

Problema

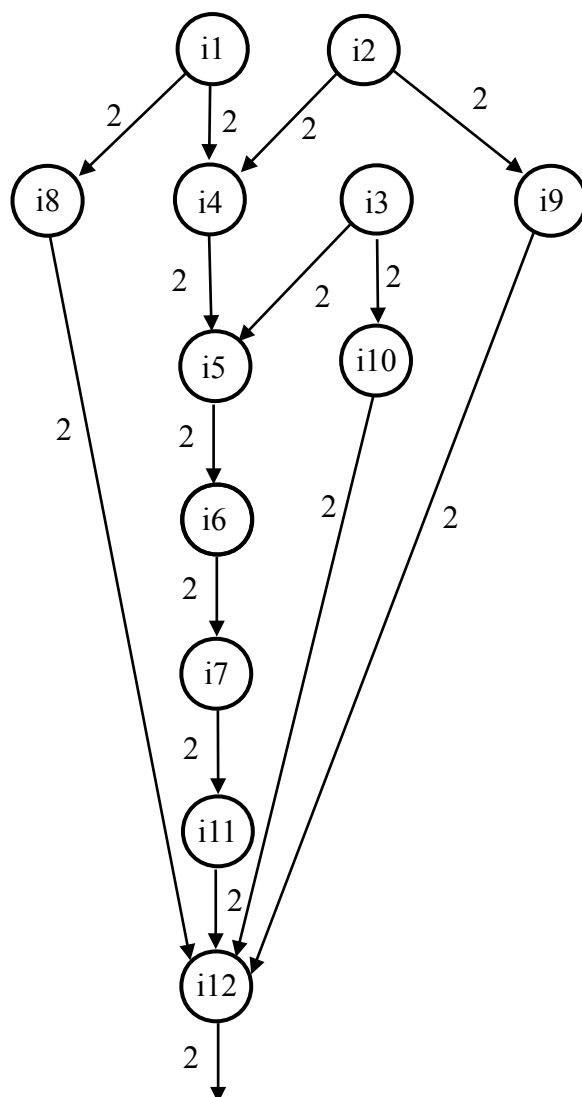
Dado el siguiente código:

```
inicio: LD      F0, 0(R2)      % i1
        LD      F2, 0(R4)      % i2
        LD      F4, 0(R6)      % i3
        ADDD    F0, F0, F2      % i4
        ADDD    F0, F0, F4      % i5
        DIVD    F0, F0, F8      % i6
        SD      0(R1), F0      % i7
        ADDI    R2, R2, #8      % i8
        ADDI    R4, R4, #8      % i9
        ADDI    R6, R6, #8      % i10
        ADDI    R1, R1, #8      % i11
        JUMP    inicio         % i12
```

- Obtenga el diagrama de flujo de datos de la secuencia. Considere que todas las instrucciones tienen una latencia de 2 ciclos.
- Transforme la secuencia en instrucciones VLIW para que pueda ser ejecutada por un procesador con un formato de instrucción que permite emitir simultáneamente operaciones a cualquiera de las cuatro unidades funcionales (son polivalentes). Considere las mismas latencias que en el apartado anterior y que puede efectuar los reordenamientos que crea oportunos, siempre que no afecten al resultado.
- Desenrolle el bucle original 4 veces y planifíquelo en forma de instrucciones VLIW teniendo en cuenta las características del procesador y las latencias. Utilice todos los registros que sean necesarios y compacte el código VLIW lo máximo posible.
- ¿Qué mejora en el rendimiento se ha obtenido si se compara el código del apartado b con el del c?

Solución

a)



b) Son posibles varias soluciones según se realice el ordenamiento de las operaciones en las instrucciones. En la solución que se propone, se han agrupado las instrucciones de incremento de los índices con la salvedad de R1. Si se optase por incrementar R1 antes de la instrucción de almacenamiento SD 0(R1), F0 sería necesario realizar un decremento negativo para compensar, es decir, SD -8(R1), F0.

	Unidad funcional 1	Unidad funcional 2	Unidad funcional 3	Unidad funcional 4
inicio:	LD F0, 0(R2)	LD F2, 0(R4)	LD F4, 0(R6)	ADDI R2, R2, #8
				ADDI R4, R4, #8
	ADDD F0, F0, F2			ADDI R6, R6, #8
	ADDD F0, F0, F4			
	DIVD F0, F0, F8			
	SD 0(R1), F0	ADDI R1, R1, #8		JMP inicio

c) La secuencia de código que se obtiene tras desenrollar 4 veces el bucle original es la siguiente:

```

inicio: LD    F0,0(R2)           // Iter 1
        LD    F2,0(R4)           // Iter 1
        LD    F4,0(R6)           // Iter 1
        LD    F10,8(R2)          // Iter 2
        LD    F12,8(R4)          // Iter 2
        LD    F14,8(R6)          // Iter 2
        LD    F20,16(R2)         // Iter 3
        LD    F22,16(R4)         // Iter 3
        LD    F24,16(R6)         // Iter 3
        LD    F30,24(R2)         // Iter 4
        LD    F32,24(R4)         // Iter 4
        LD    F34,24(R6)         // Iter 4
        ADDD  F0,F0,F2           // Iter 1
        ADDD  F0,F0,F4           // Iter 1
        DIVD  F0,F0,F8           // Iter 1
        ADDD  F10,F10,F12        // Iter 2
        ADDD  F10,F10,F14        // Iter 2
        DIVD  F10,F10,F8         // Iter 2
        ADDD  F20,F20,F22        // Iter 3
        ADDD  F20,F20,F24        // Iter 3
        DIVD  F20,F20,F8         // Iter 3
        ADDD  F30,F30,F32        // Iter 4
        ADDD  F30,F30,F34        // Iter 4
        DIVD  F30,F30,F8         // Iter 4
        SD    0(R1),F0           // Iter 1
        SD    8(R1),F10          // Iter 2
        SD    16(R1),F20         // Iter 3
        SD    24(R1),F30         // Iter 4
        ADDI  R2,R2,#32
        ADDI  R4,R4,#32
        ADDI  R6,R6,#32
        ADDI  R1,R1,#32
        JUMP  inicio

```

Una transformación en código VLIW realizando la máxima compactación posible se muestra en la siguiente tabla. Los colores representan: 1ª iteración (negro) e instrucciones auxiliares, 2ª iteración (rojo), 3ª iteración (verde), 4ª iteración (morado).

	Unidad funcional 1	Unidad funcional 2	Unidad funcional 3	Unidad funcional 4
inicio:	LD F0, 0(R2)	LD F2, 0(R4)	LD F10, 8(R2)	LD F12, 8(R4)
	LD F20, 16(R2)	LD F22, 16(R4)	LD F30, 24(R2)	LD F32, 24(R4)
	ADDD F0, F0, F2	LD F4, 0(R6)	ADDD F10, F10, F12	LD F14, 8(R6)
	ADDD F20, F20, F22	LD F24, 16(R6)	ADDD F30, F30, F32	LD F34, 24(R6)
	ADDD F0, F0, F4		ADDD F10, F10, F14	
	ADDD F20, F20, F24		ADDD F30, F30, F34	
	DIVD F0, F0, F8		DIVD F10, F10, F8	ADDI R2, R2, #32
	DIVD F20, F20, F8		DIVD F30, F30, F8	ADDI R4, R4, #32
	SD 0(R1), F0		SD 8(R1), F10	ADDI R6, R6, #32
	SD 16(R1), F20	ADDI R1, R1, #32	SD 24(R1), F30	JMP inicio

d) El código consume 10 instrucciones VLIW ocupando un total de 160 bytes (10 instrucciones * 16 bytes por instrucción). En comparación con el código VLIW sin desenrollar, el rendimiento, prácticamente, se ha incrementado por 4 ya que el número de ciclos por iteración es de 11 en comparación con el caso sin desenrollar que es 10 ciclos pero procesando 1 único elemento, y no 4 como en el caso que nos ocupa.