

Problema

Dado el siguiente fragmento de código:

```
ADDI R5,R0,#1
LD   R6,0(R5)
LD   R8,8(R5)
LD   R9,16(R5)
LD   R7,24(R5)
ADD  R1,R8,R9
ADD  R8,R9,R7
SD   0(R8),R1
LD   R8,0(R6)
```

- Señale todas las dependencias de datos existentes en el fragmento.
- ¿Existen dependencias de memoria? En caso afirmativo indique cuáles y a qué se deben.
- Renombre el código e indique qué dependencias permanecen.
- ¿Cómo se gestionan en un procesador superescalar las dependencias de datos y de memoria que permanecen tras el renombramiento?

Solución

a) En el fragmento de código, las dependencias de datos existentes son:

i1:	ADDI	R5,R0,#1	
i2:	LD	R6,0(R5)	// dependencia RAW con i1 por R5
i3:	LD	R8,8(R5)	// dependencia RAW con i1 por R5
i4:	LD	R9,16(R5)	// dependencia RAW con i1 por R5
i5:	LD	R7,24(R5)	// dependencia RAW con i1 por R5
i6:	ADD	R1,R8,R9	// dependencia RAW con i3 por R8
			// dependencia RAW con i4 por R9
i7:	ADD	R8,R9,R7	// dependencia RAW con i4 por R9
			// dependencia RAW con i5 por R7
			// dependencia WAR con i6 por R8
			// dependencia WAW con i3 por R8
i8:	SD	0(R8),R1	// dependencia RAW con i6 por R1
			// dependencia RAW con i3 por R8
			// dependencia RAW con i7 por R8
i9:	LD	R8,0(R6)	// dependencia RAW con i2 por R6
			// dependencia WAW con i3 por R8
			// dependencia WAW con i7 por R8
			// dependencia WAR con i6 por R8
			// dependencia WAR con i8 por R8

b) Existen dependencias ambiguas de memoria de las instrucciones i2, i3, i4 e i5 con i8. Las instrucciones de carga i2, i3, i4 e i5 leen las posiciones de memoria $M[0+R5]$, $M[8+R5]$, $M[16+R5]$, y $M[24+R5]$, respectivamente, mientras que la instrucción de almacenamiento i8 tiene que escribir en $M[0+R8]$, lo que implica la existencia de un riesgo WAR.

Entre la instrucción i8 e i9 existe otra dependencia ambigua de tipo RAW ya que se puede dar el caso de que el contenido de R8 y R5 coincidan y, por lo tanto, el destino y fuente de ambas instrucciones.

c) El código con los registros renombrados y en el que han desaparecido las dependencias falsas de datos es:

```
i1: ADDI    Rr1,R0,#1
i2: LD      Rr2,0(Rr1)    // dependencia RAW con i1 por Rr1
i3: LD      Rr3,8(Rr1)    // dependencia RAW con i1 por Rr1
i4: LD      Rr4,16(Rr1)   // dependencia RAW con i1 por Rr1
i5: LD      Rr5,24(Rr1)   // dependencia RAW con i1 por Rr1
i6: ADD     Rr6,Rr3,Rr4    // dependencia RAW con i3 por Rr3
                          // dependencia RAW con i4 por Rr4
i7: ADD     Rr7,Rr4,Rr5    // dependencia RAW con i4 por Rr4
                          // dependencia RAW con i5 por Rr5
i8: SD      0(Rr7),Rr6     // dependencia RAW con i6 por Rr6
                          // dependencia RAW con i7 por Rr7
i9: LD      Rr9,0(Rr2)    // dependencia RAW con i3 por Rr3
                          // dependencia RAW con i2 por Rr2
```

d) Las dependencias de datos RAW se eliminan mediante el mecanismo que establecen las estaciones de reserva y que obligan a que los operandos estén disponibles para poder emitir una instrucción.

El renombramiento de registros no elimina ninguna de las dependencias de memoria ya que al emitirse la instrucciones no es posible conocer si hay coincidencia en las direcciones. Para eliminar las dependencias de memoria falsas se establece el mecanismo de terminación ordenada con el buffer de almacenamiento. Las WAW se evitan, claramente, mediante un almacenamiento ordenado. Los riesgos WAR se evitan garantizando que una instrucción de almacenamiento posterior en el código a una carga, no escribe antes en memoria gracias al almacenamiento diferido.

Los riesgos de memoria RAW se gestionan mediante hardware adicional que permite la detección de la coincidencia de memoria y el reenvío del dato del almacenamiento (origen) hacia la carga (destino).