstrucción**. Mediante la segmentación se puede solapar la ejecución de múltiples instrucciones. El procesamiento segmentado aprovecha la misma filosofía de trabajo de la fabricación en cadena: cada etapa de la segmentación (o **segmento**) completa una parte (subtarea) de la tarea total. Los segmentos están conectados cada uno con el siguiente, de forma que la salida de uno pasa a ser la entrada del siguiente. Así, los segmentos configuran un cauce a través del que se va procesando la tarea deseada (por ello, en algunas ocasiones, a los procesadores segmentados también se les llama **procesadores encauzados** o, simplemente, **cauces**). Lo más interesante de la segmentación es que las diferentes subtareas pueden procesarse de forma simultánea, aunque sea sobre diferentes datos.

Una contribución clave de la segmentación es la posibilidad de comenzar una nueva tarea sin necesidad de que la anterior se haya terminado. La medida de la eficacia de un procesador segmentado no es el tiempo total transcurrido desde que se comienza una determinada tarea hasta que se termina (**tiempo de latencia del procesador**), sino el tiempo máximo que puede pasar entre la finalización de dos tareas consecutivas.

Evidentemente, la segmentación no es posible sin incrementar los recursos de proceso, de la misma manera que la fabricación en cadena no es posible sin aumentar el número de operarios. Por ello, los procesadores segmentados necesitan redundancia de muchos recursos: registros, circuitos aritméticos, etc.

Considérese una tarea, compuesta por n subtareas. Si estas subtareas se procesan de forma totalmente secuencial, el tiempo necesario para procesar la tarea total será la suma de los tiempos necesarios para la terminación de cada una de las subtareas (véase la figura 2.1). En este esquema, T_j^i representa la subtarea j dentro de la tarea i). Por otra parte, para comenzar el tratamiento de una nueva tarea será necesario esperar ese mismo tiempo. Esto se debe a que habrá algunas unidades funcionales que serán necesarias para llevar a cabo varias de las subtareas y, por ello, esas subtareas no podrán superponerse en el tiempo.

Departamento de Informática Universidad de Valladolid

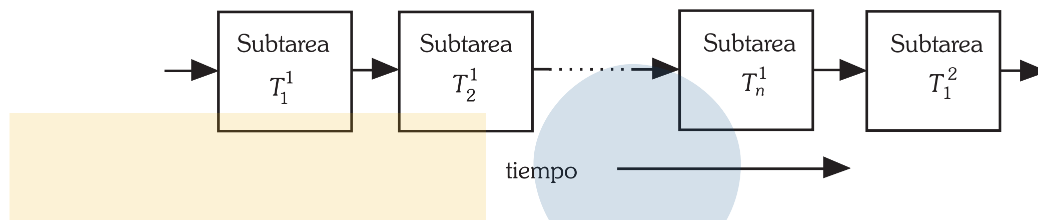


Fig. 2.1. Tarea procesada de forma totalmente secuencial.

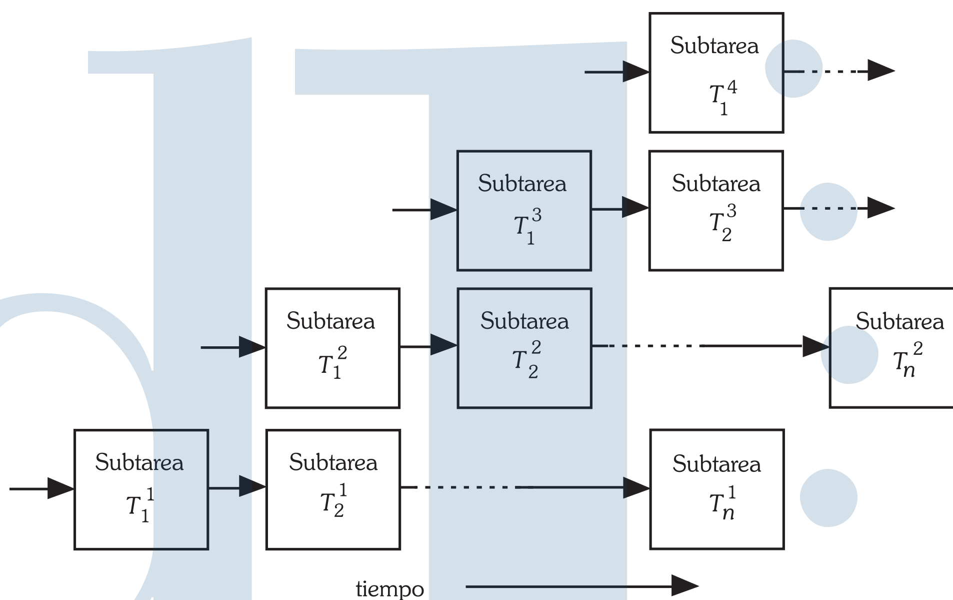


Fig. 2.2. Tarea ejecutada mediante un procesador segmentado.

Si, para procesar esa misma tarea, se emplea un procesador segmentado, basta que se haya terminado la primera subtarea para poder empezar a procesar una nueva tarea (véase la figura 2.2). En la citada figura puede verse el continuo flujo de tareas que se van procesando a través de los n segmentos encargados de procesar cada una de las subtareas. Puede observarse que el tiempo total de procesamiento de una tarea completa puede ser el mismo, aunque frecuentemente será mayor, que el tiempo empleado para el procesamiento secuencial de la misma tarea mostrado en la figura 2.1. Esto, sin embargo, carece de importancia, ya que lo verdaderamente importante es el ritmo al que las tareas van saliendo del procesador (**velocidad de emisión de tareas**). Al número de segmentos del procesador, n, se le llama, en muchas ocasiones, **profundidad de la segmentación**.

Para que el tiempo de latencia del procesador segmentado sea el mínimo posible, es necesario que el procesador esté **equilibrado**, es decir, que todas las subtareas en que se ha dividido la tarea total tarden en procesarse el mismo tiempo. Esto es debido a que las tareas no podrán evolucionar al segmento siguiente hasta que no se haya terminado la subtarea más lenta. Por ello, si el procesador no está equilibrado, los segmentos más rápidos estarán cierto tiempo sin hacer trabajo alguno, lo que disminuirá el rendimiento del procesador.

La **relación de precedencia** de un conjunto de subtareas $T_1, ..., T_n$, que componen cierta

tarea T, especifica para cada subtarea T_j, que ésta no puede comenzar hasta que haya terminado ciertas subtareas T_i. Las relaciones de precedencia para todas las subtareas de T forman su **grafo de precedencia**.

En el ejemplo de la figura 2.2 se ha supuesto que las tareas que se procesan en el cauce tienen un **grafo de precedencia lineal**. Esto significa que una subtarea T_j no puede comenzar hasta que todas las subtareas previas, es decir $T_i, \ \forall i < j$, hayan finalizado. A los procesadores segmentados que sólo pueden procesar tareas con grafo de precedencia de este tipo, se les denomina **cauces lineales**. Más adelante veremos casos de procesadores segmentados en que se pueden procesar tareas con grafo de precedencia no lineal. En este tipo de cauces, algunas de las subtareas pueden ejecutarse varias veces para procesar la tarea total. Ello implica que alguna de las etapas puede utilizarse varias veces, provocando una **realimentación** dentro del cauce.

2.2. Rendimiento de los procesadores segmentados

Para estudiar el rendimiento de los procesadores segmentados empezaremos por calcular la ganancia de velocidad ideal en un procesador de este tipo. Esta ganancia ideal sólo se podrá obtener si el procesador está equilibrado y, además, si no se pierden ciclos de reloj. La ganancia de velocidad en un procesador segmentado de n segmentos vendrá dada por la relación entre el tiempo de ejecución de m tareas en un procesador convencional, $t(1)$, y el tiempo de ejecución de esas mismas m tareas, en un procesador segmentado, $t(n)$. En las mejores condiciones, y si el procesador está equilibrado, se puede suponer que el periodo de reloj es el mismo en ambos casos. Observando la figura 2.1, podemos deducir fácilmente que el tiempo empleado en ejecutar m tareas en un procesador convencional, medido en ciclos de reloj, será

$$t(1) = nm$$

Por otra parte, para un procesador segmentado, el tiempo empleado en ejecutar esas m tareas tendrá dos partes: el tiempo empleado en llenar el cauce, o lo que es lo mismo, el tiempo empleado en ejecutar la primera tarea (n ciclos) y, luego un ciclo de reloj por cada una de las tareas restantes ($m - 1$ ciclos). Por tanto:

$$t(n) = n + (m - 1) \qquad [2.1]$$

Con estos datos, podemos deducir que la ganancia de velocidad de un procesador segmentado, respecto a un procesador convencional, es

$$S(n)_{ideal} = \frac{t(1)}{t(n)} = \frac{nm}{n + m - 1} \qquad [2.2]$$

Cuando el número de tareas ejecutadas, m, sea muy grande, es decir, cuando haya pasado un poco de tiempo, tendremos que la ganancia estacionaria será:

$$S(n)_{\infty} = \lim_{m\to\infty} S_{ideal} = \lim_{m\to\infty} \frac{nm}{n + m - 1} = \lim_{m\to\infty} \frac{n}{\frac{n}{m} + \frac{m-1}{m}} = n \qquad [2.3]$$

Comparando esta expresión con la ecuación 1.4 vemos que la ganancia ideal estacionaria obtenida con un procesador segmentado de n etapas en la misma que la que se consigue con un sistema paralelo de n procesadores no segmentados.

Evidentemente, esta ganancia ideal no se consigue nunca por diversas razones:

Departamento de Informática Universidad de Valladolid

- Las **detenciones del cauce**, causadas porque, en ciertas condiciones, el cauce no puede avanzar en alguna de sus etapas y ello provoca la parada de la cadena de proceso desde la etapa detenida hacia atrás. Para entender el concepto de detención, recuérdese la idea de la fabricación en cadena: si alguna de las etapas de la cadena deja de cumplir su función, la cadena se parará desde ese punto hacia atrás hasta que la etapa detenida pueda volver a funcionar. En sucesivos apartados se verán diversas causas que pueden provocar detenciones; por ahora estudiaremos la incidencia de las detenciones sobre el rendimiento de los procesadores segmentados.

  Las detenciones causarán una pérdida de cierto número de ciclos de reloj que reducirán la ganancia ideal calculada en 2.3. Esto se debe a que el número de ciclos de reloj necesarios para procesar la m tareas con el procesador segmentado $t(n)$ aumentará en el número de ciclos perdidos por detención. Por tanto, si se han perdido p ciclos por detenciones, la expresión 2.2 se transformará ahora en

$$S(n) = \frac{t(1)}{t(n)} = \frac{nm}{n+m-1+p} \qquad [2.4]$$

  que, en la situación estacionaria alcanzará el valor

$$S(n)_\infty = \lim_{m\to\infty} S = \lim_{m\to\infty} \frac{nm}{n+m-1+p} = \lim_{m\to\infty} \frac{n}{\frac{n}{m} + \frac{m-1}{m} + \frac{p}{m}} = \frac{n}{1+\overline{p}} \qquad [2.5]$$

  En esta expresión, el cociente $\overline{p} = p/m$ representa el número medio de ciclos por tarea perdidos por detenciones.

- El **desequilibrio entre los segmentos**, que causa que el ciclo de reloj de la máquina segmentada tenga que ser mayor que el de una máquina equivalente no segmentada. El periodo de reloj en un procesador segmentado no equilibrado será:

$$T_s = \text{máx}(T_i)_{i=1,..,n}$$

  donde T_i es el tiempo necesario para procesar cada subtarea. Si llamamos T al periodo de la máquina equivalente no segmentada, tendremos que incluir este factor en la expresión de la ganancia de velocidad, ya que ahora el ciclo de reloj no es homogéneo y no nos sirve como unidad de medida. Tendremos entonces, que los tiempos necesarios para procesar m tareas en ambas máquinas serán, en segundos,

$$t(1) = nmT$$
$$t(n) = (n+m-1+p)T_s \qquad [2.6]$$

  Por tanto, la expresión 2.5, que nos da la ganancia de velocidad, deberá corregirse ahora en la forma siguiente:

$$S(n)_\infty = \lim_{m\to\infty} S = \lim_{m\to\infty} \frac{nmT}{(n+m-1+p)T_s} = \frac{T}{T_s}\frac{n}{1+\overline{p}} \qquad [2.7]$$

Departamento de Informática
Universidad de Valladolid

Para calcular la eficiencia de un procesador segmentado, aplicaremos la expresión 1.5 que expresa la relación entre la ganancia de velocidad conseguida y la ideal:

$$E(n) = \frac{S}{S_{ideal}} = \frac{S}{n} = \frac{mT}{(n+m-1+p)T_s}$$

que para valores grandes de m se transformará en

$$E(n)_\infty = \lim_{m\to\infty} E = \lim_{m\to\infty} \frac{mT}{(n+m-1+p)T_s} = \frac{T}{T_s}\frac{1}{1+\overline{p}} \quad [2.8]$$

Definiremos la **productividad** (*throughput*) de un procesador segmentado como el *número de tareas que puede completar por unidad de tiempo*. Ésta es una medida más absoluta que las anteriores y mide la potencia global de cálculo del procesador. Podremos calcular la productividad dividiendo el número de tareas emitidas por el tiempo empleado en procesarlas. Teniendo en cuenta la ecuación 2.6, la productividad (H) vendrá dada por

$$H = \frac{m}{(n+m-1+p)T_s}$$

que para el estado estacionario será

$$H_\infty = \lim_{m\to\infty} H = \lim_{m\to\infty} \frac{m}{(n+m-1+p)T_s} = \frac{1}{(1+\overline{p})T_s}$$

2.3. Clasificación de los procesadores segmentados

Puede establecerse una clasificación de los procesadores segmentados atendiendo al uso que se da a la segmentación. Esta clasificación fue propuesta por Händler (1977):

Segmentación aritmética: La ALU de un computador puede segmentarse para la ejecución de algoritmos aritméticos complejos. La segmentación aritmética es muy útil para procesar instrucciones vectoriales, es decir, operaciones que deben repetirse de la misma forma sobre todas las componentes de un vector; esto se consigue provocando que un segmento de la unidad aritmética trabaje sobre una de las componentes, mientras los demás trabajan sobre las componentes siguientes, realizándose así las diferentes subfunciones necesarias para la ejecución de la operación aritmética. En la actualidad, este tipo de segmentación se emplea en muchos procesadores para ejecutar operaciones aritméticas con números representados en punto flotante. Un ejemplo clásico de este tipo de procesadores es el multiplicador segmentado basado en un árbol de Wallace, tal como se muestra en la figura 2.3. En esta figura, los bloques CSA representan a sumadores con salvaguarda de llevadas y los bloques CLA, a sumadores con generación anticipada de llevadas. Las líneas inclinadas hacia la izquierda indican que los vectores de llevadas deben desplazarse a la izquierda un lugar antes de entrar en la etapa siguiente (véase, por ejemplo, (Bastida, 1995)). Por otra parte, es necesario señalar que las sumas deben efectuarse con extensión de signo y que es necesario dejar bits de guarda para tener en cuenta las posibles llevadas salientes. Las cajas sombreadas representan **registros cerrojos** (***latchs***) que, en este caso, se llaman

Departamento de Informática Universidad de Valladolid

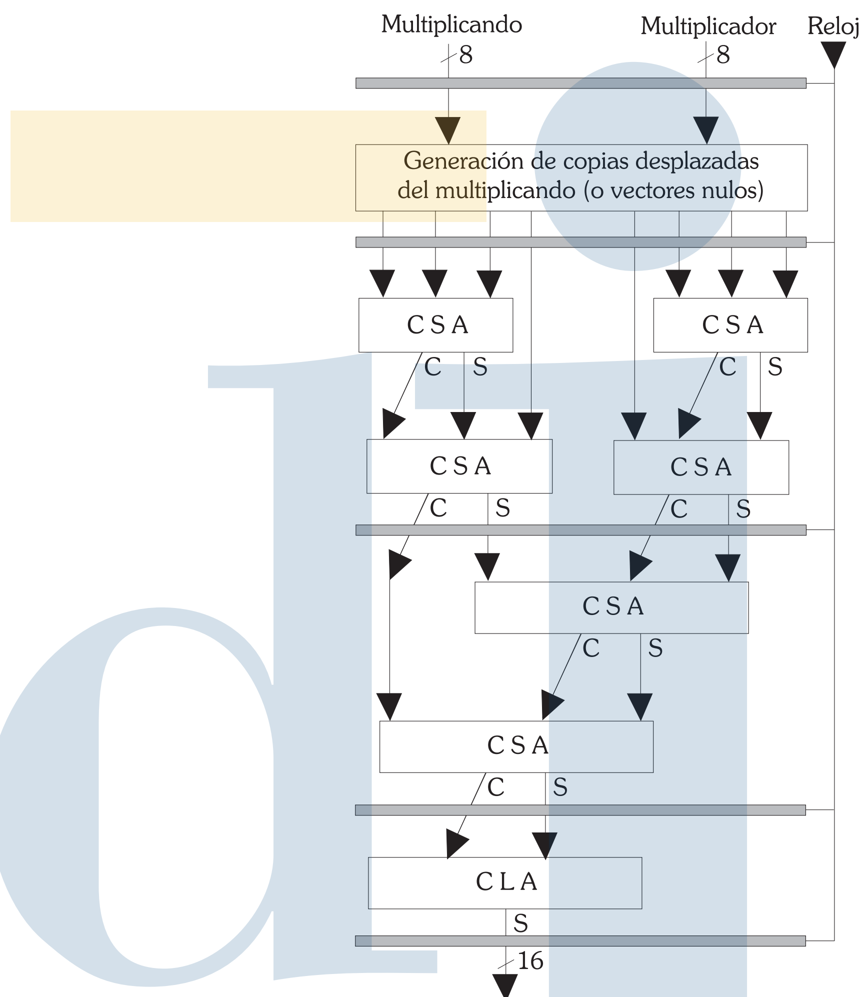


Fig. 2.3. Multiplicador segmentado basado en un árbol de Wallace.

registros de segmentación. Estos registros retienen las informaciones salientes de cada etapa durante un ciclo de reloj para que se mantengan en la entrada de la siguiente etapa. Se ha supuesto que el retardo del sumador con anticipación es doble que el del sumador con salvaguarda de llevadas.

Segmentación de instrucciones: La ejecución de un flujo de instrucciones puede adoptar una estructura segmentada que permita el solapamiento de la ejecución de una instrucción con la lectura, decodificación, búsqueda de operandos, etc. de las instrucciones siguientes. Esta técnica también se denomina **anticipación de instrucciones** (***instruction lookahead***). En la figura 2.4 se muestra un ejemplo de segmentación de instrucciones. La citada figura corresponde a un computador de tipo registro-registro, es decir, un computador en que los accesos a memoria están restringidos a instrucciones específicas (LOAD y STORE), y en que el modo de direccionamiento empleado para estos accesos a memoria y para las bifurcaciones, es el relativo al contador de programa. Se supone que el computador dispone de arquitectura Harvard, es decir, que posee memorias caché separadas para código

Departamento de Informática Universidad de Valladolid

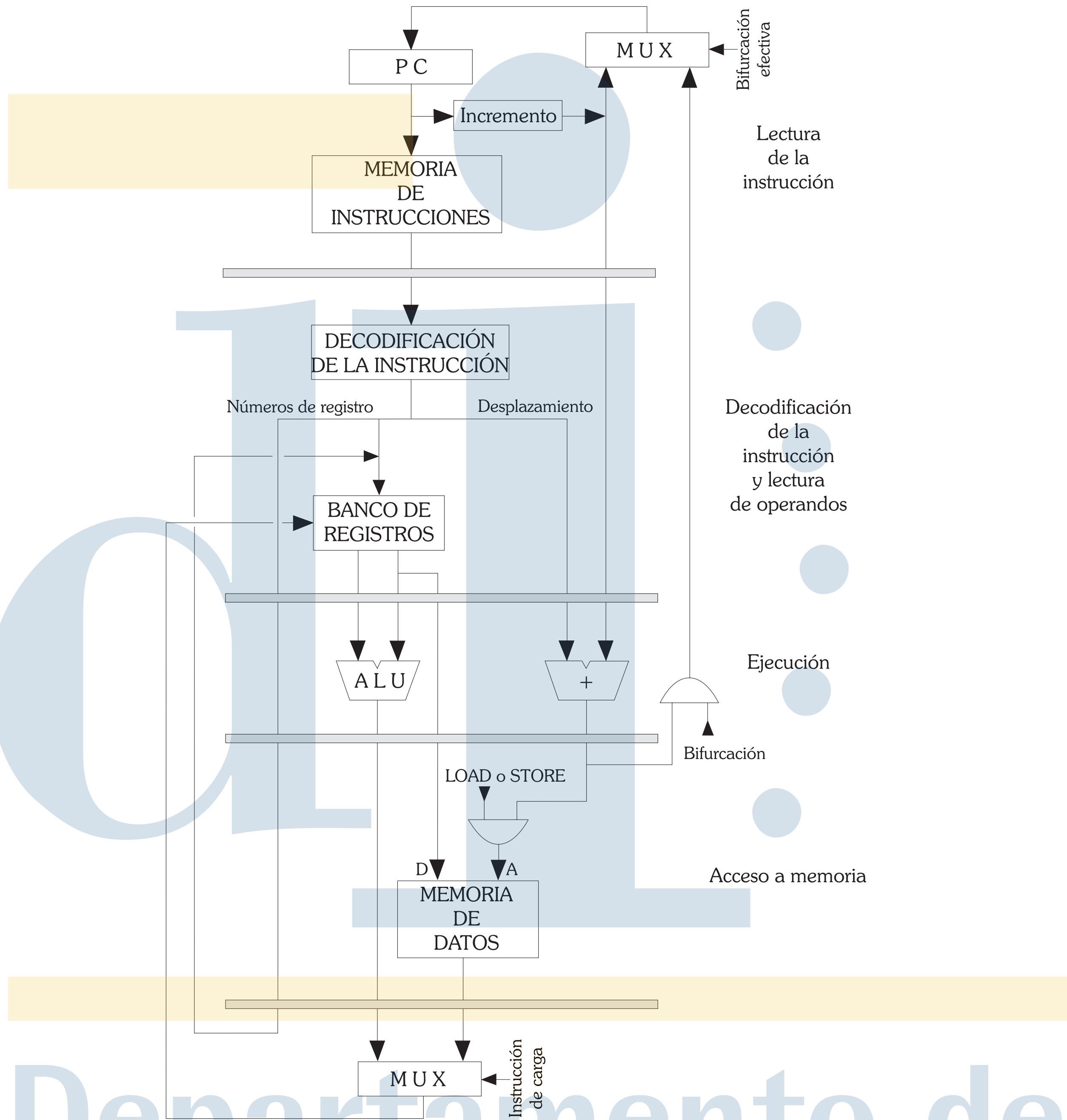


Fig. 2.4. Ejemplo de segmentación de instrucciones.

y datos. En el ejemplo, se ha dividido la ejecución completa de cada instrucción en cuatro segmentos:

A. Lectura.

B. Decodificación y lectura de operandos.

C. Ejecución.

Departamento de Informática Universidad de Valladolid

D. Acceso a la memoria de datos (si es necesario).

Prácticamente, todos los computadores actuales disponen de segmentación de instrucciones.

Segmentación de procesadores: Este tipo de procesamiento se produce cuando el mismo flujo de datos es tratado por una serie de procesadores, de forma que cada uno de ellos efectúe una subtarea del proceso total. Cada procesador dejará sus resultados en una memoria, que también será accesible desde el siguiente, para que éste procese esos resultados para ejecutar la siguiente subtarea sobre el flujo de datos. Este tipo de segmentación se emplea solamente en procesadores vectoriales de muy altas prestaciones.

Por otra parte, los procesadores segmentados pueden ser tanto **monofunción** como **multifunción**. Los primeros sólo pueden realizar una función fija; por el contrario, los procesadores segmentados multifunción pueden efectuar diferentes funciones, en instantes distintos, conectando de formas diferentes los segmentos del cauce.

Los procesadores segmentados multifunción son **estáticamente configurables** si no pueden cambiar su configuración mientras se está ejecutando una función; esto implica que no pueden ejecutarse varias funciones diferentes al mismo tiempo. Por el contrario son **dinámicamente configurables** si pueden cambiar su configuración en cualquier momento; por ello, los cauces de este tipo pueden mantener varias configuraciones simultáneas y, por tanto, ejecutar varias funciones diferentes a la vez.

Para que estos conceptos puedan apreciarse más claramente, pondremos como ejemplo el procesador segmentado que se mostró en la figura 2.4. En este cauce podrían ejecutarse diferentes clases de instrucciones, utilizando cada una de ellas diferentes etapas: por ejemplo, una instrucción de bifurcación sólo emplearía las etapas A, B y C; sin embargo, una instrucción de almacenamiento de datos en memoria (STORE) emplearía las etapas A, B, C y D; por otra parte, una instrucción de carga (LOAD) emplearía las etapas A, B, C, D y escribiría el resultado en uno de los registros del banco, por lo que volvería a emplear parte de la etapa B; por último, una instrucción aritmética pasaría por las etapas A, B, C y depositaría el resultado en un registro, por lo que tendría que emplear nuevamente la etapa B.

Un cauce multifunción sería capaz de efectuar todas las operaciones anteriores con los mismos segmentos, combinándolos de forma diferente en cada caso. Si este cauce pudiera estar ejecutando al mismo tiempo diferentes clases de instrucciones (evidentemente en diferentes etapas), sería un cauce dinámicamente configurable.

2.4. Conflictos y sus tipos

Hay circunstancias que pueden disminuir el rendimiento de un procesador segmentado debido a que provocan la detención del cauce. Estas circunstancias se denominan **riesgos** o **conflictos**. Existen tres clases de conflictos (Hennessy & Patterson, 2003):

Conflictos estructurales: son detenciones producidas en el procesador por insuficiencia del hardware, debido a que una etapa no puede avanzar porque el hardware necesario está siendo utilizado por otra.

Departamento de Informática Universidad de Valladolid

Conflictos por dependencia de datos: surgen, principalmente, cuando una instrucción necesita los resultados de otra anterior, que todavía no los tiene disponibles, por no haberse terminado de ejecutar completamente.

Conflictos de control: se deben a las instrucciones de control de flujo, en que no se puede leer la instrucción siguiente hasta que no se conozca su dirección.

2.5. Control de conflictos

En esta sección estudiaremos las técnicas para detectar, prevenir o evitar los diferentes tipos de conflictos; también veremos la forma de conseguir que causen la menor pérdida posible de rendimiento una vez que ya se han producido.

2.5.1. Conflictos estructurales

Los conflictos estructurales, en los cauces monofunción, se resuelven, si ello fuera posible, aumentando las posibilidades del hardware duplicando todos los recursos que sea necesario. Uno de los conflictos estructurales más frecuentes en los cauces monofunción son los relacionados con los accesos a memoria; por ejemplo, se producirá un conflicto cuando una etapa trate de acceder a memoria para leer una instrucción y otra lo haga para acceder a un dato. Estos conflictos se resuelven, en primer lugar, mediante la **arquitectura Harvard** en que existen memorias caché diferenciadas para código y datos. Aún así, podría mantenerse algún conflicto, por accesos simultáneos a datos por parte de varios segmentos. Esto se puede evitar agregando nuevos puertos a la caché de datos o, también, limitando los accesos a datos a determinadas instrucciones para que sea difícil que se superpongan (arquitecturas **registro-registro** o de **carga-almacenamiento**). También existe la posibilidad de que el compilador tenga en cuenta la estructura del procesador, detecte este tipo de conflictos y, cuando ello ocurra, intente modificar el orden de ejecución de las instrucciones, siempre que sea posible, para intentar evitarlos.

En los cauces no lineales, el control de estos conflictos adquiere una nueva dimensión. El problema todavía se agrava más en los cauces multifunción dinámicamente configurables. En estos cauces existe la posibilidad del uso simultáneo de varios segmentos por parte de ejecuciones distintas de la misma función o por varias de las funciones. Este tipo de conflictos estructurales se denominan **colisiones**. El control de colisiones se hace más complejo si el grafo de precedencia de las tareas no es lineal. Las técnicas de prevención de colisiones se basan en las **tablas de reservas** (Davidson, 1971).

Una tabla de reservas contiene la información sobre la ocupación de cada segmento en cada uno de los ciclos máquina hasta que termine de ejecutarse cierta tarea. Las filas de la tabla representan a cada uno de los segmentos y las columnas representan los ciclos necesarios para la ejecución de la tarea. Para un cauce monofunción y lineal, la tabla de reservas sólo tendrá marcadas las casillas de la diagonal, sin embargo, en los cauces no lineales, la construcción de la tabla no será tan simple. Un ejemplo de este tipo de tablas puede verse en la figura 2.5 para las instrucciones aritméticas y de carga en el procesador segmentado de la figura 2.4.

Para simplificar el estudio del control de las colisiones consideraremos inicialmente un procesador segmentado monofunción. Tomaremos como ejemplo la función cuya tabla de reservas

Departamento de Informática
Universidad de Valladolid

	1	2	3	4
A	X			
B		X		X
C			X	
D				

(a)

	1	2	3	4	5
A	X				
B		X			X
C			X		
D				X	

(b)

Fig. 2.5. Tabla de reservas para las instrucciones aritméticas (a) y de carga (b), en el procesador segmentado de la figura 2.4

	1	2	3	4	5	6
A	X					X
B		X				X
C			X			
D				X	X	

(a)

10011

(b)

Fig. 2.6. (a) Ejemplo de tabla de reservas y (b) vector de colisiones correspondiente a dicha tabla.

se muestra en la figura 2.6(a). El problema que se nos plantea es: si quisiéramos efectuar dos veces sucesivas la misma tarea, ¿Cuántos ciclos máquina tendríamos que esperar para arrancar dicha tarea por segunda vez? La solución es sencilla: basta superponer a la tabla de reservas original, la misma tabla desplazada un lugar a la derecha. Si alguna de las etapas está ocupada en el mismo ciclo por la tabla original y por la desplazada, significa que no podrá ejecutarse la tarea la segunda vez, arrancada sólo un ciclo más tarde, por existir una colisión. Se debe intentar superponer la tabla desplazándola más lugares hacia la derecha hasta que no haya ningún ciclo que esté ocupado en ambas tablas al mismo tiempo. El número de desplazamientos entre ambas tablas nos dará el número de ciclos que habrá que esperar (también denominado **latencia**) para arrancar la tarea por segunda vez sin que haya colisiones. A las latencias en que ocurre colisión se les denomina **latencias prohibidas**. Precisamente a partir de esta idea, se construye el **vector de colisiones** que es *un vector binario cuya componente i es 1 si, y sólo si, se produce una colisión cuando se arranca la tarea la segunda vez i ciclos después de haberse iniciado la primera*. En el caso de la tabla de reservas anterior, el vector de colisiones se muestra en la figura 2.6(b).

Como fácilmente puede comprenderse, el número de componentes del vector de colisiones

di: Departamento de Informática Universidad de Valladolid

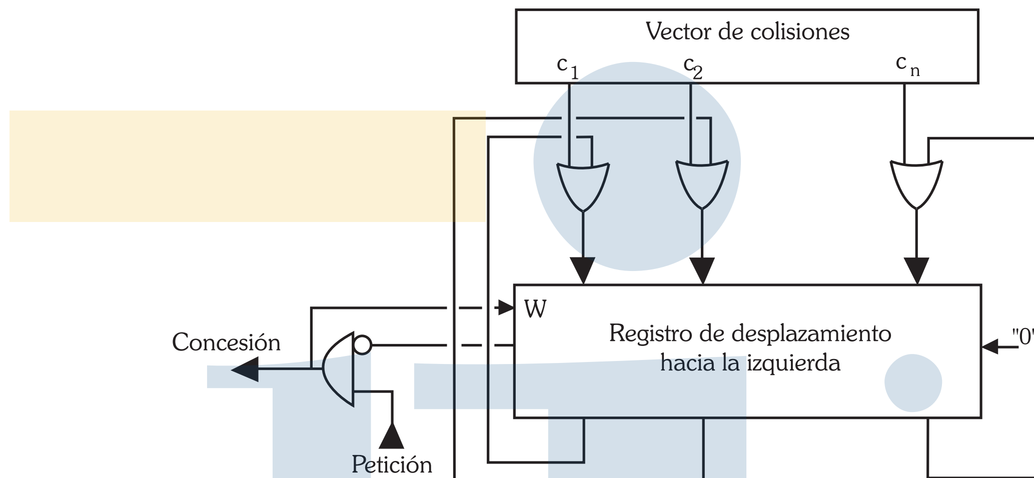


Fig. 2.7. Sistema de prevención de colisiones para un cauce monofunción.

estará acotado superiormente por el número de ciclos necesarios para completar la ejecución de la tarea menos uno, ya que no se contempla la colisión de la ejecución de una tarea consigo misma; en cualquier caso, es posible omitir las últimas componentes nulas del vector.

Ahora estudiaremos la forma en que puede ayudarnos en la práctica el vector de colisiones para prevenir éstas. Hay que tener en cuenta que los cálculos necesarios para esa prevención deben hacerse de forma simultánea y en el mismo tiempo del ciclo máquina; por ello, los circuitos necesarios deben ser muy sencillos y rápidos. En la figura 2.7 se muestra el esquema de principio de un circuito para la prevención de las colisiones. Como puede observarse, este circuito basa su funcionamiento en un registro de desplazamiento hacia la izquierda con posibilidad de carga paralela. El funcionamiento comprende los siguientes pasos (Davidson, 1971):

1. Al principio de cada ciclo, se provoca un desplazamiento del registro hacia la izquierda.
2. Una petición de acceso al cauce debe concederse si el bit saliente del registro de desplazamiento es 0; por el contrario, debe denegarse si el bit saliente es 1.
3. Ahora actuaremos de forma diferente en función de la concesión o no del acceso al cauce:
   - Si una petición ha sido concedida, se efectúa una operación OR entre el contenido del registro de desplazamiento y el vector de colisiones, el resultado de esta operación se carga en el registro.
   - Si, por el contrario, la petición no ha sido concedida, no se hará ninguna operación sobre el registro de desplazamiento y sólo se retendrá la petición para repetirla en el ciclo siguiente.
4. Se vuelve al primer paso del proceso.

El número de ciclos que habrá que esperar, en caso de denegarse la solicitud de acceso, no podrá ser mayor que la longitud del vector de colisiones, ya que en ese número de ciclos el registro de desplazamiento se habrá vaciado.

Departamento de Informática Universidad de Valladolid

La estrategia de control que se ha expuesto se basa en comenzar una nueva tarea en el primer ciclo de reloj en que no haya conflicto con las tareas ya comenzadas. A este tipo de estrategia se le denomina **estrategia avara** (***greedy strategy***) debido a que se trata de arrancar una nueva tarea tan pronto como sea posible. Esta estrategia es fácil de implementar, como se acaba de ver, pero no es necesariamente la que proporciona un mejor rendimiento. Si la estrategia avara fuera la mejor, y muchas veces lo es, se dará el caso de que podremos implementar una estrategia con buen rendimiento por procedimientos sencillos. Para explicar por qué la estrategia avara no es siempre la mejor, recurriremos al llamado **diagrama de estados** que mostrará el estado del registro de desplazamiento del circuito de la figura 2.7 cuando se ejecuta dos veces la misma tarea, retrasada la segunda respecto a la primera el número de ciclos señalado en cada arco del grafo. Sólo se representarán en el diagrama los estados correspondientes a las latencias que no producen colisión. Para generar el diagrama de estados se deben seguir los siguientes pasos:

1. Se parte del vector de colisiones como estado inicial del diagrama.

2. De cada estado deben salir tantos arcos como número de ceros haya en él. Cada uno de estos arcos nos llevará al estado del registro de desplazamiento cuando cada uno de los ceros salga por la izquierda, y se etiquetará con el número de la componente que ocupe ese cero en el vector. Hay que tener en cuenta que, para llegar al nuevo estado, es necesario efectuar la operación OR con el vector de colisiones, como se indicó antes. Si el estado al que se llega con esta operación no está aún en el diagrama, debe agregarse a la lista de estados con los que repetiremos la operación.

3. Cuando se haya terminado de efectuar la operación anterior con todos los estados existentes, se añadirá, a cada estado, un arco que le una con el estado inicial. A este arco se le asignará una etiqueta que sea igual al número de componentes del vector de colisiones más 1 y el superíndice +. El signo + sobre una etiqueta de arco indica que ese arco se refiere a la latencia indicada y a todas las mayores que ella, por esto, puede cambiarse la etiqueta de alguno de estos arcos a un valor inferior al número de componentes del vector de colisiones si, para las latencias mayores a ese valor, corresponde el mismo arco del diagrama.

Los arcos etiquetados con el superíndice + provienen del hecho de que el registro de desplazamiento se vacía después de cierto número de operaciones. A partir de ese momento, el circuito vuelve a su estado inicial.

En la figura 2.8 se muestra el diagrama de estados correspondiente al vector de colisiones de la figura 2.6(b). En el citado diagrama puede observarse que si aplicamos la estrategia avara, no se alcanzará el mejor rendimiento, porque el arco con latencia menor (2) nos llevará al estado `11111`, y éste nos obliga a salir por un arco de latencia 6. Si se arrancan sucesivas tareas siguiendo ese ciclo del grafo, el rendimiento final será peor que si hubiéramos salido por el arco de latencia 3.

Al diseñar un cauce, debe construirse el diagrama de estados para ver cuál es el recorrido cíclico en el grafo con mejor rendimiento. Éste se medirá por el número medio de tareas iniciadas por ciclo de reloj a lo largo de todo el ciclo (r) o, también, por la **latencia media** que es el número medio de ciclos de reloj que hay que esperar entre dos iniciaciones consecutivas de la tarea (esta medida será la inversa de la anterior). El recorrido cíclico del grafo que consigue la

Departamento de Informática Universidad de Valladolid

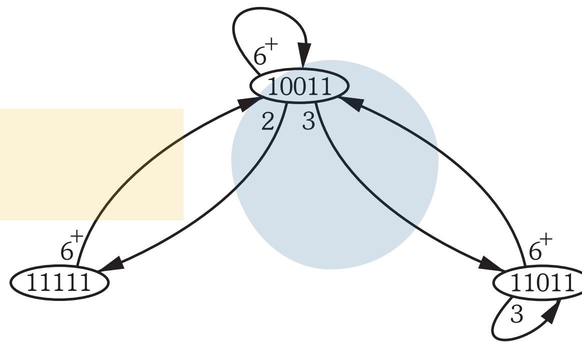


Fig. 2.8. Ejemplo de diagrama de estados.

mínima latencia media (***minimal average latency***, MAL) será el de mayor rendimiento. En el ejemplo anterior la estrategia avara tiene una latencia media de 4, mientras que su MAL es 3.

Como veremos a continuación, la MAL está acotada inferiormente por el número máximo de ciclos reservados (marcas) para una etapa (fila) en la tabla de reservas (Shar, 1972). Para demostrar esto, llamaremos N_i al número de ciclos reservados en la fila i de la tabla de reservas. El índice de utilización de la etapa i vendrá dada por rN_i. Este índice valdrá, como máximo, 1 (cuando todas las casillas de esa fila de la tabla de reservas estén marcadas, aunque no por una sola iniciación de la tarea), es decir,

$$rN_i \leq 1 \quad \Longrightarrow \quad r \,\text{máx}(N_i)_{i=1,..,n} \leq 1 \quad \Longrightarrow \quad r \leq \frac{1}{\text{máx}(N_i)_{i=1,..,n}}$$

donde n es el número de segmentos del cauce. Como se explicó antes, r es el inverso de la latencia media. Por tanto, el inverso del máximo valor de r será la MAL. Tomando los valores inversos de la última inecuación, tendremos que

$$MAL \geq \text{máx}(N_i)_{i=1,..,n}$$

que es lo que pretendíamos demostrar.

La conclusión final de todo esto es que, si llegamos a conseguir que la latencia entre dos iniciaciones consecutivas de la misma tarea, sea el número máximo de marcas en una fila de la tabla de reservas, se obtendrá el máximo rendimiento del cauce. Esto puede conseguirse, como se mostrará en el siguiente algoritmo debido a Patel y Davidson (1976), introduciendo retardos entre los segmentos del cauce. Para explicar el funcionamiento del algoritmo, lo aplicaremos sobre la tabla de reservas ya mostrada en la figura 2.6 y a la que corresponde el diagrama de estados de la figura 2.8. En este diagrama puede apreciarse que la mínima latencia media (MAL) se produce en el ciclo que sólo pasa por el estado 11011, y su valor es 3. Como se acaba de demostrar, la cota inferior del MAL, para este caso, es 2, que es el número máximo de marcas en una fila de la tabla de reservas.

El algoritmo propuesto puede seguirse en la figura 2.9. Para ilustrarlo mejor, se muestran las tablas de reservas para más tiempo que el necesario para una sola iniciación de la tarea. El algoritmo toma cada marca de la tabla de reservas, en orden temporal, y copia dicha marca con una periodicidad igual a la latencia deseada, en el caso de nuestro ejemplo, esta latencia es 2, como ya se mencionó antes. Señalaremos estas copias con una P (prohibida) indicando que esas

Departamento de Informática Universidad de Valladolid

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P		P		P		P
B											
C											
D											

(a)

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P		P		P		P
B		X		P		P		P		P	
C											
D											

(b)

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P		P		P		P
B		X		P		P		P		P	
C			X		P		P		P		P
D											

(c)

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P		P		P		P
B		X		P		P		P		P	
C			X		P		P		P		P
D				X		P		P		P	

(d)

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P		P		P		P
B		X		P		P		P		P	
C			X		P		P		P		P
D				X	X	P	P	P	P	P	P

(e)

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P	X	P	P	P	P	P
B		X		P		P		P		P	
C			X		P		P		P		P
D				X	X	P	P	P	P	P	P

(f)

	1	2	3	4	5	6	7	8	9	10	11
A	X		P		P	X	P	P	P	P	P
B		X		P		R	X	P	P	P	P
C			X		P		P		P		P
D				X	X	P	P	P	P	P	P

(g)

Fig. 2.9. Algoritmo para mejorar el rendimiento de un procesador segmentado introduciendo retardos.

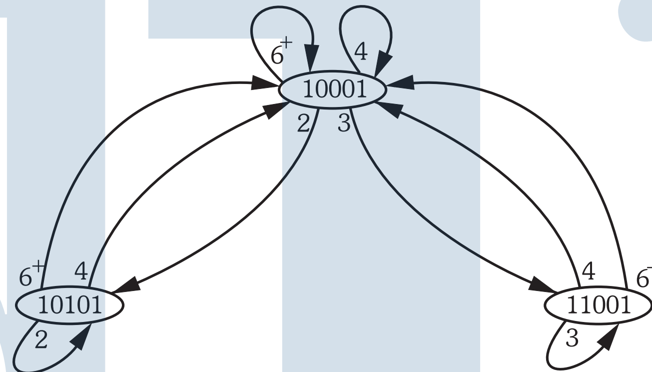
Departamento de Informática Universidad de Valladolid

	1	2	3	4	5	6	7
A	X					X	
B		X				R	X
C			X				
D				X	X		

(a)

10001

(b)



(c)

Fig. 2.10. (a) Tabla de reservas después de introducir retardos para mejorar el rendimiento, (b) vector de colisiones correspondiente dicha tabla y (c) diagrama de estados correspondiente a ese vector de colisiones.

casillas estarán ocupadas en las siguientes iniciaciones de la tarea. Puede ocurrir (figura 2.9(g)) que debamos poner una marca en una casilla ya ocupada (y, por tanto, señalada con una P); en este caso, introduciremos un retardo en esa etapa (señalado en la tabla con una R) para conseguir que ocupe un lugar libre. Cuando nos veamos obligados a ello, deberemos tener en cuenta que las etapas siguientes en el tiempo deberán ser retardadas en la misma medida. Como resultado de la aplicación de este proceso, llegaremos a una tabla de reservas con algunas etapas retardadas; para nuestro ejemplo concreto, esta tabla se muestra en la figura 2.10(a), su vector de colisiones se muestra en (b) y el diagrama de estados correspondiente puede verse en (c). Con esta tabla se conseguirá la latencia mínima posible para nuestro cauce, que es 2, pasando por los estados 100010 y 101010, quedando cíclicamente en éste último.

Muchos de los conceptos anteriores son también aplicables cuando se ejecutan varias tareas distintas en un cauce multifunción. En este caso, será necesario calcular los vectores de colisiones de cada función ejecutada después de cada una de las demás. Para ilustrar esta idea, analicemos lo que ocurriría en el cauce al que corresponde la tabla de la figura 2.6, si además de esa función (llamémosla X), pudiera ejecutar otra función Y, con la tabla de reservas y vector de colisiones que se muestran en la figura 2.11. Para saber en qué momento podremos arrancar la función Y cuando ya se está ejecutando la función X, superpondremos las tablas de reservas de ambas operaciones y desplazaremos la correspondiente a la función Y hacia la derecha hasta que no haya superposiciones (colisiones); de esta forma, podremos construir el vector de

Departamento de Informática Universidad de Valladolid

	1	2	3	4	5	6
A	X				X	
B		X			X	
C			X			
D				X		X

(a)

01110

(b)

Fig. 2.11. (a) Tabla de reservas para una segunda función en un cauce bifunción y (b) vector de colisiones correspondiente a dicha tabla.

colisiones de Y después de X (**vector de colisiones cruzadas**), que será:

$$\mathbf{v}_{yx} = (10011)$$

Análogamente podríamos proceder al cálculo del vector de colisiones cruzadas de X después de Y, con lo que se obtendría:

$$\mathbf{v}_{xy} = (11110)$$

Con estos vectores, y los vectores de colisiones de las propias funciones ($\mathbf{v}_{xx}$ y $\mathbf{v}_{yy}$), construiremos la llamada **matriz de colisiones** para una función (M_X), que es una matriz cuyas filas son los vectores de colisiones correspondientes a esa función cuando después se ejecuta cada una de las demás. Si los vectores tuvieran diferentes números de componentes, la matriz toma tantas columnas como corresponda a la longitud del vector más largo, completando los demás con ceros por la derecha. Para las operaciones anteriores, las matrices de colisiones serán:

$$M_X = \begin{pmatrix} \mathbf{v}_{xx} \\ \mathbf{v}_{yx} \end{pmatrix} = \begin{pmatrix} 10011 \\ 10011 \end{pmatrix}$$

$$M_Y = \begin{pmatrix} \mathbf{v}_{xy} \\ \mathbf{v}_{yy} \end{pmatrix} = \begin{pmatrix} 11110 \\ 01110 \end{pmatrix}$$

Con estas matrices de colisiones, podremos determinar si, en un determinado ciclo de reloj, se podría admitir el comienzo de una nueva tarea en el procesador segmentado. Para hacerlo, ampliaremos el circuito mostrado en la figura 2.7 para tener en cuenta todos los casos posibles en el orden de ejecución de las funciones. Este circuito ampliado se muestra en la figura 2.12, y su funcionamiento es completamente análogo al anterior. En general, un procesador segmentado, en que se puedan ejecutar p funciones diferentes, se podrá controlar con p registros de desplazamiento.

También puede construirse para este caso el diagrama de estados. Este diagrama, que es análogo al correspondiente a los cauces monofuncionales, difiere sin embargo de éstos en los siguientes aspectos:

Departamento de Informática Universidad de Valladolid

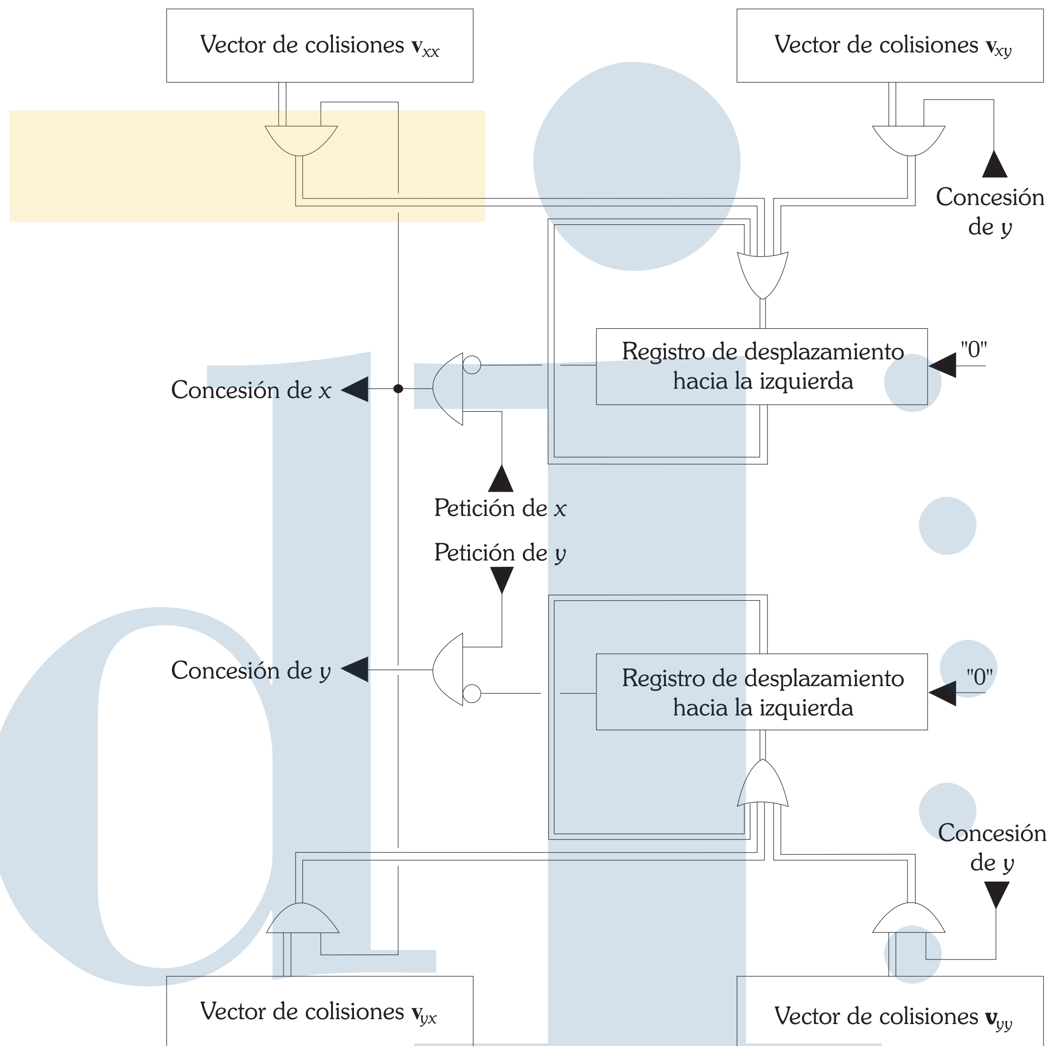


Fig. 2.12. Sistema de prevención de colisiones para un cauce bifunción.

- Los nodos, en vez de estar identificados por un vector, están identificados por una matriz.
- El estado inicial no es único, sino que hay tantos estados iniciales como funciones pueda efectuar el procesador segmentado.
- En los arcos que conectan los nodos debe indicarse, además de la latencia, la operación que se ejecuta.

Como muestra de diagrama de estados, en la figura 2.13 puede verse el correspondiente al ejemplo anterior.

Departamento de Informática Universidad de Valladolid

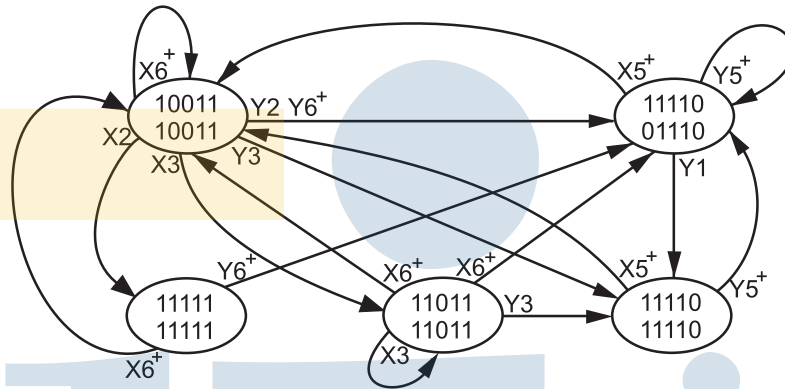


Fig. 2.13. Diagrama de estados para un procesador segmentado bifunción.

2.5.2. Conflictos por dependencias de datos

Como ya se explicó anteriormente, este tipo de conflictos surgen cuando varias tareas acceden a los mismos datos. Normalmente estos conflictos se producen en procesadores segmentados de instrucciones, sobre todo si están profundamente segmentados, pero también podrían producirse en cauces aritméticos.

Supongamos dos instrucciones A y B que se ejecutan en ese mismo orden en un procesador segmentado de instrucciones. Las dependencias de datos que se pueden presentar pueden ser de tres clases:

RAW (***read after write*, lectura después de escritura**): la instrucción B trata de leer un operando antes de que la instrucción A lo haya escrito; de esta forma, B puede tomar un valor incorrecto de ese operando. Éste es el tipo de dependencia más común. Trataremos de exponer ahora una condición más formalizada para la existencia de estos riesgos (Keller, 1975): Si denotamos por O_A al conjunto de salida o rango de la instrucción A, y por I_B al conjunto de entrada o dominio de la instrucción B (recordar el apartado 1.3.4), habrá posibilidad de riesgo RAW si:

$$O_A \cap I_B \neq \emptyset$$

Esto, que corresponde con la dependencia de flujo definida en la apartado 1.3.4, es una condición necesaria pero no suficiente. La existencia o no de riesgo depende de la separación temporal de las instrucciones A y B, así como de la profundidad de la segmentación.

WAR (***write after read*, escritura después de lectura**): la instrucción B trata de escribir en un registro antes de que sea leído por la A; por ello, A tomaría un valor incorrecto, ya que debería tomar el valor antes de modificarse. Este riesgo es poco frecuente, debido a que la escritura de los resultados se efectúa en los últimos segmentos y es difícil que esa escritura se adelante al segmento de lectura de una instrucción anterior. El riesgo WAR se podría presentar, sobre todo, en procesadores con autoindexación, en los que la fase de lectura de operando, que es una de las primeras, puede modificar un registro antes de que una instrucción anterior lo haya leído. Aun en este tipo de procesadores este riesgo es difícil que se produzca. No obstante, como veremos más adelante, esta clase de riesgos

Departamento de Informática Universidad de Valladolid

se pueden producir si se alterara el orden de la ejecución de las instrucciones. Existirá peligro de producirse este riesgo si

$$I_A \cap O_B \neq \emptyset$$

es decir, si existe antidependencia entre las instrucciones A y B, aunque, igual que en el caso anterior, esta condición es necesaria pero no suficiente.

WAW (***write after write*, escritura después de escritura**): la instrucción B intenta escribir un operando antes de que sea escrito por la A. Dado que las escrituras se están realizando en orden incorrecto, el resultado final depositado es el correspondiente a la instrucción A cuando debería ser el depositado por la B. Este conflicto sólo se produce en procesadores segmentados que escriben en más de una etapa y esto no es muy frecuente. Este riesgo, igual que el anterior, se podría producir en máquinas que permiten direccionamientos autoindexados. Al igual que el riesgo WAR, los riesgos WAW también podrían aparecer si se cambiara el orden de ejecución de las instrucciones. Podrá existir un riesgo WAW entre las instrucciones A y B si existe dependencia de salida entre las citadas instrucciones, es decir, si se verifica que:

$$O_A \cap O_B \neq \emptyset$$

Aunque, como en los casos anteriores, esta condición no es suficiente para la existencia del riesgo.

Debe observarse que RAR (lectura después de lectura) no constituye ningún riesgo puesto que ninguna de las dos instrucciones cambia el valor del dato.

Existen técnicas para detectar los conflictos por dependencias de datos. Estas técnicas precisan algo de hardware adicional: unos buffers, que guarden los números de los registros implicados en las últimas instrucciones, junto con unos comparadores. Los buffers tendrán una estructura de registro de desplazamiento, de forma que la posición del número de registro dentro del desplazador nos dirá en qué segmento se encuentra y la clase de operación que se efectúa sobre él (lectura o escritura). Los comparadores analizarán si coinciden los operandos de la instrucción que va a ejecutarse, con los de alguna de las anteriores con las que pueda entrar en conflicto. Si existe conflicto, como primera solución, se detendrá la ejecución de la instrucción hasta que las anteriores hayan avanzado y la dependencia desaparezca (esto es lo que se llama insertar una **burbuja**). Frecuentemente, en este caso, se dice que se impide la **emisión de la instrucción**, ya que se llama así al paso de la instrucción de la etapa de decodificación a la siguiente, donde comienza la ejecución efectiva de la instrucción (localización de operandos). Una solución para reducir la duración de estas detenciones es hacer uso del resultado, en cuanto se disponga de él, aunque no se haya escrito en el registro afectado; este método se llama **anticipación**

Existen también técnicas para evitar las posibles detenciones que podría producir un conflicto. Estos métodos pueden clasificarse en dos grupos:

Planificación estática: las técnicas de este tipo consisten en habilitar los medios software para que el compilador detecte los riesgos y cambie el orden de las instrucciones que los producen en la medida de lo posible. Si el compilador no encuentra ninguna instrucción que pueda evitar la detención, insertará una instrucción NOP (no operación) para dejar un lugar libre en la cadena de procesamiento sin tener que efectuar una detención.

oi. Departamento de Informática Universidad de Valladolid

Para ver los efectos de la planificación estática, tomaremos como ejemplo el siguiente fragmento de código (se supone que el destino es el último operando):

```
ADD  R1, R0
MOV  R5, R1
LOAD A,  R2
LOAD B,  R3
MUL  R3, R2
```

En esta porción de código, `A` y `B` representan dos direcciones de memoria. En el ejemplo, la instrucción `MUL` tiene una dependencia de datos respecto de las instrucciones que acceden a memoria y hasta que éstas no tengan los datos, `MUL` no podrá leer los operandos. En este fragmento de código se puede observar que las instrucciones `ADD` y `MOV` son independientes de las siguientes por lo que pueden ejecutarse en cualquier orden respecto a ellas. Según esto, un compilador que lleve a cabo planificación estática puede adoptar la solución de reordenar estas instrucciones de la forma siguiente:

```
LOAD A,  R2
LOAD B,  R3
ADD  R1, R0
MOV  R5, R1
MUL  R3, R2
```

Esta reordenación de las instrucciones evitará la dependencia de datos, ya que mientras se ejecutan `ADD` y `MOV`, dará tiempo a que las instrucciones `LOAD` tengan disponibles los datos.

La mayoría de los computadores segmentados actuales utilizan planificación estática, bien efectuada por el propio compilador, o bien, por parte de algún post-procesador que optimice el código generado por el compilador.

Planificación dinámica: con esta clase de métodos es el propio procesador, durante la ejecución de las instrucciones, el que cambia el orden de emisión de las mismas para mantener los segmentos del cauce tan ocupados como sea posible. Esto puede hacerse incluyendo la detección y evitación de conflictos en el código del microprograma, en los procesadores microprogramados, o bien añadiendo el hardware oportuno en los procesadores cableados.

Existen dos métodos clásicos para implementar la planificación dinámica:

- El **marcaje (*scoreboarding*)**: un marcador (*scoreboard*) es un sistema de tablas (mantenidas a nivel hardware) que tiene información sobre la ocupación de los recursos de la máquina. Mediante el marcador podemos saber cuándo una instrucción tiene o no dependencias pendientes. Si tuviera alguna dependencia, aplazamos su emisión intentándola con otra instrucción posterior en que esas dependencias no se produzcan. El emitir las instrucciones fuera de orden puede tener el inconveniente de que se pueden producir nuevos riesgos que también deberán ser considerados por el marcador. Cada instrucción pasa por el marcador (en una de las primeras fases

Departamento de Informática Universidad de Valladolid

de la segmentación) donde se mantiene una tabla con las dependencias de datos. En esta tabla está la información suficiente para saber cuándo la instrucción puede emitirse o no. Si la instrucción no se pudiera emitir en ese momento, intenta emitir otra y controla los cambios producidos para saber cuándo la dependencia de datos está resuelta. El marcador contiene las siguientes informaciones:

1. Estado de las instrucciones, es decir, el segmento en el que se encuentra cada instrucción.
2. Estado de las unidades funcionales: indica la ocupación de cada unidad funcional además de los operandos con los que está trabajando.
3. Estado de los registros de resultado: nos informará sobre las escrituras en los registros para saber cuándo están actualizados.

Como puede apreciarse, el hardware adicional necesario para implementar un marcador es tan grande que puede no compensar los beneficios obtenidos: un marcador puede necesitar, aproximadamente, tanto hardware como el resto del procesador, para conseguir, tan sólo, la misma mejora de rendimiento que la planificación estática.

La técnica del marcaje se empleó por primera vez en el computador CDC6600 en los años 60 y no se emplea mucho en la actualidad debido a la gran complejidad del hardware añadido que no compensa los beneficios obtenidos.

- El **algoritmo de Tomasulo** (1967): este algoritmo es similar a los sistemas de marcadores, con la diferencia de que la información sobre la ocupación de recursos está distribuida en las llamadas **estaciones de reserva**. Cada unidad funcional tiene una estación de reserva que controla cuándo puede comenzar una nueva operación en esa unidad funcional, es decir cuándo sus operandos están listos.

  Cada estación de reserva tiene las siguientes informaciones:

  1. La operación a efectuar.
  2. Los números de las estaciones de reserva que corresponden a las unidades funcionales que producirán los operandos de la instrucción. Debe existir un valor (normalmente 0) para indicar que alguno de los operandos ya está disponible; para este caso se activa el campo descrito a continuación.
  3. Los valores de los operandos fuente si es que ya están disponibles.

  El algoritmo de Tomasulo utiliza **anticipación**, es decir, si un operando está disponible, no se espera a que sea almacenado en el registro de destino, sino que se lleva a las estaciones de reserva que lo estén esperando; de esa forma, éstas lo podrán utilizar para sus respectivas operaciones. Otra característica importante es que todas las estaciones de reserva tienen acceso a un bus, denominado **bus común de resultados**, en que las unidades funcionales dejan los resultados de las operaciones. La ventaja que tiene el disponer de este bus es que, si hubiera varias estaciones de reserva esperando por el mismo dato, cuando ese dato aparezca en el bus común, podrá ser capturado por todas y continuar la ejecución.

  Para este algoritmo también se precisa mucho hardware, con la diferencia respecto a los marcadores, de que ese hardware está distribuido entre todas las estaciones de reserva.

  El algoritmo de Tomasulo se utilizó en el IBM-360/91 también en los años 60.

Departamento de Informática Universidad de Valladolid

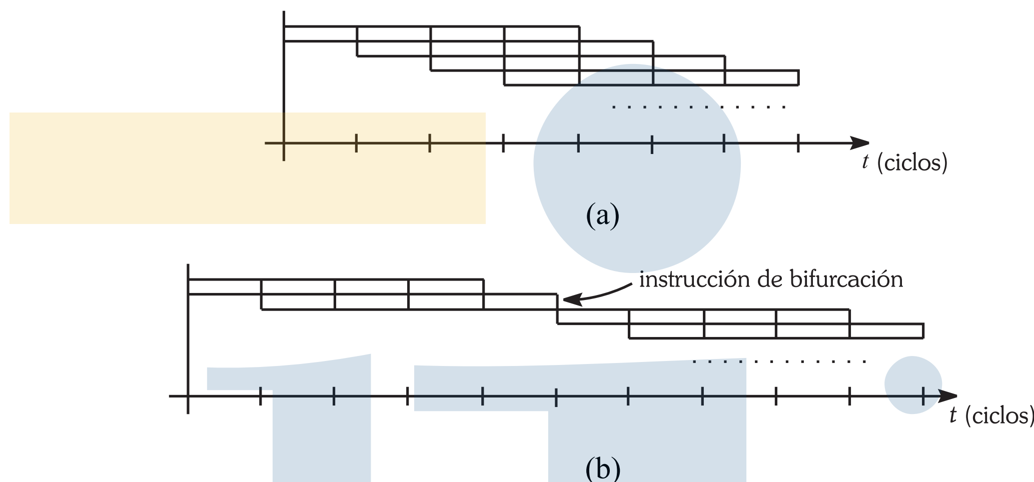


Fig. 2.14. (a) Ejecución de instrucciones sin bifurcaciones y (b) detención producida por un conflicto de control.

Las técnicas de planificación dinámica pueden ser algo más eficaces que las estáticas, pero también son bastante más complicadas. Una de las misiones del diseñador de un procesador segmentado será decidir el tipo de planificación más adecuado, sopesando las ventajas e inconvenientes de cada uno.

2.5.3. Conflictos de control

Este tipo de conflictos se producen en los procesadores de instrucciones segmentados. Como se mencionó anteriormente, los conflictos de control se deben a las instrucciones de control del flujo, que no permiten que se lea la instrucción siguiente hasta que no se conozca su dirección (que precisamente es calculada por la propia instrucción de control de flujo). Parece claro que una instrucción de control de flujo provocará una detención del procesador, ya que el cálculo de la dirección de destino, al menos en principio, se efectúa en la fase de escritura de resultado (de la instrucción de control) y esta fase suele ser la última. Esto es especialmente válido en las instrucciones de bifurcación condicional. Una imagen gráfica de un conflicto de control puede verse en la figura 2.14. Conflictos de esta misma índole se producen también debido a los desvíos e interrupciones. En estos casos, los conflictos son más graves porque no están previstos en el programa y, por tanto, son mucho más difíciles de controlar.

Calcularemos la pérdida de rendimiento debida a los saltos si no se toma ninguna medida correctora. Supongamos que, en cuanto se detecte una instrucción de control de flujo, se detiene inmediatamente el procesador hasta que la dirección de la instrucción siguiente se haya calculado. Evidentemente, cuanto mayor sea el porcentaje de instrucciones de control de flujo, mayor será la influencia de los conflictos de control en el rendimiento del cauce.

Sea P_s la probabilidad de aparición de una instrucción de control, y b el numero de ciclos perdidos por la detención (**penalización del salto**). En estas condiciones, el tiempo de ejecución de m tareas será:

$$t = (n + (m - 1) + bmP_s)T_s$$

Departamento de Informática Universidad de Valladolid

Podemos mejorar un poco este tiempo si dejamos que se continúen leyendo instrucciones. De esta forma, si la instrucción de control no es efectiva (es decir si la condición no se cumple), habremos adelantado algo. Para los cálculos siguientes vamos a suponer que, cuando la condición de bifurcación se haya evaluado, todavía no se ha emitido la instrucción siguiente, es decir, que dicha instrucción no ha efectuado ninguna acción difícilmente reversible (modificación de registros, *flags*, etc.). Estas suposiciones pueden servir para hacernos una idea de la incidencia de los conflictos de control en el rendimiento de un procesador segmentado:

Sea P_t la probabilidad de que una instrucción de control sea efectiva, entonces, y con los anteriores supuestos, el tiempo de ejecución de m tareas será:

$$t(n) = (n + (m-1) + bmP_tP_s)T_s$$

La productividad del procesador vendrá dada por:

$$H = \frac{m}{(n + (m-1) + bmP_tP_s)T_s} = \frac{mf}{n + (m-1) + bmP_tP_s}$$

donde f es la frecuencia del reloj. Esta productividad, en el estado estacionario, es decir, para valores grandes de m, tenderá a:

$$H_\infty = \lim_{m\to\infty} H = \lim_{m\to\infty} \frac{mf}{n + (m-1) + bmP_tP_s} = \frac{f}{1 + bP_tP_s}$$

Si suponemos que las bifurcaciones se resuelven en el último segmento, tendremos que el número de ciclos perdidos, b, será $n-1$, con lo que la productividad estacionaria se podrá escribir como:

$$H_\infty = \frac{f}{1 + (n-1)P_tP_s}$$

Otra forma de medir las pérdidas por riesgos de control sería calcular el CPI teniendo en cuenta estos riesgos (pero prescindiendo de los demás), es decir:

$$\begin{aligned} CPI &= (1-P_s) + P_s[P_t(1+b) + (1-P_t)] = 1 + bP_sP_t \\ &= 1 + (n-1)P_tP_s \end{aligned} \qquad [2.9]$$

Como se ve las pérdidas debidas a los conflictos de control son bastante importantes, ya que, en condiciones ideales, el CPI estacionario debe ser 1. Esto significa que la eficiencia se calculará dividiendo este CPI ideal por el obtenido en 2.9, por tanto:

$$E_\infty = \frac{1}{1 + bP_sP_t} = \frac{1}{1 + (n-1)P_sP_t}$$

Una de las soluciones para evitar estas pérdidas de rendimiento es hacer que el compilador encuentre algo útil para hacer mientras calcula la dirección del destino del salto. Esto se conseguirá reordenando las instrucciones de la misma forma que se hacía en la planificación estática de los riesgos por dependencias de datos. Si el compilador no pudiera encontrar ninguna instrucción para ocupar los huecos libres dejados por la instrucción de salto, rellenaría con instrucciones NOP. Para llevar a cabo este método, denominado **bifurcación retardada**, tienen que ponerse de acuerdo en la forma de operar el hardware y el compilador. En la figura 2.15 se muestra la

Departamento de Informática Universidad de Valladolid

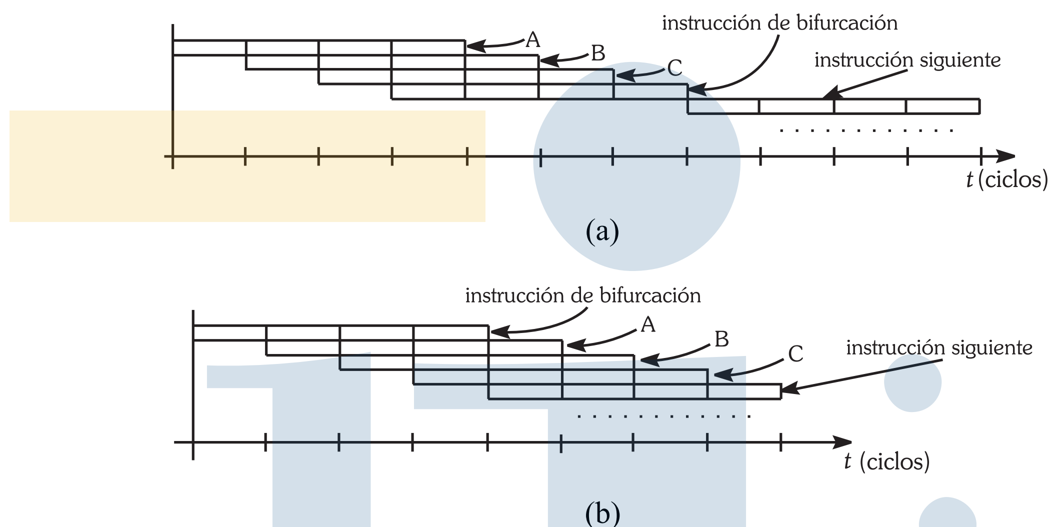


Fig. 2.15. Efecto de la bifurcación retardada: (a) Detención producida por una bifurcación sin tomar medidas correctoras y (b) efecto del cambio de orden de las instrucciones producido por un compilador en una máquina con bifurcación retardada.

forma de operar de la bifurcación retardada: el hardware de la máquina debe garantizar que la instrucción de bifurcación cambie el contador de programa siempre en la misma etapa, tanto si el salto es efectivo como si no es; en la citada figura se ha supuesto que esa etapa es la última, aunque no necesariamente eso es siempre así.

Existen otros métodos más sofisticados para mejorar las bifurcaciones. Estos métodos se basan en la predicción del destino de los saltos para las instrucciones de bifurcación condicional. Esta predicción puede hacerse a nivel estático (compilador) o a nivel dinámico (en ejecución). El problema de todos los métodos de predicción es que, si ésta es incorrecta, hay que descartar las instrucciones que hayan entrado en el cauce equivocadamente, y eso puede acarrear una pérdida de rendimiento muy grande:

- La **predicción estática** se basa en los diferentes comportamientos de los distintos tipos de instrucciones de bifurcación condicional. Concretamente las bifurcaciones debidas a iteraciones se cumplen casi siempre y el comportamiento de las bifurcaciones condicionales puede preverse por el contexto (aunque no siempre). Basándose en estas consideraciones, el diseñador de la máquina puede suministrar dos códigos de operación para las bifurcaciones condicionales: uno para el salto probable y otro para el salto no probable. Se puede mejorar este enfoque con la predicción denominada **semiestática**, que consiste en pasar el programa, compilado con predicción estática, por un simulador de la máquina para que se estudie si el comportamiento de los saltos es el previsto o no; si las predicciones fueran incorrectas, se podrían corregir los fallos de la predicción estática.

- La **predicción dinámica** (Sussenguth, 1971) consiste en llevar una tabla con la historia de cada salto. Esta tabla tendría la forma mostrada en la figura 2.16. La información estadística de la tabla se iría actualizando continuamente con cada ejecución de una instrucción de bifurcación condicional, esto significa que al comienzo de la ejecución del programa la

Departamento de Informática Universidad de Valladolid

Dirección de la instrucción de salto	Información estadística	Destino del salto
...	...	...

Fig. 2.16. Tabla de saltos para predicción dinámica.

estimación de la probabilidad de efectividad del salto no será muy correcta; sin embargo, esa probabilidad se irá corrigiendo automáticamente con el tiempo.

El problema de la predicción dinámica radica en que su gran complejidad, en cuanto al hardware, no está justificada por su fiabilidad en la predicción.

2.6. Procesadores segmentados y arquitectura RISC

Como es sabido, los procesadores **RISC** (*Reduced Instruction Set Computer*, **computadores con conjunto reducido de instrucciones**) se caracterizan por tener un conjunto de instrucciones muy simple y, también, pocos modos de direccionamiento asimismo sencillos. Otra característica típica de los procesadores RISC es ejecutar, en media, una instrucción por ciclo.

Todo esto es así, precisamente, porque este tipo de procesadores son segmentados. Pero cabe preguntarse ¿Porqué es conveniente que un procesador segmentado posea un sencillo y reducido conjunto de instrucciones y un limitado espectro de simples modos de direccionamiento? La respuesta a esta pregunta se encuentra, en gran medida, en el **equilibrio de la segmentación**. Sólo con un limitado juego de instrucciones sencillas y con unos simples modos de direccionamiento, es posible aproximarse a un equilibrio entre los segmentos. Si incluyéramos, por ejemplo, algún modo de direccionamiento complicado en un procesador segmentado, pondríamos en peligro el equilibrio de la segmentación, ya que el tratamiento de este modo sería más costoso, en tiempo, que el de los demás. Esto implicaría que la etapa de obtención de operandos sería más larga debido a este modo, con lo que se perdería el equilibrio (también pudiera hacerse añadiendo etapas a la segmentación, solución que también comprometería el rendimiento). Por otra parte, y como ya se analizó en la sección 2.5.2, algunos modos de direccionamiento más elaborados pueden aumentar el peligro de riesgo por dependencias de datos, concretamente los autoindexados. Algo parecido ocurre con el conjunto de instrucciones: si introducimos instrucciones más complejas, éstas necesitarán más etapas para poder completarse, por lo que el equilibrio también se vería comprometido. Uno de los principales objetivos de los procesadores RISC es obtener un procesador con un ciclo de reloj muy corto y una productividad de una instrucción por ciclo. Esto no sería posible si alguna de las etapas es más complicada que las demás, y por extensión, si algún modo es más elaborado o si alguna instrucción es más sofisticada.

Hay más características propias de los procesadores RISC relacionadas con la segmentación:

- La limitación de los accesos a memoria, permitiéndose sólo en las instrucciones LOAD y STORE. Esto se debe a que, si se permitiera el acceso libre a memoria, serían frecuentes

Departamento de Informática Universidad de Valladolid

los riesgos estructurales debidos a la misma. Por otra parte, si se permitiera el acceso a memoria en cualquier tipo de instrucción, peligraría el equilibrio entre los segmentos debido a la complicación en la etapa de lectura de operando.

- El empleo de arquitectura Harvard, esto es, memorias caché separadas para código y datos. La arquitectura Harvard evita conflictos estructurales entre la etapa de lectura de la instrucción y las etapas que acceden a memoria en las instrucciones LOAD y STORE.
- La utilización de formatos de instrucción fijos, esto favorece que la fase de lectura de la instrucción sea homogénea. Si los formatos fueran de longitud variable, también lo sería la duración del ciclo de lectura con lo que se comprometería el equilibrio de la segmentación.

2.7. Procesadores superescalares y supersegmentados

Diremos que un procesador es superescalar con grado N, si dispone de N cauces de segmentación de forma que puedan entrar N instrucciones a la vez en el procesador (siempre que no haya conflictos que lo impidan). Esto exige multiplicar todas las unidades funcionales del procesador y organizar la compilación de forma que sólo entren simultáneamente en el procesador instrucciones independientes entre las que no pueda haber conflictos. Este tipo de procesadores permitiría, en condiciones óptimas, ejecutar N instrucciones por ciclo: en este caso, se dice que su **nivel de paralelismo de instrucción (*instruction level parallelism*, ILP)** es N

Vamos a analizar el rendimiento de los procesadores superescalares. Para calcular este rendimiento, tomaremos como máquina base de comparaciones a una máquina segmentada convencional con n etapas (máquina escalar). El tiempo para ejecutar m tareas en esta máquina de referencia será, según la ecuación 2.1:

$$t(1,1) = n + m - 1$$

En una máquina superescalar de grado N, el tiempo necesario para ejecutar esas m tareas será:

$$t(N,1) = n + \frac{m-N}{N} = n + m/N - 1$$

La ganancia de velocidad conseguida sobre la máquina de referencia será:

$$S(N,1) = \frac{t(1,1)}{t(N,1)} = \frac{n+m-1}{n+m/N-1} = \frac{N(n+m-1)}{m+N(n-1)}$$

Para valores grandes de m, la ganancia estacionaria será:

$$S_\infty(N,1) = \lim_{m\to\infty} S(N,1) = \lim_{m\to\infty} \frac{N(n+m-1)}{m+N(n-1)} = N$$

Como parecía lógico esperar.

Uno de los inconvenientes de los procesadores superescalares es que los ciclos perdidos por detenciones también se multiplican por el grado. Un diagrama del funcionamiento de un procesador superescalar se muestra en la figura 2.17.

Departamento de Informática Universidad de Valladolid

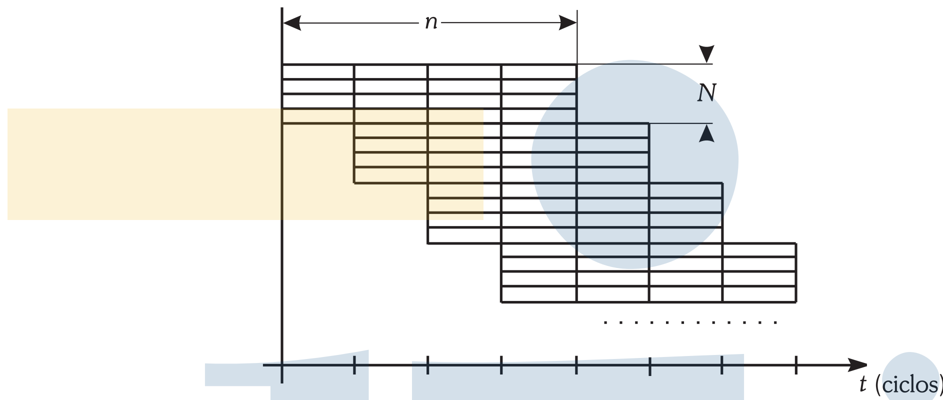


Fig. 2.17. Funcionamiento de un procesador superescalar de grado 4.

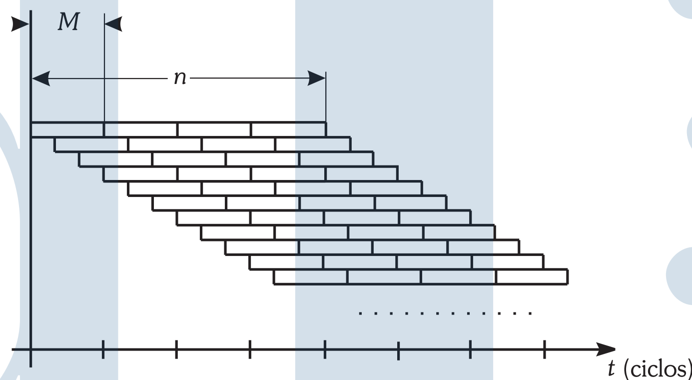


Fig. 2.18. Funcionamiento de un procesador supersegmentado de grado 3.

Un procesador supersegmentado de grado M, es un procesador segmentado con un ciclo de reloj base M veces más pequeño que el de un procesador segmentado convencional. Eso significa que los ciclos máquina son mucho más elementales. El tiempo necesario para procesar m tareas, medido en ciclos de reloj de la máquina de referencia será

$$t(1, M) = n + \frac{m-1}{M}$$

Por tanto, la ganancia en velocidad respecto a la máquina de referencia vendrá dada por

$$S(1, M) = \frac{t(1,1)}{t(1,M)} = \frac{n+m-1}{n+\frac{m-1}{M}} = \frac{M(n+m-1)}{nM+m-1}$$

Para el estado estacionario, la ganancia de velocidad tenderá a:

$$S_\infty(1, M) = \lim_{m\to\infty} S(1, M) = \lim_{m\to\infty} \frac{M(n+m-1)}{nM+m-1} = M$$

El diagrama temporal de un procesador supersegmentado se muestra en la figura 2.18.

Departamento de Informática Universidad de Valladolid

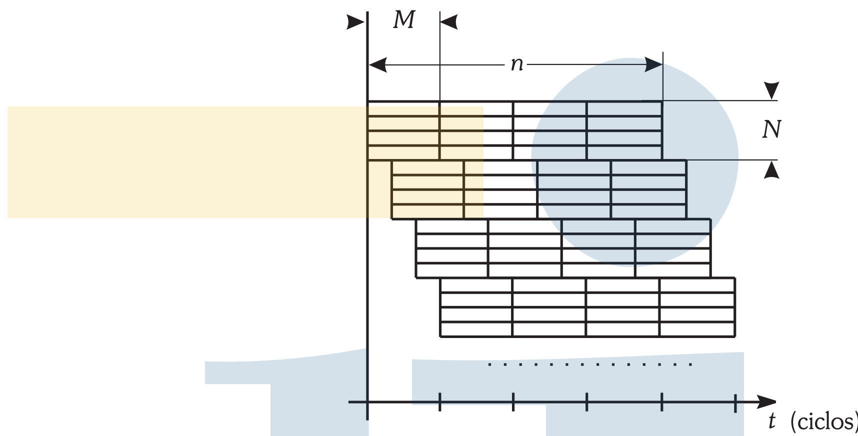


Fig. 2.19. Funcionamiento de un procesador superescalar y supersegmentado de grado (4,3).

Tabla 2.1. Resumen de las características de los procesadores superescalares y supersegmentados tomando como referencia una máquina segmentada convencional.

Tipo de máquina	Escalar con n segmentos	Superescalar de grado N	Super-segmentada de grado M	Superescalar supersegmentada de grado (N,M)
Ciclo máquina básico	1	1	$1/M$	$1/M$
Instrucciones por emisión	1	N	1	N
Ciclos entre emisiones consecutivas	1	1	$1/M$	$1/M$
ILP sin riesgos	1	N	M	MN
Ganancia de velocidad	1	N	M	MN

Los procesadores superescalares y supersegmentados pueden combinarse (figura 2.19). En este caso, el tiempo necesario para ejecutar m tareas, medido en ciclos de la máquina de referencia, será:

$$t(N,M) = n + \frac{m-N}{MN}$$

Por tanto, la ganancia de velocidad vendrá dada por:

$$S(N,M) = \frac{t(1,1)}{t(N,M)} = \frac{n+m-1}{n+\frac{m-N}{MN}} = \frac{MN(n+m-1)}{MNn+m-N}$$

que, en el estado estacionario, tenderá a:

$$S_\infty(N,M) = \lim_{m\to\infty} S(N,M) = \lim_{m\to\infty} \frac{MN(n+m-1)}{MNn+m-N} = MN$$

Departamento de Informática Universidad de Valladolid

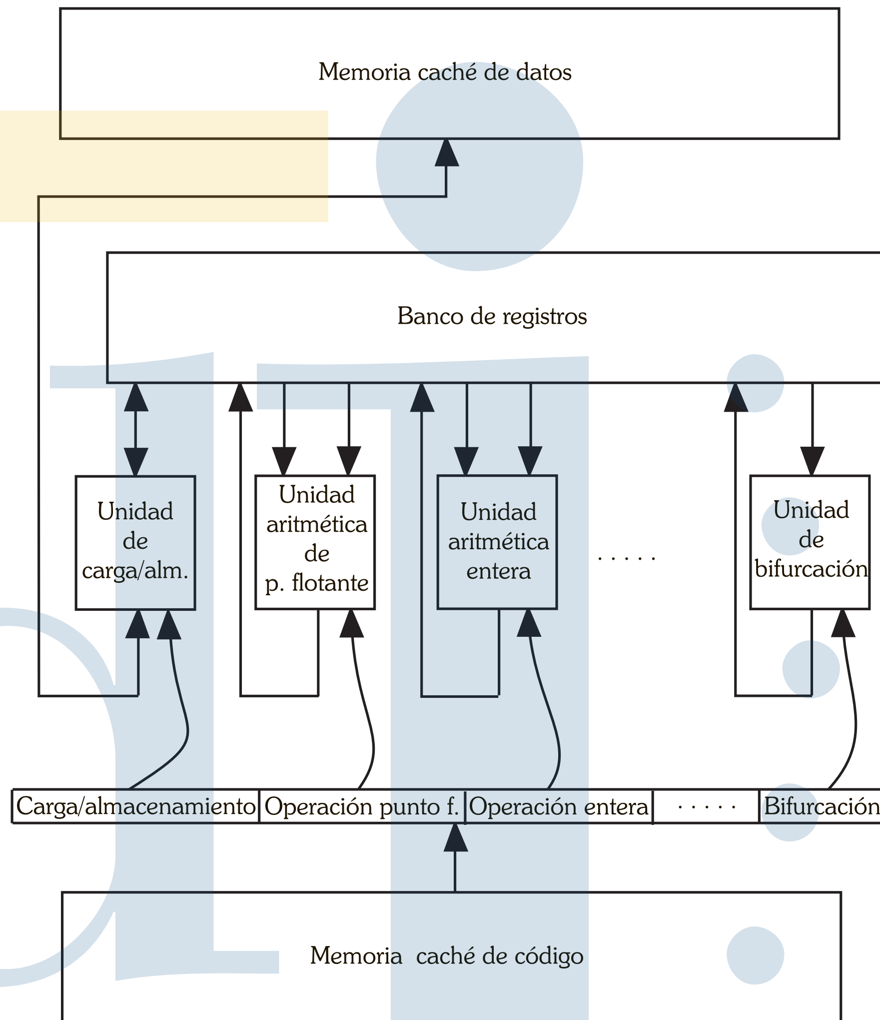


Fig. 2.20. Arquitectura VLIW.

En la tabla 2.1 se muestran, de forma comparativa, las características de los procesadores superescalares, supersegmentados y de ambos combinados.

2.8. Segmentación en procesadores VLIW

Las arquitecturas VLIW (***Very Large Instruction Word***, **palabra de instrucción muy larga**) surgen como una evolución de los conceptos de microprogramación horizontal y procesamiento superescalar, desarrollando el paralelismo en los niveles de instrucción e intrainstrucción. La idea central de estas arquitecturas va en la misma línea que los procesadores RISC: bajar más el nivel del lenguaje máquina y hacer que esa distancia adicional en la barrera semántica (diferencia entre los lenguajes de alto nivel y el lenguaje máquina) la supere el compilador. Estos conceptos se muestran en la figura 2.20, en que puede verse que la instrucción tiene muchos campos que activan, por separado, cada una de las unidades funcionales. Todas esas

Departamento de Informática Universidad de Valladolid

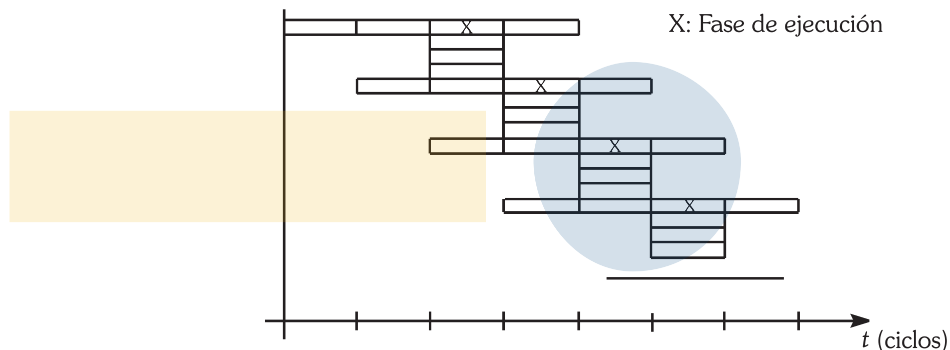


Fig. 2.21. Esquema temporal de funcionamiento de un procesador con arquitectura VLIW.

unidades funcionales trabajan sobre el mismo banco de registros. La ventaja de este tipo de instrucciones es que puede explotarse el paralelismo a nivel de instrucción sin necesidad de leer las diferentes instrucciones, aunque se necesitará una memoria con mayor ancho de banda, para leer toda la instrucción de una sola vez. Los anchos típicos de palabra de instrucción para este tipo de máquinas están entre 256 y 1024 bits. En estas máquinas, el trabajo del compilador es mucho más arduo que en un procesador convencional, porque es él quien tiene que efectuar una detección del paralelismo para averiguar, partiendo del lenguaje de alto nivel, qué acciones pueden llevarse a cabo en la misma instrucción. Cuando estas instrucciones de gran tamaño se ejecutan en un procesador segmentado, el diagrama de tiempos será el mostrado en la figura 2.21. Como puede fácilmente observarse, el diagrama es parecido al de un procesador superescalar, salvo en que sólo se hacen simultáneas las fases de ejecución, correspondientes a las diferentes unidades funcionales que trabajan en paralelo, ya que no es necesario simultanear el resto de las fases (esto es una cuestión de criterio, porque las demás fases también tendrán mayor complicación debido a la mayor longitud y complejidad de la instrucción). En principio, podemos decir que las máquinas VLIW se comportan como las máquinas superescalares, pero con tres importantes diferencias:

- La decodificación de las instrucciones VLIW es más fácil que la de las convencionales. Esto es debido a que, en realidad, las instrucciones VLIW son de un nivel más bajo.
- La densidad del código de las máquinas VLIW es más baja que en las máquinas convencionales, debido a que no todos los campos se utilizarán en todas las instrucciones. La densidad del código dependerá, por un lado del grado de paralelismo del programa y, por otro, de la calidad del compilador en orden a extraer ese paralelismo y adaptarlo a la estructura de la máquina.
- El código ejecutable de una máquina VLIW es muy poco compatible. Esto se debe a que el paralelismo se explota a un nivel muy bajo, nivel en el que el código no es muy transportable.

Una de las características más notables de los computadores VLIW es que el paralelismo se extrae completamente en el momento de la compilación. Eso significa que no existirán planificaciones dinámicas, ni resolución de conflictos en ejecución. Esto, por supuesto, complica mucho

Departamento de Informática Universidad de Valladolid

el compilador pero elimina cualquier hardware o software de bajo nivel para evitar conflictos o detectar el paralelismo en ejecución.

Bibliografía y referencias

BASTIDA, J. 1995. *Introducción a la Arquitectura de Computadores*. Universidad de Valladolid.

DAVIDSON, E.S. 1971 (Jan.). The Design and Control of Pipelined Function Generators. *In: Proceedings of the 1971 International Conference on Systems, Networks and Computers*.

HÄNDLER, W. 1977. The Impact of Classification Schemes on Computer Architecture. *In: Proceedings of the 1977 International Conference on Parallel Processing*.

HENNESSY, J.L., & PATTERSON, D.A. 2003. *Computer Architecture. A Quantitative Approach*. 3 edn. Morgan Kaufmann Publishers. Existe traducción al castellano de una edición anterior: Arquitectura de computadores: Un enfoque cuantitativo, McGraw-Hill, 1993.

HWANG, K. 1993. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill.

KELLER, R.M. 1975. Look Ahead Processors. *ACM Computing Surveys*, **7**(4).

KOGGE, P.M. 1981. *The Architecture of Pipelined Computers*. McGraw-Hill.

PATEL, J.H, & DAVIDSON, E.S. 1976. Improving the Throughput of a Pipeline by Insertion of Delays. *In: Proceedings of the Third Annual Computer Architecture Symposium*. IEEE num. 76CH 0143-5C.

PATTERSON, D.A., & HENNESSY, J.L. 2005. *Computer Organization and Design. The hardware/software interface*. 3 edn. Elsevier. Existe traducción al castellano de una edición anterior: Estructura y diseño de computadores. Inteficie circuitería/programación, Reverté, 2000.

SHAR, L.E. 1972. *Design and Scheduling of Statically Configured Pipelines*. Digital Systems, Lab Report SU-SEL-72-042. Stanford University.

STONE, H.S. 1990. *High-performance Computer Architecture*. 2 edn. Addison-Wesley.

SUSSENGUTH, E. 1971 (Jan.). *Instruction sequence control*. U.S. Patent num. 3,559,183.

TOMASULO, R.M. 1967. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal of Research and Development*, **11**(1).

Departamento de Informática Universidad de Valladolid

CUESTIONES Y PROBLEMAS

2.1 Un procesador convencional no segmentado tiene una velocidad de reloj de 2.5 GHz. y un CPI de 4. Se dispone de otro procesador segmentado, compatible con el anterior,

lineal y con 5 etapas. La velocidad del reloj de este nuevo procesador es de 1 GHz. Si un programa contiene 200 instrucciones y se ejecuta en ambos procesadores:

a) ¿Cuál es la ganancia de velocidad conseguida con el nuevo procesador?

b) Calcular la velocidad en MIPS para ambos procesadores.

c) Calcular la ganancia y la velocidad en MIPS del procesador segmentado en estado estacionario.

2.2 Considerar la ejecución de un programa de 20.000 instrucciones en un procesador segmentado lineal de 5 etapas con una velocidad de reloj de 3 GHz. No se deben tener en cuenta las pérdidas de rendimiento debidas a los diferentes tipos de conflictos:

a) Calcular la ganancia de velocidad de este procesador segmentado respecto a un procesador convencional equivalente cuya velocidad de reloj sea de 4 GHz.

b) Calcular la eficiencia y la productividad para el procesador segmentado.

c) Calcular la ganancia de velocidad, la eficiencia y la productividad del procesador segmentado en estado estacionario.

2.3 Considérese un procesador segmentado con la tabla de reservas mostrada en la figura 2.22:

a) ¿Cuales son las latencias prohibidas?

b) Calcular el vector de colisiones.

c) Dibujar el diagrama de estados para el vector de colisiones calculado.

d) Recorrer en el diagrama anterior el ciclo avaro (*greedy*).

e) Determinar la mínima latencia media (MAL) de ese diagrama.

2.4 Para cada uno de los vectores de colisiones siguientes:

01101101 y 0110010101

a) Construir el diagrama de estados.

b) Señalar en cada diagrama los ciclos avariciosos (*greedy*).

c) Encontrar el ciclo de mayor rendimiento. ¿Cuál es la mínima latencia media de ese ciclo?

2.5 Para la tabla de reservas mostrada en la figura 2.23, añadir los retardos que sean necesarios para conseguir una mayor velocidad de funcionamiento.

	1	2	3	4	5	6
A	X					
B		X				X
C			X			

Fig. 2.22.

Departamento de Informática Universidad de Valladolid

2.6 Considerar la tabla de reservas mostrada en la figura 2.24 correspondiente a un procesador segmentado de 4 segmentos:

a) Calcular las latencias prohibidas y el vector de colisiones.

b) Dibujar el diagrama de estados para el vector de colisiones calculado.

c) Calcular la mínima latencia media y el ciclo avaro.

d) Si la frecuencia de reloj de ese procesador fuera de 2.5 GHz: ¿Cuál sería la productividad de ese procesador?

e) Si fuera necesario, introducir retardos en el procesador para mejorar la latencia mínima.

f) Construir la tabla de reservas y el diagrama de estados para el nuevo procesador.

g) Para la misma velocidad de reloj, calcular la ganancia de velocidad del nuevo procesador comparándola con el anterior. Calcular también su productividad.

2.7 Confeccionar un procedimiento en algún lenguaje de alto nivel que, a partir de un vector de colisión, construya su diagrama de estados, calcule el ciclo avaro y el ciclo con la mínima latencia media.

2.8 Sea un procesador segmentado con 4 segmentos cuyas frecuencias relativas de riesgos de la instrucción i, respecto a las siguientes, son:

	Frecuencia	Número de ciclos perdidos
Instrucción $i+1$	25 %	2
Instrucción $i+2$	10 %	1

Suponiendo que la frecuencia de reloj de una máquina no segmentada equivalente es 1.5 veces mayor que el de esta máquina segmentada:

a) ¿Cuál es la ganancia de velocidad conseguida con la máquina segmentada si se ignoran los riesgos?

	1	2	3	4	5	6	7	8
A	X		X	X			X	
B		X			X			
C			X					X
D				X	X	X		
E			X			X		

Fig. 2.23.

	1	2	3	4	5	6	7
A	X		X				
B		X					X
C			X	X			
D				X			

Fig. 2.24.

Departamento de Informática Universidad de Valladolid

b) ¿Cuál es el número medio de ciclos perdidos por instrucción a causa de los riesgos?

c) ¿Cuál será la ganancia de velocidad teniendo en cuenta los riesgos?

d) Calcular el CPI de la máquina en este último caso.

e) Calcular la velocidad de reloj de la máquina segmentada, respecto a la no segmentada, para la que ambas máquinas tienen la misma productividad.

2.9 Sea el procesador segmentado de la figura 2.25 en el que se pueden ejecutar dos funciones:

f_1, en que se utilizan sucesivamente los segmentos S_1, S_4 y S_2

f_2, en que se utilizan sucesivamente los segmentos S_1, S_2, S_3, S_4 y S_2

a) Calcular la tabla de reservas para ambas funciones.

b) Calcular los vectores de colisiones cruzadas.

c) Calcular las matrices de colisiones para ambas funciones.

d) Dibujar el diagrama de estados.

2.10 Sea un procesador RISC segmentado con cuatro etapas. Estas etapas son:

- Lectura y decodificación de la instrucción.
- Búsqueda de operandos.
- Acceso a memoria.
- Ejecución de la operación con escritura del resultado en un registro

En este procesador existen los siguientes tipos de instrucciones:

1. Instrucciones aritméticas.
2. Instrucciones de almacenamiento.
3. Instrucciones de carga.
4. Bifurcaciones.

Tomando el procesador como un cauce multifunción dinámicamente configurable, y suponiendo que todas las instrucciones del mismo tipo utilizan los mismos segmentos:

a) Construir la tabla de reservas para cada operación.

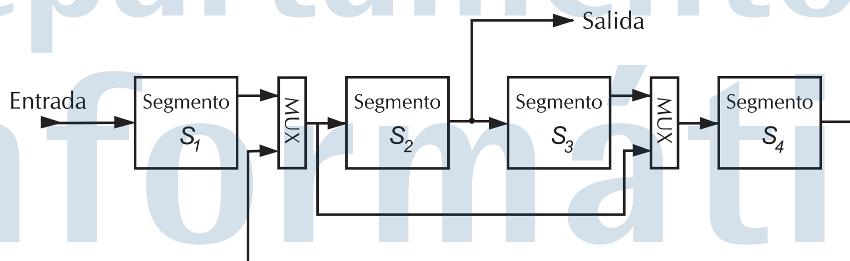


Fig. 2.25.

Departamento de Informática Universidad de Valladolid

b) Construir los vectores de colisiones cruzadas para estas operaciones.

c) Construir la matriz de colisiones para cada operación.

2.11 El siguiente fragmento de programa corresponde a una máquina segmentada con estructura registro-registro cuyas etapas son:

1. Lectura y decodificación de la instrucción.
2. Búsqueda de operandos.
3. Ejecución y escritura del resultado.

```
.....
STORE R4, A
INC   R0
MUL   R0, R2
STORE R2, B
ADD   R0, R2
STORE R2, C
LOAD  Z,  R3
LOAD  Q,  R5
ADD   R3, R5
ADD   R2, R5
STORE R5, Q
......
```

Si en el ensamblador de esta máquina el último operando es el destino:

a) Señalar los posibles riesgos por dependencias de datos existentes en este programa.

b) ¿Podría el compilador evitar alguna de las detenciones?

2.12 a) Suponer que las frecuencias de las instrucciones de control de flujo, referidas al conjunto de instrucciones de una máquina segmentada son las siguientes:

Saltos incondicionales, llamadas y retornos	10 %
Saltos condicionales	25 % (60 % efectivos)

Dicha máquina tiene 4 segmentos, y las instrucciones de control de flujo incondicionales se resuelven cuando termina el segundo segmento y las condicionales en el tercero. Suponiendo que la primera de las etapas de la segmentación puede llevarse a efecto siempre, con independencia del resultado de los saltos previos, e ignorando los demás tipos de riesgos. ¿Cuál es la pérdida de velocidad debida a los riesgos de control?

b) Repetir el apartado anterior suponiendo que sólo es irreversible la última etapa de segmentación.

2.13 Sea un procesador con cuatro segmentos y un porcentaje de instrucciones de control de flujo del 28 % de las cuales el 58 % son condicionales, el 28 % son iterativas y el resto incondicionales. De los saltos condicionales, el 60 % son efectivos y de los iterativos lo son el 90 %. Suponiendo que los saltos incondicionales se resuelven en la tercera etapa y los condicionales en la cuarta, calcular el tiempo medio de ejecución de las instrucciones prescindiendo de otros conflictos.

Departamento de Informática Universidad de Valladolid

2.14 Suponer un procesador segmentado en el que el 25 % de las instrucciones son saltos, de ellos el 60 % son condicionales, el 30 % son iterativos y el resto incondicionales. Los saltos condicionales se pueden predecir correctamente un 65 % de las ocasiones y los iterativos, un 85 %. La penalización de los saltos no predichos correctamente es de 4 ciclos y la del resto sólo de 1 ciclo. Calcular la eficiencia de la máquina prescindiendo de los efectos de otros riesgos.

2.15 Supóngase que las frecuencias de las instrucciones de control de flujo, referidas al conjunto de instrucciones de un procesador segmentado cuyo reloj funciona a 1 GHz., son las siguientes:

Saltos incondicionales, llamadas y retornos	10 %
Saltos condicionales	15 % (60 % efectivos)
Saltos iterativos	12 % (90 % efectivos)

Dicho procesador tiene 4 segmentos, y las instrucciones de control de flujo incondicionales se resuelven cuando termina el tercer segmento y las condicionales e iterativas, al finalizar el cuarto. Suponiendo que la emisión de las instrucciones se produce a partir de la tercera etapa, y que la instrucción no se emite hasta que se está seguro de que debe ejecutarse:

a) ¿Cuál será el CPI de la máquina si se prescinde de otras clases de riesgos?

b) Calcular la ganancia de velocidad de esta máquina referida a otra secuencial equivalente que funcione con un reloj de 1.5 GHz.

Departamento de Informática
Universidad de Valladolid