TEMA 2

El procesador segmentado. Riesgos

Las dependencias entre datos. Soluciones simples

Ana Pont

El procesador segmentado. Riesgos

Las dependencias entre datos. Soluciones simples

Riesgos de la segmentación

Hay situaciones, llamadas riesgos (*hazards*) que impiden que en una máquina segmentada se pueda ejecutar la siguiente instrucción del flujo de instrucciones durante su ciclo de reloj designado. Los riesgos reducen el rendimiento del procesador logrado por la segmentación.

Hay tres tipos de riesgos:

- *Riesgos estructurales*, surgen de conflictos en los recursos cuando el *hardware* no puede soportar todas las combinaciones posibles de instrucciones en ejecución solapadas simultáneamente. Por ejemplo en una UAL no se puede pedir realizar dos operaciones aritméticas al mismo tiempo, a no ser que esté replicada. Esto es, se produce un riesgo estructural cuando varias etapas intentan utilizar al mismo tiempo un elemento de la ruta de datos. Nuestra ruta de datos no tiene riesgos estructurales, pues aquellos elementos susceptibles de poder ser utilizados simultáneamente (operador aritmético-lógico, memoria, caminos de datos, etc.) están convenientemente replicados

- *Riegos por dependencia de datos*, surgen cuando una instrucción depende de los resultados de una instrucción anterior, de forma que, si se ejecutaran de forma solapada, los resultados podrían no ser los correctos.

- *Riesgos de control*, surgen de la segmentación de los saltos y otras instrucciones que cambian el contenido del registro CP. Lo que implica que las instrucciones siguientes que se están solapando con la de salto no deberían haberse iniciado.

Estos riesgos pueden provocar la necesidad de detener la segmentación. La diferencia principal entre detenciones en una máquina segmentada y en una no segmentada estriba en que hay múltiples instrucciones ejecutándose a la vez.

Una detención en una máquina segmentada requiere con frecuencia que se prosigan algunas instrucciones, mientras se retardan otras. Normalmente cuando una instrucción está retenida, todas las instrucciones posteriores a esta instrucción también se detienen. Las instrucciones anteriores pueden continuar, pero no se buscarán nuevas durante la detención.

Riesgos por dependencias de datos

Estos riesgos se presentan cuando se tienen en el procesador varias instrucciones ejecutándose, y algunas de las últimas en entrar necesita datos que todavía no han sido actualizados por las primeras. Por ejemplo, se considérese la ejecución de las siguientes instrucciones:

```
add $1 $2, $1
sub $3 $1 $4
```

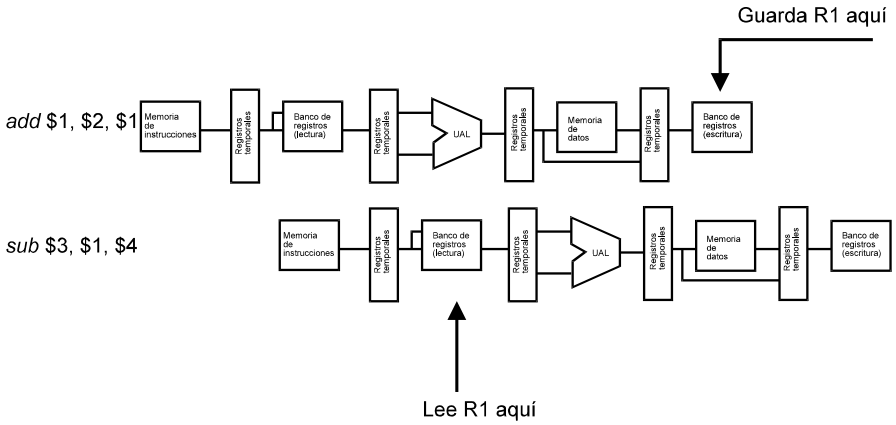

Figura 1. Riesgo por dependencia con el contenido de R1

A menos que se tomen precauciones para prevenirlo, la instrucción sub leerá el valor de **R1** antes de que este sea escrito por la instrucción add.

La Figura 2 muestra un ejemplo de código a través de su diagrama instrucciones/tiempo en el que se pueden observar varias dependencias de datos.

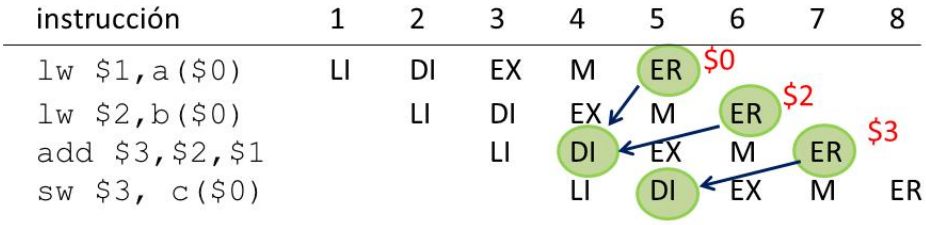

Figura 2. Diagrama instrucciones/tiempo para un código con dependencia de datos

El valor de $1 se obtiene en el ciclo 5 pero ha de ser leído por la instrucción *add* en el ciclo 4; el valor de $2 se escribe en el ciclo 6 pero ha de ser leído por la instrucción *add* en el ciclo 4. Finalmente, el valor de $3 se escribe en el ciclo 7 pero ha de ser leído por la instrucción *sw* en el ciclo 5.

Este problema puede resolverse mediante el software incluyendo instrucciones **NOP** entre las instrucciones afectadas por la dependencia. También puede solucionarse mediante el hardware, de una forma sencilla incluyendo **estados de espera**, es decir mediante **detención** del flujo de entrada hasta que se resuelva el conflicto o bien modificando la ruta de datos generando caminos alternativos denominados **cortocircuitos.**

Veamos a continuación algunas soluciones simples al problema de la dependencia de datos.

Solución sencilla por software: inserción de NOP

La aproximación más sencilla para resolver este conflicto es que al generar el código máquina el compilador detecte el conflicto e inserte las instrucciones NOP. La figura 3 muestra en el diagrama instrucciones/tiempo como el compilador inserta dos instrucciones NOP para resolver las dependencias de datos entre las instrucciones 1 y 2 con la instrucción 3 (add) causadas por las escrituras previas en los registros $0 y $2. También mediante dos instrucciones NOP se resuelve la

dependencia de datos causada por la instrucción 3 (add) al escribir en el registro $3 que posteriormente es leído por la instrucción 4 (sw).

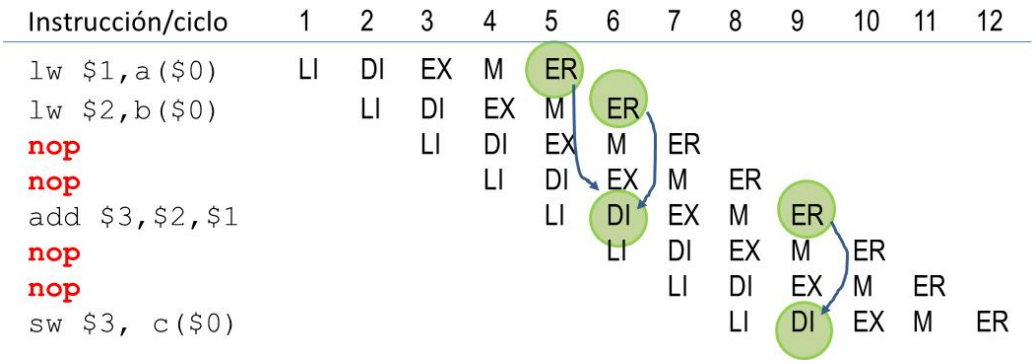

Figura 3. Solución de los conflictos por dependencias de datos mediante la inserción de NOP

Nótese que tras la inserción de las instrucciones NOP el código resultante tiene 4 instrucción más, total 8 instrucciones pero que el CPI para este fragmento de código sigue siendo 1:

$$CPI = \frac{ciclos - (K-1)}{I} = \frac{12-4}{8} = 1$$

Sin embargo, lo que si aumenta es el tiempo de ejecución necesario ya que ahora emplearemos 12 ciclos, 4 más que con el código original.

Solución sencilla por hardware: ciclos de parada

La aproximación más sencilla para resolver este conflicto a través del hardware es proceder a detener la instrucción en la segmentación hasta que se resuelva el riesgo. Con esta estrategia, primero se detecta el riesgo y, después se detienen las instrucciones en la segmentación hasta que se resuelve el este. Esta técnica se conoce también como inserción de ciclos de parada. En la Figura 4 se puede observar la utilización de los ciclos de parada para resolver este riesgo en el ejemplo anterior.

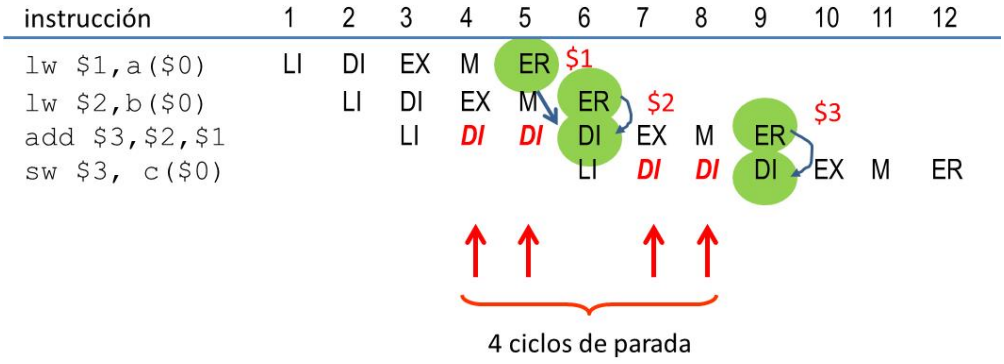

Figura 4. Solución de los conflictos por dependencias de datos mediante ciclos de parada

Vemos que en este caso el CPI aumenta:

$$CPI = \frac{ciclos - (K-1)}{I} = \frac{12-4}{4} = 2$$

Sin embargo, el tiempo de ejecución es el mismo puesto que empleamos 12 ciclos igual que con la solución anterior.

La solución mediante ciclos de parada o detención del flujo de instrucciones necesita realizar modificaciones en el hardware. Además de la lógica asociada a la detención de la dependencia es necesario modificar el control de forma que se inhiba la señal del reloj asociada a los registros de las etapas previas, mientras que no se debe inhibir en las etapas posteriores a fin de conseguir que se detenga la entrada de instrucciones pero no se impida a las que ya están dentro seguir avanzando hasta llegar a completarse. La figura 5 muestra un esquema sencillo de cómo debe actuar este control.

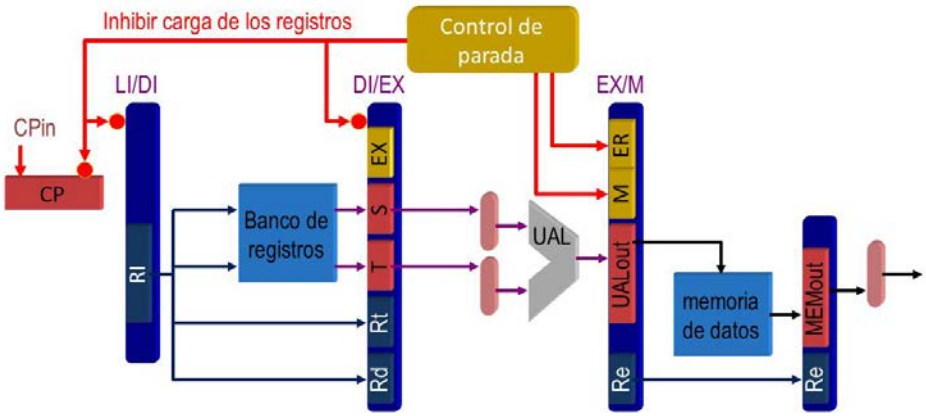

Figura 5. Inhibición de la señal de reloj para generar ciclos de parada

Valoración de las soluciones propuestas

Las soluciones propuestas son ambas sencillas y conservadoras, es decir tanto una como la otra esperan a que el problema se resuelva ya sea mediante el hardware o mediante el software.

En ambos casos el tiempo de ejecución es el mismo ya que se requieren los mismos ciclos para ejecutar un determinado programa. En el caso del ejemplo que nos ocupa, han sido 12 ciclos en ambos casos.

Cabe decir que el compilador puede ayudar a disminuir el tiempo de ejecución reordenando el código. Es decir, cambiando el orden de ejecución de las instrucciones siempre y cuando esto no altere el resultado final de las mismas y, ayudando de este modo a paliar o aliviar los efectos de las dependencias de datos. En este curso no vamos a entrar a fondo en la reordenación de código pero en algunos ejemplos de clase o del laboratorio la solución es tan evidente que de forma natural podemos sugerir el cambio en el orden de las instrucciones para minimizar o eliminar completamente este riesgo. Conviene por tanto, prestar atención a esta posibilidad.

Como diferencias más importantes entre ambas soluciones cabe destacar:

- ✓ La inserción de instrucciones inútiles incrementa el factor I, es decir, el número de instrucciones que se han de ejecutar es mayor, aunque estas no hagan trabajo útil.
- ✓ Los ciclos de parada incrementan el factor CPI, lo que tiene como efecto aumentar el tiempo que por término medio requiere una instrucción para ejecutarse.

Pero además cabe indicar que la solución hardware (inserción de ciclos de parada) conlleva una modificación importante en control del procesador de forma que la complejidad de la lógica de inserción de los ciclos de parada podría alargar el retardo de las etapas y habría que bajar la frecuencia del reloj, lo que afectaría negativamente a la Productividad.

Ejemplo

1. Para el fragmento de código siguiente detéctense las dependencias de datos y soluciónense las mismas tanto mediante inserción de NOP como ciclos de parada.

```
(1)   sw $2, A($1)
(2)   sub $3, $2, $3
(3)   sub $2, $2, $1
(4)   lw $6, 100($3)
```

Solución.

En este fragmento de código solo aparece una dependencia de datos: la instrucción (2) escribe su resultado en el registro $3. Este registro es leído posteriormente por la instrucción (4) para la calcular la dirección de memoria donde se almacenará el contenido de $6.

Para facilitar la respuesta en este tipo de cuestiones, conviene familiarizarse con una tabla como la siguiente:

Riesgos por dependencias de datos:

Riesgo	Registro implicado	Nº de instrucción en la que se escribe	Nº de instrucción en la que se lee
1	$3	(2)	(4)

Veamos el diagrama instrucciones/tiempo asociado a este fragmento cuando la solución se realiza mediante inserción de instrucciones NOP

I/T	1	2	3	4	5	6	7	8	9	10
sw	LI	DI	EX	M	ER					
sub		LI	DI	EX	M	ER				
sub			LI	DI	EX	M	ER			
NOP				LI	DI	EX	M	ER		
lw					LI	DI	EX	M	ER	

Para solucionar este conflicto hemos insertado una NOP entre las instrucciones (3) y (4) consiguiendo con eso dar tiempo suficiente a que el valor de $3 esté actualizado antes de que la instrucción `lw` lo utilice en el ciclo 6.

Para ejecutar correctamente este código hemos necesitado 9 ciclos de reloj.

El CPI resultante es 1.

$$CPI = \frac{ciclos - (K-1)}{I} = \frac{9-4}{5} = 1$$

Ahora mostramos el mismo diagrama instrucciones/tiempo cuando la solución empleada es a través de inserción de ciclos de espera:

I/T	1	2	3	4	5	6	7	8	9	10
sw	LI	DI	EX	M	ER					
sub		LI	DI	EX	M	ER				
sub			LI	DI	EX	M	ER			
lw				LI	LI	DI	EX	M	ER	

Nótese como se inserta una espera en el ciclo 5 y la instrucción `lw` no avanza en el procesador. Con esto se consigue que al llegar esta instrucción a su etapa de decodificación y lectura en registros (DI) la instrucción (2) ya haya actualizado su dato, hecho que ocurre en el ciclo 6.

Para ejecutar correctamente este código hemos necesitado 9 ciclos de reloj.

El CPI resultante es 1.25.

$$CPI = \frac{ciclos - (K-1)}{I} = \frac{9-4}{4} = 1.25$$