

ARQUITECTURA DE COMPUTADORES GRADO EN INGENIERÍA INFORMÁTICA



PROBLEMAS DEL BLOQUE 2 SOLUCIONES

TEMA 2-1: Planificación Estática de Instrucciones	2
TEMA 2-2: Planificación Dinámica de Instrucciones. Ejecución Fuera de Orden.....	20
TEMA 2-3: Ejecución Especulativa de Instrucciones.....	37
TEMA 2-4: Procesadores Superescalares	55
Agradecimientos.....	76

TEMA 2-1: Planificación Estática de Instrucciones

Problema 2-1-1 (HP96, ex.4.1, pp.362). Enumerar todas las dependencias (salida, antidependencias, y verdaderas) en el siguiente fragmento de código. Indicar si las dependencias verdaderas son de tipo trans-iteración o no. Mostrar por qué el bucle no es paralelizable.

```
for (i=2; i<100;i=i+1) {
    a[i]=b[i]+a[i];    /* S1 */
    c[i-1]=a[i]+d[i];  /* S2 */
    a[i-1]=2 * b[i];    /* S3 */
    b[i-1]=2 * b[i];    /* S4 */
}
```

Solución.

Primero escribamos el mismo código en ensamblador para analizarlo mejor. Supondremos que los elementos de los vectores a[], b[] y c[] son todos enteros y por lo tanto, los incrementos del puntero que hace referencia en ellos (en nuestro caso R2) aumentará de 4 en 4.

```
1          LA      R1, A
2          LA      R2, B
3          LA      R3, C
4          LA      R4, D
5          ADDI    R1, R1, 8
6          ADDI    R2, R2, 8
7          ADDI    R3, R3, 8
8          ADDI    R4, R4, 8
9          ADDI    R5, R0, 98

10     Loop:      LW      R6, 0(R1)
11              LW      R7, 0(R2)
12              ADD     R8, R6, R7
13              SW      R8, 0(R1)          ;a[i] = b[i] + a[i];

14              LW      R6, 0(R1)
15              LW      R9, 0(R4)
16              ADD     R10, R6, R9
17              SW      R10, -4(R3)        ;c[i-1] = a[i] + d[i];

18              MULI    R11, R7, 2
19              SW      R11, -4(R1)        ;a[i-1] = 2 * b[i];

20              SW      R11, -4(R2)        ;b[i-1] = 2 * b[i];

21              SUBI    R5, R5, 1
22              ADDI    R1, R1, 4
23              ADDI    R2, R2, 4
24              ADDI    R3, R3, 4
25              ADDI    R4, R4, 4
26              BEQZ    R5, Loop
```

Ahora analicemos este código y encontremos las dependencias. En la siguiente tabla se indican para cada instrucción del programa si existe una dependencia y de qué tipo es.

Instrucción	Nº instrucción, dependencias	Nº instrucción, dependencias	Nº instrucción, dependencias	Nº instrucción, dependencias	Nº instrucción, dependencias	Nº instrucción, dependencias
1	5 RAW, WAW	10 RAW	13 RAW	14 RAW	19 RAW	22 RAW, WAW
2	6 RAW, WAW	11 RAW	20 RAW	23 RAW, WAW		
3	7 RAW, WAW	17 RAW	24 RAW, WAW			
4	8 RAW, WAW	15 RAW	25 RAW, WAW			
5	1 RAW, WAW	10 RAW	13 RAW	14 RAW	19 RAW	22 RAW, WAW
6	2 RAW, WAW	11 RAW	20 RAW	23 RAW, WAW		
7	3, RAW, WAW	17 RAW	24 RAW, WAW			
8	4 RAW, WAW	15 RAW	25 RAW, WAW			
9	21 RAW, WAW	26 RAW				
10	1 RAW	10 WAW	12 RAW	14 WAW	16 RAW	22 RAW, WAW
11	2 RAW	11 WAW	12 RAW	18 RAW	23 RAW, WAW	
12	10 RAW	11 RAW	12 WAW	13 RAW	14 WAR	
13	1 RAW	5 RAW	12 RAW	22 RAW		
14	10 WAW	14 WAW	16 RAW	22 RAW		
15	4 RAW	8 RAW	15 WAW	16 RAW	25 RAW	
16	10 RAW	14 RAW	15 RAW	16 WAW	17 RAW	
17	3 RAW	7 RAW	16 RAW	24 RAW		
18	11 RAW	18 WAW	19 RAW	20 RAW		
19	1 RAW	5 RAW	18 RAW	22 RAW		
20	2 RAW	6 RAW	18 RAW	23 RAW		
21	9 RAW	21 RAW, WAW	26 RAW			
22	1 RAW, WAW	5 RAW, WAW	10 RAW	13 RAW	14 RAW	19 RAW
23	2 RAW, WAW	6 RAW, WAW	11 RAW	20 RAW	23 RAW, WAW	
24	3 RAW, WAW	7 RAW, WAW	17 RAW	24 RAW, WAW		
25	4 RAW, WAW	8 RAW, WAW	15 RAW	25 RAW, WAW		
26	9 RAW	21 RAW				

NOTA: En la fila de la instrucción 22 faltan 2 dependencias con ella misma de tipo RAW y WAW. La dependencia RAW se produce por trans-iteración. En negrita aparecen marcadas las dependencias verdaderas (RAW) que son causadas por trans-iteración, es decir que se producen entre una iteración x y su siguiente iteración $x+1$.

No se pueden paralelizar distintas iteraciones del bucle debido a las dependencias trans-iteración. Observar que en cada iteración se calcula $a[i]$, $b[i]$ y $a[i-1]$. Paralelizar este bucle significa que cada iteración se puede realizar independientemente de cualquier otra. Si ejecutáramos distintas iteraciones en paralelo podría ocurrir que dos iteraciones actualizarían una misma variable con valores distintos, como por ejemplo en este problema la variable $a[]$. Cuando por ejemplo $i=2$, se actualiza $a[i]=a[2]$, y cuando $i=3$ se actualiza $a[i-1]=a[2]$. Por otro lado, en cada iteración existe una dependencia verdadera entre las instrucciones 10 y 12, la cual no permite ningún reordenamiento entre ellas y además impide también que se puedan ejecutar en paralelo.

Problema 2-1-2 (HP96, ex.4.3, pp.363). Para el siguiente fragmento de código, enumerar todas las dependencias de control. Para cada una de ellas, indica si la sentencia puede ser planificada antes de la sentencia "if". Suponer que todas las referencias de datos tienen valores definidos antes de su uso, y que sólo b y c se usarán de nuevo después de este fragmento. Ignorar cualquier posible excepción.

```
if ( a > c ) {
    d = d + 5;
    a = b + d + e; }
else {
    e = e + 2;
    f = f + 2;
    c = c + f;
}
b = a + f;
```

Solución.

Antes de nada, escribamos el código en ensamblador que surgiría del código anterior. Usamos los registros R1-R6 como base para las direcciones de las variables, y los R7-R12 como los contenedores.

1		BNGT	R7, R9, else	
2		LW	R10, 0(R4)	
3		ADDI	R10, R10, 5	
4		SW	R10, 0(R4)	;d = d + 5
5		LW	R8, 0(R2)	
6		LW	R11, 0(R5)	
7		ADD	R7, R8, R10	
8		ADD	R7, R7, R11	
9		SW	R7, 0(R1)	;a = b + d + e
10		J	endif	
11	else:	LW	R11, 0(R5)	
12		ADDI	R11, R11, 2	
13		SW	R11, 0(R5)	;e = e + 2
14		LW	R12, 0(R6)	
15		ADDI	R12, R12, 2	
16		SW	R12, 0(R6)	;f = f + 2
17		ADD	R9, R9, R12	
18		SW	R9, 0(R3)	;c = c + f
19	endif:	LW	R12, 0(R6)	
20		ADD	R8, R7, R12	
21		SW	R8, 0(R2)	;b = a + f

Las dependencias de control son aquellas que aparecen debido a que un programa contiene saltos que pueden modificar el flujo secuencial de un programa, y dependiendo del flujo del programa unas instrucciones se ejecutan y otras no. En este ejercicio nos aparecen dos grupos de instrucciones con dependencias de control, uno que depende de la instrucción BNGT y otro que depende de la instrucción J.

El adelantamiento de instrucciones se puede realizar siempre y cuando el flujo del programa no cambie. “ $d = d + 5$ ” se puede adelantar antes del if porque no modifican el resultado del programa. El código queda como sigue:

```

2          LW      R10, 0(R4)
3          ADDI    R10, R10, 5
4          SW      R10, 0(R4)          ;d = d + 5
5          LW      R8, 0(R2)
6          LW      R11, 0(R5)
14         LW      R12, 0(R6)

1          BNGT    R7, R9, else

7          ADD     R7, R8, R10
8          ADD     R7, R7, R11
9          SW      R7, 0(R1)          ;a = b + d + e

10         J       endif

12  else:      ADDI    R11, R11, 2
13          SW      R11, 0(R5)          ;e = e + 2
15          ADDI    R12, R12, 2
16          SW      R12, 0(R6)          ;f = f + 2
17          ADD     R9, R9, R12
18          SW      R9, 0(R3)          ;c = c + f

19  endif:      LW      R12, 0(R6)
20          ADD     R8, R7, R12
21          SW      R8, 0(R2)          ;b = a + f

```

Podemos apreciar que las instrucciones que se ejecutan antes de los saltos son sólo las cargas y las que no modifican el resultado del programa. La instrucción 11 se ha quitado debido a que era la misma que la 6.

Problema 2-1-3 (HP96, ex.4.4, pp.363). Suponiendo que las latencias de segmentación en la ruta de datos del procesador escalar DLX son las siguientes:

Operación ALU PF	→	Otra operación ALU PF	: 3 ciclos
Operación ALU PF	→	Almacenamiento doble palabra	: 2 ciclos
Carga doble palabra	→	FP Operación ALU PF	: 1 ciclos
Carga doble palabra	→	Almacenamiento doble palabra	: 0 ciclos

Desenrolla el siguiente bucle tantas veces como sea necesario para planificarlo sin retardos. Suponer que los saltos condicionales tienen un retardo de 1 ciclo en su resolución (saltos retardados). Mostrar la planificación suponiendo que el número de iteraciones es par. El bucle realiza la siguiente operación: $Y[i] = a X[i] + Y[i]$.

```

loop:    LD      F0,0(R1)    ; X[i]
         MULTD   F0,F0,F2    ; * a
         LD      F4,0(R2)    ; Y[i]
         ADDD    F0,F0,F4    ; a X + Y
         SD      0(R2),F0    ;
         SUBI    R1,R1,8     ; puntero X[i]
         SUBI    R2,R2,8     ; puntero Y[i]
         BNEQZ   R1,loop

```

Solución.

Desenrollamos dos veces el bucle y reordenamos las instrucciones:

```

loop:    LD      F0,  0(R1)
         LD      F6, -8(R1)
         MULTD   F0, F0, F2
         MULTD   F6, F6, F2
         LD      F4,  0(R2)
         LD      F8, -8(R2)
         ADDD    F0, F0, F4
         ADDD    F6, F6, F8
         SUBI    R1, R1, 16
         SD      F0,  0(R2)
         SD      F6, -8(R2)
         BNEZ    R1,loop
         SUBI    R2, R2, 16

```

Observamos que desenrollando dos veces se obtienen bastantes instrucciones como para no necesitar retardos debido a las latencias. Además, el desenrollamiento no requiere código prólogo o epílogo ya que se nos hace suponer que el número de iteraciones es par.

PREGUNTA 2-1-4 (Examen Final 13-1-2006). Codifica lo más eficientemente posible el siguiente programa que puede ejecutar el procesador DLX32p en instrucciones VLIW para que puedan ser ejecutadas por el procesador DLX32vliw. Utiliza las técnicas de planificación estática de instrucciones: Desenrollamiento de Bucles y Desenrollamiento Simbólico de Bucles (Software Pipelining).

```

Loop:  LD    F10, 0(R10)
        SUBD F14, F10, F12
        SD   0(R10), F14
        SUBI R10, R10, 8
        BNEZ R10, Loop

```

Para ello, considera las siguientes latencias de las instrucciones dependientes:

<i>Instrucción produce resultado</i>	<i>Instrucción utiliza resultado</i>	<i>Latencia (ciclos)</i>
operación ALU-FP	operación ALU-FP	3
operación ALU-FP	Almacenamiento DW	2
Carga DW	operación ALU-FP	1
Carga DW	Almacenamiento DW	0
operación enteros	operación enteros	0

Cada instrucción VLIW se compone de las siguientes instrucciones DLX32: 2 accesos a memoria, 2 operaciones en punto flotante, 1 operación de enteros en la que se incluyen saltos condicionales. Demuestra que la ejecución del programa es la más eficiente posible.

Solución.

Desenrollamos el código 9 veces y se realiza el renombramiento de registros de las tres primeras iteraciones y se repite para las otras 6 iteraciones, manteniendo la memoria desambiguada.

LOOP:

```

1  LD    F0,0(R10)           ; Iteración i
2  SUBD  F4,F0,F2
3  SD    0(R10),F4

4  LD    F6,-8(R10)          ; Iteración i+1
5  SUBD  F8,F6,F2
6  SD    -8(R10),F8

7  LD    F10,-16(R10)        ; Iteración i+2
8  SUBD  F12,F10,F2
9  SD    -16(R10),F12

10 LD    F0,-24(R10)         ; Iteración i+3
11 SUBD  F4,F0,F2
12 SD    -24(R10),F4

13 LD    F6,-32(R10)         ; Iteración i+4
14 SUBD  F8,F6,F2
15 SD    -32(R10),F8

16 LD    F10,-40(R10)        ; Iteración i+5

```

```

17  SUBD F12,F10,F2
18  SD   -40(R10),F12

19  LD   F0,-48(R10)    ; Iteración i+6
20  SUBD F4,F0,F2
21  SD   -48(R10),F4

22  LD   F6,-56(R10)    ; Iteración i+7
23  SUBD F8,F6,F2
24  SD   -56(R10),F8

25  LD   F10,-64(R10)   ; Iteración i+8
26  SUBD F12,F10,F2
27  SD   -64(R10),F12

28  SUBI R10,R10,#72
29  BNEZ R10,LOOP

```

Aplicando Desenrollamiento Simbólico de Bucles:

LOOP:

```

1  SD   0(R10),F4 ; Almacena M[i]
2  SUBD F4,F0,F2 ; Suma a M[i+3]
3  LD   F0,-48(R10); Carga M[i+6]

4  SD   -8(R10),F8 ; Almacena M[i+1]
5  SUBD F8,F6,F2 ; Suma a M[i+4]
6  LD   F6,-56(R10); Carga M[i+7]

7  SD   -16(R10),F12 ; Almacena M[i+2]
8  SUBD F12,F10,F2 ; Suma a M[i+5]
9  LD   F10,-64(R10); Carga M[i+8]

10 SUBI R10,R10,#24
11 BNEZ R10,LOOP

```

Codificación en instrucciones VLIW para el procesador DLX32vliw, el cuerpo del bucle principal quedaría expresado de la siguiente forma:

<i>Acceso Memoria 1</i>	<i>Acceso Memoria 2</i>	<i>Operación FP - 1</i>	<i>Operación FP - 2</i>	<i>Int/ salto</i>	<i>Ciclo</i>
LD F0,-48(R10)	SD 0(R10),F4	SUBD F4,F0,F2			1
LD F6,-56(R10)	SD -8(R10),F8	SUBD F8,F6,F2	SUBI R10,R10,#24		2
LD F10,-40(R10)	SD 8(R10),F12	SUBD F12,F10,F2	BNEZ R10,LOOP		3

Prestaciones: 3 resultados en 3 ciclos, 1 ciclo por iteración original, 3.6 operaciones por ciclo (11 operaciones / 3 ciclos), 73% eficiencia (11 instrucciones simples codificadas / 15 instrucciones simple posibles). Requiere sólo 7 registros de punto flotante (FP). Estas prestaciones quedarían sólo ligeramente empeoradas por la existencia del código del Prólogo y el Epílogo.

PREGUNTA 2-1-5 (Examen Parcial 15-1-2010). Aplica las técnicas de desenrollamiento de bucles y la de planificación de trazas utilizando la arquitectura *DLX32vliw* para generar el código ensamblador de la siguiente subrutina. Suponer que la vía “if” de la sentencia `if-then-else` es la que se ejecuta el 95% de las iteraciones. Analiza de forma cuantitativa las prestaciones del procesador *DLX32vliw* con respecto al procesador *DLX32mf*.

```
subrutina(float A[1000], float B[1000], float C[1000]){
    int i;
    float acumulador = 0;
    for (i=0; i<1000; i++){
        A[i] += (B[i] + C[i]);
        if (A[i] > 100) A[i] = 100;
        else A[i] = 50;
        acumulador += A[i]; }
}
```

Utilizar para la generación de código 32 registros de enteros, 32 registros de punto flotante, todos ellos de 32 bits, el registro de estado `FPcond`, así como las siguientes instrucciones:

Nemónico	Nombre	Descripción RTL	Latencia en ciclos por dependencia de ...		Tipo de instrucción	Observaciones
			Datos	Control		
<code>lwcl Fd, Rs, inm</code>	Carga dato punto flotante 32 bits	$Fd \leftarrow M[inm + (RS)]$	1	0	Enteros(ENT), Registro-Inmediato	
<code>swcl Fd, Rs, inm</code>	Almacenar dato punto flotante 32 bits	$Fd \rightarrow M[inm + (RS)]$	0	0	Enteros(ENT), Registro-Inmediato	
<code>add.s Fd, Fs, Ft</code>	Suma datos punto flotante 32 bits	$Fd \leftarrow Fs + Ft$	3	0	Flotante (FP), Registro - Registro	
<code>addi Rt, Rs, inm</code>	Suma con inmediato	$Rt \leftarrow Rs + inm$	0	0	Enteros(ENT), Registro - Registro	
<code>subi Rt, Rs, inm</code>	Resta con inmediato	$Rt \leftarrow Rs - inm$	0	0	Enteros(ENT), Registro - Registro	
<code>c.lt.s Fs, Ft</code>	Comparación menor-que de registros flotantes de 32 bits con inicialización del registro estado <code>FPcond</code>	$FPcond = (Fs < Ft) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <code>FPcond</code> se inicializa en la etapa EX
<code>c.gt.s Fs, Ft</code>	Comparación mayor-que de registros flotantes de 32 bits con inicialización del registro estado <code>FPcond</code>	$FPcond = (Fs > Ft) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <code>FPcond</code> se inicializa en la etapa EX
<code>j inm</code>	Salto incondicional	$PC \leftarrow inm$	0	1	Enteros (ENT), Saltos	Tiene un ciclo de salto retardado
<code>bne Rs, Rt, inm</code>	Salto condicional	$If (Rs \neq Rt), PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
<code>beq Rs, Rt, inm</code>	Salto condicional	$If (Rs == Rt), PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
<code>bclfi inm</code>	Salto condicional utilizando registro de estado <code>FPcond</code>	$If (!FPcond), PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Inmediato	El estado del registro <code>FPcond</code> se consulta en la etapa ID
<code>bclti inm</code>	Salto condicional utilizando registro de estado <code>FPcond</code>	$If (FPcond), PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Inmediato	El estado del registro <code>FPcond</code> se consulta en la etapa ID
<code>nop</code>	No-Operación	No modifica el estado de registros	0	0		Equivale a un ciclo de penalización

Ayuda: suponer que los registros involucrados en el cómputo del bucle `for` que requieren una inicialización previa están inicializados utilizando las instrucciones apropiadas, las cuales no tienen que coincidir con las que aparecen en la tabla. Indicar qué registros se inicializan y cuáles son los valores inicializados.

SOLUCIÓN PREGUNTA 2-1-5 (más extensa de lo que se solicita)Código DLX32mf sin optimizar

Tiempo: 29 ciclos (tanto la vía IF como la vía ELSE)

Ciclos	Etiqueta	Instrucción	Comentarios
		$R1 \leftarrow 0$	Prólogo: inicialización del puntero de memoria
		$R2 \leftarrow 1000$	Prólogo: inicialización del número de iteraciones
		$F0 \leftarrow 100$	Prólogo: inicialización de la constante utilizada en comparación
		$F1 \leftarrow 0$	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		$F2 \leftarrow 50$	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		$F3 \leftarrow 0$	Prólogo: acumulador
1	INICIO:	lwcl F4, R1, 0	$F4 \leftarrow A[i], i \equiv R4$
2		lwcl F5, R1, 4000	$F5 \leftarrow B[i], i \equiv R4$
3		lwcl F6, R1, 8000	$F6 \leftarrow C[i], i \equiv R4$
4		add.s F4, F4, F5	$F4 \leftarrow A[i] + B[i]$
5,6,7		3 x nop	3 ciclos de penalización por la dependencia de F4
8		add.s F4, F4, F6	$F4 \leftarrow A[i] + B[i] + C[i]$
9,10,11		3 x nop	3 ciclos de penalización por la dependencia de F4
12		c.gt.s F4, F0	$FPcond = (F4 > 100) ? 1 : 0$
13		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
14		bclf ELSE	Salto condicional sii "FPcond = 0"
15		1 x nop	Salto retardado
16	IF:	add.s F4, F1, F0	$A[i] \leftarrow 100$
17		1 x nop	1 nop por dependencia de F4 para llegar a 3 ciclos
18		j ACCUMULADOR	Salto incondicional que evita el código de la vía ELSE
19		1 x nop	Salto retardado
16	ELSE:	add.s F4, F1, F2	$A[i] \leftarrow 50$
17,18,19		3 x nop	3 ciclos de penalización por la dependencia de F4
20	ACCUMULADOR:	add.s F3, F3, F4	acumulador += A[i]
21		swcl F4, R1, 0	Almacena A[i] en memoria
22,23		2 x nop	2 ciclos de penalización por la dependencia de F3
24		swcl F3, R0, 12000	Almacena acumulador en memoria
25		addi R1, R1, 4	Aumenta el puntero de memoria de los tres vectores A, B y C
26		subi R2, R2, 1	Se reduce el número de iteraciones que restan por realizar
27		1 x nop	Penalización porque subi se ejecuta en etapa EX y bne se resuelve en etapa ID
28		bne R2, R0, INICIO	Salto condicional al inicio de la iteración sii $R2 \neq 0$
29		1 x nop	Salto retardado

Código DLX32mf con Planificación de Trazas

Tiempo: 29 ciclos

Ciclos	Etiqueta	Instrucción	Comentarios
		R1 \leftarrow 0	Prólogo: puntero de memoria
		R2 \leftarrow 1000	Prólogo: número de iteraciones
		F0 \leftarrow 100	Prólogo: constante utilizada en comparación
		F1 \leftarrow 0	Prólogo: constante utilizada para inicializar registros de punto flotante
		F2 \leftarrow 50	Prólogo: constante utilizada para inicializar registros de punto flotante
		F3 \leftarrow 0	Prólogo: acumulador
1	INICIO:	lwcl F4, R1, 0	F4 \leftarrow A[i], i \equiv R4
2		lwcl F5, R1, 4000	F5 \leftarrow B[i], i \equiv R4
3		lwcl F6, R1, 8000	F6 \leftarrow C[i], i \equiv R4
4		add.s F4, F4, F5	F4 \leftarrow A[i] + B[i]
5,6,7		3 x nop	3 ciclos de penalización por la dependencia de F4
8		add.s F4, F4, F6	F4 \leftarrow A[i] + B[i] + C[i]
9,10,11		3 x nop	3 ciclos de penalización por la dependencia de F4
12		c.gt.s F4, F0	FPcond = (F4 > 100) ? 1 : 0
13		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
14		bclf ELSE	Salto condicional sii "FPcond = = 0" (sólo el 5% de las iteraciones)
15		1 x nop	Salto retardado
16	IF:	add.s F4, F1, F0	A[i] \leftarrow 100
17,18,19		3 x nop	3 ciclos de penalización por la dependencia de F4
20	ACCUMULADOR:	add.s F3, F3, F4	acumulador += A[i]
21		swcl F4, R1, 0	Almacena A[i] en memoria
22,23		2 x nop	2 ciclos de penalización por la dependencia de F3
24		swcl F3, R0, 12000	Almacena acumulador en memoria
25		addi R1, R1, 4	Aumenta el puntero de memoria de los tres vectores A, B y C
26		subi R2, R2, 1	Se reduce el número de iteraciones que restan por realizar
27		1 x nop	Penalización porque subi se ejecuta en etapa EX y bne se resuelve en etapa ID
28		bne R2, R0, INICIO	Salto condicional al inicio de la iteración sii R2 != 0
29		1 x nop	Salto retardado
		j FUERA-BUCLE	Salto incondicional cuando termina el bucle
		1 x nop	Salto retardado
16	ELSE:	add.s F4, F1, F2	Código Compensación: A[i] \leftarrow 50
17		1 x nop	Código Compensación: 1 nop por dependencia de F4 para llegar a 3 ciclos
18		j ACCUMULADOR	Código Compensación: Salto incondicional que evita el código de la vía ELSE
19		1 x nop	Código Compensación: Salto retardado
	FUERA-BUCLE:		

Código DLX32mf con Reordenación

Tiempo: 24 ciclos (tanto la vía IF como la vía ELSE)

Reducción ciclos = $(29-24)/29=17.2\%$

Ciclos	Etiqueta	Instrucción	Comentarios
		$R1 \leftarrow 0$	Prólogo: inicialización del puntero de memoria
		$R2 \leftarrow 1000$	Prólogo: inicialización del número de iteraciones
		$F0 \leftarrow 100$	Prólogo: inicialización de la constante utilizada en comparación
		$F1 \leftarrow 0$	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		$F2 \leftarrow 50$	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		$F3 \leftarrow 0$	Prólogo: acumulador
1	INICIO:	$lwcl\ F4, R1, 0$	$F4 \leftarrow A[i], i \equiv R4$
2		$lwcl\ F5, R1, 4000$	$F5 \leftarrow B[i], i \equiv R4$
3		$lwcl\ F6, R1, 8000$	$F6 \leftarrow C[i], i \equiv R4$
4		$add.s\ F4, F4, F5$	$F4 \leftarrow A[i] + B[i]$
5,6,7		$3\ x\ nop$	3 ciclos de penalización por la dependencia de F4
8		$add.s\ F4, F4, F6$	$F4 \leftarrow A[i] + B[i] + C[i]$
9,10,11		$3\ x\ nop$	3 ciclos de penalización por la dependencia de F4
12		$c.gt.s\ F4, F0$	$FPcond = (F4 > 100) ? 1 : 0$
13		$1\ x\ nop$	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
14		$bclf\ ELSE$	Salto condicional sii "FPcond = 0"
15		$1\ x\ nop$	Salto retardado
16	IF:	$add.s\ F4, F1, F0$	$A[i] \leftarrow 100$
17		$addi\ R1, R1, 4$	Aumenta el puntero de memoria de los tres vectores A, B y C
18		$j\ ACCUMULADOR$	Salto incondicional que evita el código de la vía ELSE
19		$subi\ R2, R2, 1$	Se reduce el número de iteraciones que restan por realizar
16	ELSE:	$add.s\ F4, F1, F2$	$A[i] \leftarrow 50$
17		$subi\ R2, R2, 1$	Se reduce el número de iteraciones que restan por realizar
18		$addi\ R1, R1, 4$	Aumenta el puntero de memoria de los tres vectores A, B y C
19		$1\ x\ nop$	1 ciclos de penalización por la dependencia de F4
20	ACCUMULADOR:	$add.s\ F3, F3, F4$	$acumulador += A[i]$
21		$swcl\ F4, R1, -4$	Almacena A[i] en memoria
22		$1\ x\ nop$	1 ciclos de penalización por la dependencia de F3
23		$bne\ R2, R0, INICIO$	Salto condicional al inicio de la iteración sii $R2 \neq 0$
24		$swcl\ F3, R0, 12000$	Almacena acumulador en memoria

Código DLX32mf con Reordenación y Planificación de Trazas

Tiempo: 24 ciclos (tanto la vía IF como la vía ELSE)

Reducción ciclos $= (29-24)/29 = 17.2\%$

Ciclos	Etiqueta	Instrucción	Comentarios
		R1 \leftarrow 0	Prólogo: puntero de memoria
		R2 \leftarrow 1000	Prólogo: número de iteraciones
		F0 \leftarrow 100	Prólogo: constante utilizada en comparación
		F1 \leftarrow 0	Prólogo: constante utilizada para inicializar registros de punto flotante
		F2 \leftarrow 50	Prólogo: constante utilizada para inicializar registros de punto flotante
		F3 \leftarrow 0	Prólogo: acumulador
1	INICIO:	lwcl F4, R1, 0	F4 \leftarrow A[i], i \equiv R4
2		lwcl F5, R1, 4000	F5 \leftarrow B[i], i \equiv R4
3		lwcl F6, R1, 8000	F6 \leftarrow C[i], i \equiv R4
4		add.s F4, F4, F5	F4 \leftarrow A[i] + B[i]
5,6,7		3 x nop	3 ciclos de penalización por la dependencia de F4
8		add.s F4, F4, F6	F4 \leftarrow A[i] + B[i] + C[i]
9,10,11		3 x nop	3 ciclos de penalización por la dependencia de F4
12		c.gt.s F4, F0	FPcond = (F4 > 100) ? 1 : 0
13		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
14		bclf ELSE	Salto condicional sii "FPcond == 0" (sólo el 5% de las iteraciones)
15		1 x nop	Salto retardado
16	IF:	add.s F4, F1, F0	A[i] \leftarrow 100
17		1 x nop	1 ciclo de penalización por la dependencia de F4
18		addi R1, R1, 4	Aumenta el puntero de memoria de los tres vectores A, B y C
19		subi R2, R2, 1	Se reduce el número de iteraciones que restan por realizar
20	ACCUMULADOR:	add.s F3, F3, F4	acumulador += A[i]
21		swcl F4, R1, -4	Almacena A[i] en memoria
22		1 x nop	1 ciclo de penalización por la dependencia de F3
23		bne R2, R0, INICIO	Salto condicional al inicio de la iteración sii R2 != 0
24		swcl F3, R0, 12000	Salto retardado
		j FUERA-BUCLE	Salto incondicional cuando termina el bucle
		1 x nop	Salto retardado
16	ELSE:	add.s F4, F1, F2	Código Compensación: A[i] \leftarrow 50
17		addi R1, R1, 4	Código Compensación: Aumenta el puntero de memoria de los tres vectores A, B y C
18		j ACCUMULADOR	Código Compensación: Salto incondicional que evita el código de la vía ELSE
19		subi R2, R2, 1	Código Compensación: Se reduce el número de iteraciones que restan por realizar
	FUERA-BUCLE:		

Código DLX32mf con Reordenación y Desenrollamiento de Bucles (2 iteraciones, iteración 1 en negro, iteración 2 en rojo)Tiempo: $17.5 = 35/2$ ciclos/iteración (tanto las vías IF como las vías ELSE)Reducción ciclos = $(29 - 17.5)/29 = 40\%$

Ciclos	Etiqueta	Instrucción	Comentarios
		$R1 \leftarrow 0$	Prólogo: inicialización del puntero de memoria
		$R2 \leftarrow 1000$	Prólogo: inicialización del número de iteraciones
		$F0 \leftarrow 100$	Prólogo: inicialización de la constante utilizada en comparación
		$F1 \leftarrow 0$	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		$F2 \leftarrow 50$	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		$F3 \leftarrow 0$	Prólogo: acumulador
1	INICIO:	$lwcl\ F4, R1, 0$	$F4 \leftarrow A[i], i \equiv R4$
2		$lwcl\ F5, R1, 4000$	$F5 \leftarrow B[i], i \equiv R4$
3		$lwcl\ F6, R1, 8000$	$F6 \leftarrow C[i], i \equiv R4$
4		$add.s\ F4, F4, F5$	$F4 \leftarrow A[i] + B[i]$
5		$lwcl\ F7, R1, 4$	$F7 \leftarrow A[i+1], i+1 \equiv R4+4$
6		$lwcl\ F8, R1, 4004$	$F8 \leftarrow B[i+1], i+1 \equiv R4+4$
7		$lwcl\ F9, R1, 8004$	$F9 \leftarrow C[i+1], i+1 \equiv R4+4$
8		$add.s\ F7, F7, F8$	$F7 \leftarrow A[i+1] + B[i+1]$
9		$add.s\ F4, F4, F6$	$F4 \leftarrow A[i] + B[i] + C[i]$
10,11		$2 \times nop$	Por dependencia de F4
12		$add.s\ F7, F7, F9$	$F7 \leftarrow A[i+1] + B[i+1] + C[i+1]$
13		$c.gt.s\ F4, F0$	$FPcond = (F4 > 100) ? 1 : 0$
14		$1 \times nop$	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
15		$bclf\ ELSE1$	Salto condicional sii "FPcond = 0"
16		$1 \times nop$	Salto retardado
17	IF1:	$add.s\ F4, F1, F0$	$A[i] \leftarrow 100$
18		$addi\ R1, R1, 8$	Aumenta el puntero de memoria de los tres vectores A, B y C
19		$j\ ACUMULADOR$	Salto incondicional que evita el código de la vía ELSE1
20		$subi\ R2, R2, 2$	Se reduce el número de iteraciones que restan por realizar
17	ELSE1:	$add.s\ F4, F1, F2$	$A[i] \leftarrow 50$
18		$subi\ R2, R2, 2$	Se reduce el número de iteraciones que restan por realizar
19		$addi\ R1, R1, 8$	Aumenta el puntero de memoria de los tres vectores A, B y C
20		$1 \times nop$	1 ciclos de penalización por la dependencia de F4
21	ACUMULADOR1:	$add.s\ F3, F3, F4$	Acumulador pares += A[i]
22		$swcl\ F4, R1, -8$	Almacena A[i] en memoria
23		$c.gt.s\ F7, F0$	$FPcond = (F7 > 100) ? 1 : 0$
24		$1 \times nop$	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
25		$bclf\ ELSE2$	Salto condicional sii "FPcond = 0"
26		$1 \times nop$	Salto retardado
27	IF2:	$add.s\ F7, F1, F0$	$A[i+1] \leftarrow 100$
28		$1 \times nop$	Penalización por la dependencia de F7
29		$j\ ACUMULADOR2$	Salto incondicional que evita el código de la vía ELSE2
30		$1 \times nop$	Salto retardado
27	ELSE2:	$add.s\ F7, F1, F2$	$A[i+1] \leftarrow 50$
28,29,30		$3 \times nop$	3 ciclos de penalización por la dependencia de F7
31	ACUMULADOR2:	$add.s\ F3, F3, F7$	Acumulador += A[i+1]
32		$swcl\ F7, R1, -4$	Almacena A[i+1] en memoria
33		$1 \times nop$	3 ciclos de penalización por la dependencia de F3
34		$bne\ R2, R0, INICIO$	Salto condicional al inicio de la iteración sii $R2 \neq 0$
35		$swcl\ F3, R0, 12000$	Almacena acumulador en memoria; aprovecha el salto retardado

Código DLX32mf con Reordenación, Desenrollamiento de Bucles (2 iteraciones, iteración 1 en negro, iteración 2 en rojo) y Planificación de Trazas

Tiempo: $17.5 = 35/2$ ciclos/iteración (en los casos más frecuentes)

Reducción ciclos = $(29 - 17.5)/29 = 40\%$

Ciclos	Etiqueta	Instrucción	Comentarios
		R1 ← 0	Prólogo: inicialización del puntero de memoria
		R2 ← 1000	Prólogo: inicialización del número de iteraciones
		F0 ← 100	Prólogo: inicialización de la constante utilizada en comparación
		F1 ← 0	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		F2 ← 50	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		F3 ← 0	Prólogo: acumulador
1	INICIO:	lwcl F4, R1, 0	F4 ← A[i], i ≡ R4
2		lwcl F5, R1, 4000	F5 ← B[i], i ≡ R4
3		lwcl F6, R1, 8000	F6 ← C[i], i ≡ R4
4		add.s F4, F4, F5	F4 ← A[i] + B[i]
5		lwcl F7, R1, 4	F7 ← A[i+1], i+1 ≡ R4+4
6		lwcl F8, R1, 4004	F8 ← B[i+1], i+1 ≡ R4+4
7		lwcl F9, R1, 8004	F9 ← C[i+1], i+1 ≡ R4+4
8		add.s F7, F7, F8	F7 ← A[i+1] + B[i+1]
9		add.s F4, F4, F6	F4 ← A[i] + B[i] + C[i]
10,11		2 x nop	Por dependencia de F4
12		add.s F7, F7, F9	F7 ← A[i+1] + B[i+1] + C[i+1]
13		c.gt.s F4, F0	FPcond = (F4 > 100) ? 1 : 0
14		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
15		bclf ELSE1	Salto condicional sii "FPcond = 0"
16		1 x nop	Salto retardado
17	IF1:	add.s F4, F1, F0	A[i] ← 100
18		addi R1, R1, 8	Aumenta el puntero de memoria de los tres vectores A, B y C
19		subi R2, R2, 2	Se reduce el número de iteraciones que restan por realizar
20		1 x nop	1 ciclos de penalización por la dependencia de F4
21	ACUMULADOR1:	add.s F3, F3, F4	Acumulador += A[i] pares
22		swcl F4, R1, -8	Almacena A[i] en memoria
23		c.gt.s F7, F0	FPcond = (F7 > 100) ? 1 : 0
24		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
25		bclf ELSE2	Salto condicional sii "FPcond = 0"
26		1 x nop	Salto retardado
27	IF2:	add.s F7, F1, F0	A[i+1] ← 100
28,29,30		3 x nop	Penalización por la dependencia de F7
31	ACUMULADOR2:	add.s F3, F3, F7	Acumulador += A[i+1] impares
32		swcl F7, R1, -4	Almacena A[i+1] en memoria
33		1 x nop	3 ciclos de penalización por la dependencia de F3
34		bne R2, R0, INICIO	Salto condicional al inicio de la iteración sii R2 != 0
35		swcl F3, R0, 12000	Almacena acumulador en memoria; aprovecha el salto retardado
		j FIN	
	ELSE1:	add.s F4, F1, F2	A[i] ← 50
		subi R2, R2, 2	Se reduce el número de iteraciones que restan por realizar
		j ACUMULADOR1	Salto incondicional que evita el código de la vía ELSE1
		addi R1, R1, 8	Salto retardado + aumenta el puntero de memoria de los tres vectores A, B y C
	ELSE2:	add.s F7, F1, F2	A[i+1] ← 50
		1 x nop	1 ciclos de penalización por la dependencia de F7
		j ACUMULADOR2	Salto incondicional que evita el código de la vía ELSE2
		1 x nop	Salto retardado + 1 ciclos de penalización por la dependencia de F7
	FIN:		Goto FIN

Código DLX32vliw con Reordenación y Desenrollamiento de Bucles (2 iteraciones, iteración 1 en negro, iteración 2 en rojo)

Tiempo: $14.5 = 29/2$ ciclos/iteración (tanto las vías IF como las vías ELSE)

Reducción ciclos = $(29 - 14.5)/29 = 50\%$

							Comentarios
		R1 ← 0					Prólogo: puntero de memoria
		R2 ← 1000					Prólogo: número de iteraciones
		F0 ← 100					Prólogo: constante
		F1 ← 0					Prólogo: constante
		F2 ← 50					Prólogo: constante
		F3 ← 0					Prólogo: acumulador
Ciclos	Etiqueta	Instrucción MEM-1	Instrucción MEM-2	Instrucción FP-1	Instrucción FP-2	Instrucción INT	
1	INICIO:	lwcl F4, R1, 0	lwcl F5, R1, 4000				
2		lwcl F8, R1, 4004	lwcl F7, R1, 4				
3		lwcl F6, R1, 8000	lwcl F9, R1, 8004	add.s F4, F4, F5		subi R2, R2, 2	
4				add.s F7, F7, F8		addi R1, R1, 8	
5,6		2 x nop					
7				add.s F4, F4, F6			
8				add.s F7, F7, F9			
9,10		2 x nop					
11						c.gt.s F4, F0	
12		1 x nop					
13						bclf ELSE1	
14		1 x nop				c.gt.s F7, F0	
15	IF1:			add.s F4, F1, F0			
16		1 x nop					
17						j ACUMULADOR1	
18		1 x nop					
15	ELSE1:			add.s F4, F1, F2			
16,17,18		3 x nop					
19	ACUMULADOR1:	swcl F4, R1, -8		add.s F3, F3, F4		bclf ELSE2	
20		1 x nop					Salto retardado
21	IF2:			add.s F7, F1, F0			
22		1 x nop					
23						j ACUMULADOR2	
24		1 x nop					
21	ELSE2:			add.s F7, F1, F2			
22,23,24		3 x nop					
25	ACUMULADOR2:	swcl F7, R1, -4		add.s F3, F3, F7			
26,27		2 x nop					
28						bne R2, R0, INICIO	
29		swcl F3, R0, 12000					

Código DLX32vliw con Reordenación y Desenrollamiento de Bucles (2 iteraciones, iteración 1 en negro, iteración 2 en rojo) y Planificación de Trazas

Tiempo: $14.5 = 29/2$ ciclos/iteración (tanto las vías IF como las vías ELSE)

Reducción ciclos = $(29 - 14.5)/29 = 50\%$

							Comentarios
		R1 ← 0					Prólogo: puntero de memoria
		R2 ← 1000					Prólogo: número de iteraciones
		F0 ← 100					Prólogo: constante
		F1 ← 0					Prólogo: constante
		F2 ← 50					Prólogo: constante
		F3 ← 0					Prólogo: acumulador
Ciclos	Etiqueta	Instrucción MEM-1	Instrucción MEM-2	Instrucción FP-1	Instrucción FP-2	Instrucción INT	
1	INICIO:	lwcl F4, R1, 0	lwcl F5, R1, 4000				
2		lwcl F8, R1, 4004	lwcl F7, R1, 4				
3		lwcl F6, R1, 8000	lwcl F9, R1, 8004	add.s F4, F4, F5		subi R2, R2, 2	
4				add.s F7, F7, F8		addi R1, R1, 8	
5,6		2 x nop					
7				add.s F4, F4, F6			
8				add.s F7, F7, F9			
9,10		2 x nop					
11						c.gt.s F4, F0	
12		1 x nop					
13						bclf ELSE1	
14		1 x nop				c.gt.s F7, F0	
15	IF1:			add.s F4, F1, F0			
16,17,18		3 x nop					Elimina j ACUMULADOR1
19	ACUMULADOR1:	swcl F4, R1, -8		add.s F3, F3, F4		bclf ELSE2	
20		1 x nop					
21	IF2:			add.s F7, F1, F0			
22,23,24		3 x nop					Elimina j ACUMULADOR2
25	ACUMULADOR2:	swcl F7, R1, -4		add.s F3, F3, F7			
26,27		2 x nop					
28						bne R2, R0, INICIO	
29		swcl F3, R0, 12000					
						j FUERA-BUCLE	
						1 x nop	
	ELSE1:			add.s F4, F1, F2			
		1 x nop					
						j ACUMULADOR1	
		1 x nop					
	ELSE2:			add.s F7, F1, F2			
		1 x nop					
						j ACUMULADOR2	
		1 x nop					
	FUERA-BUCLE:						

Código DLX32vliw con Reordenación y Desenrollamiento de Bucles (4 iteraciones, iteración 1 en negro, iteración 2 en rojo, iteración 3 en azul, iteración 4 en verde) y Planificación de Trazas

Tiempo: $10.25 = 41/4$ ciclos/iteración (tanto las vías IF como las vías ELSE)

Reducción ciclos = $(29 - 10.25)/29 = 64.7\%$

		R1 ← 0	R2 ← 1000	F0 ← 100	F1 ← 0	F2 ← 50	F3 ← 0
Ciclos	Etiqueta	Instrucción MEM-1	Instrucción MEM-2	Instrucción FP-1	Instrucción FP-2	Instrucción INT	Comentarios
1	INICIO:	lwcl F4, R1, 0	lwcl F5, R1, 4000				
2		lwcl F8, R1, 4004	lwcl F7, R1, 4				
3		lwcl F6, R1, 8000	lwcl F9, R1, 8004	add.s F4, F4, F5		subi R2, R2, 4	
4		lwcl F11, R1, 4008	lwcl F10, R1, 8	add.s F7, F7, F8			
5		lwcl F12, R1, 8008	lwcl F13, R1, 12				
6		lwcl F14, R1, 4012	lwcl F15, R1, 8012	add.s F10, F10, F11		addi R1, R1, 16	
7				add.s F4, F4, F6			
8				add.s F7, F7, F9			
9		1 x nop					
10				add.s F10, F10, F12	add.s F13, F13, F14		
11						c.gt.s F4, F0	
12		1 x nop					
13						bclf ELSE1	
14					add.s F13, F13, F15	c.gt.s F7, F0	
15	IF1:			add.s F4, F1, F0			
16,17,18		3 x nop					
19	ACUMULADOR1:	swcl F4, R1, -8		add.s F3, F3, F4		bclf ELSE2	
20		1 x nop					
21	IF2:			add.s F7, F1, F0		c.gt.s F10, F0	Se replica en ELSE2
22,23,24		3 x nop					
25	ACUMULADOR2:	swcl F7, R1, -4		add.s F3, F3, F7		bclf ELSE3	
26		1 x nop					
27	IF3:			add.s F10, F1, F0		c.gt.s F13, F0	Se replica en ELSE3
28,29,30		3 x nop					
31	ACUMULADOR3:	swcl F10, R1, -12		add.s F3, F3, F10		bclf ELSE4	
32		1 x nop					
33	IF4:			add.s F13, F1, F0			
34,35,36		3 x nop					
37	ACUMULADOR4:	swcl F13, R1, -12		add.s F3, F3, F13			
38,39		2 x nop					
40						bne R2, R0, INICIO	
41		swcl F3, R0, 12000					
						j FUERA-BUCLE	
						1 x nop	
	ELSE1:			add.s F4, F1, F2			
		1 x nop					
						j ACUMULADOR1	
		1 x nop					
	ELSE2:			add.s F7, F1, F2		c.gt.s F10, F0	Se replica en IF2
		1 x nop					
						j ACUMULADOR2	
		1 x nop					

ELSE3:			add.s F10, F1, F2		c.gt.s F13, F0	Se replica en IF3
	l x nop					
					j ACUMULADOR3	
	l x nop					
ELSE3:			add.s F13, F1, F2			
	l x nop					
					j ACUMULADOR4	
	l x nop					
FUERA-BUCLE:						

PREGUNTA 2-1-6 (Examen de 14 septiembre 2010). Aplica la técnica de desenrollamiento de bucles y la de planificación de trazas utilizando la arquitectura *DLX32vliw* a la generación de código ensamblador de la siguiente subrutina. Suponer que la vía “if” de la sentencia “if-then-else” es la que se ejecuta en el 95% de las iteraciones. Analiza de forma cuantitativa las prestaciones del procesador *DLX32vliw* con respecto al procesador *DLX32mf*.

```
subrutina(float A[1000], float B[1000], float C[1000]){
    int i;
    int contador = 0;
    float acumulador = 0;
    for (i=0; i<1000; i++){
        A[i] += (B[i] + C[i]);
        if (A[i] > 100) { contador++; }
        acumulador += A[i];
    }
}
```

Para la generación de código, se puede utilizar 32 registros de enteros R_x ($x=0,\dots,31$), 32 registros de punto flotante F_x ($x=0,\dots,31$), todos ellos de 32 bits, el registro de estado *FPcond*, así como las siguientes instrucciones:

Nemónico	Nombre	Descripción RTL	Latencia en ciclos por dependencia de ...		Tipo de instrucción	Observaciones
			Datos	Control		
lwcl F_d, R_s, inm	Carga dato punto flotante 32 bits	$F_d \leftarrow M[\text{inm} + (R_s)]$	1	0	Enteros(ENT), Registro-Inmediato	
swcl F_d, R_s, inm	Almacenar dato punto flotante 32 bits	$F_d \rightarrow M[\text{inm} + (R_s)]$	0	0	Enteros(ENT), Registro-Inmediato	
add.s F_d, F_s, F_t	Suma datos punto flotante 32 bits	$F_d \leftarrow F_s + F_t$	3	0	Flotante (FP), Registro – Registro	
adi R_t, R_s, inm	Suma con inmediato	$R_t \leftarrow R_s + \text{inm}$	0	0	Enteros(ENT), Registro - Registro	
subi R_t, R_s, inm	Resta con inmediato	$R_t \leftarrow R_s + \text{inm}$	0	0	Enteros(ENT), Registro – Registro	
c.lt.s F_s, F_t	Comparación menor-que de registros flotantes de 32 bits con inicialización del registro estado <i>FPcond</i>	$\text{FPcond} = (F_s < F_t) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <i>FPcond</i> se inicializa en la etapa EX
c.gt.s F_s, F_t	Comparación mayor-que de registros flotantes de 32 bits con inicialización del registro estado <i>FPcond</i>	$\text{FPcond} = (F_s > F_t) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <i>FPcond</i> se inicializa en la etapa EX
j inm	Salto incondicional	$\text{PC} \leftarrow \text{inm}$	0	1	Enteros (ENT), Saltos	Tiene un ciclo de salto retardado
bne R_s, R_t, inm	Salto condicional	If ($R_s \neq R_t$), $\text{PC} \leftarrow \text{PC} + 4 + \text{inm}$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
beq R_s, R_t, inm	Salto condicional	If ($R_s = R_t$), $\text{PC} \leftarrow \text{PC} + 4 + \text{inm}$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
bclf inm	Salto condicional utilizando registro de estado <i>FPcond</i>	If (! <i>FPcond</i>), $\text{PC} \leftarrow \text{PC} + 4 + \text{inm}$	1	1	Enteros (ENT), Inmediato	El estado del registro <i>FPcond</i> se consulta en la etapa ID
bclt inm	Salto condicional utilizando registro de estado <i>FPcond</i>	If (<i>FPcond</i>), $\text{PC} \leftarrow \text{PC} + 4 + \text{inm}$	1	1	Enteros (ENT), Inmediato	El estado del registro <i>FPcond</i> se consulta en la etapa ID
nop	No-Operación	No modifica el estado de registros	0	0		Equivale a un ciclo de penalización

Ayudas.

- Suponer que los registros involucrados en el cómputo del bucle `for` que requieren una inicialización previa están inicializados utilizando las instrucciones apropiadas, las cuales no tienen que coincidir con las que aparecen en la tabla. Indicar qué registros se inicializan y cuáles son los valores iniciales.
- Utilizar una tabla como la que aparece a continuación para codificar las instrucciones *DLX32mf*

Ciclos	Etiqueta	Instrucción
1		

- Utilizar una tabla como la que aparece a continuación para codificar las instrucciones *DLX32vliw*

Ciclos	Etiqueta	Instrucción MEM-1	Instrucción MEM-2	Instrucción FP-1	Instrucción FP-2	Instrucción INT
1						

PREGUNTA 2-1-7 (Examen de 9 diciembre 2010). Aplica la técnica de desenrollamiento de bucles y la de planificación de trazas utilizando la arquitectura *DLX32vliw* a la generación de código ensamblador de la siguiente subrutina. Suponer que la vía “if” de la sentencia “if-then-else” es la que se ejecuta en el 95% de las iteraciones. Analiza de forma cuantitativa las prestaciones del procesador *DLX32vliw* con respecto al procesador *DLX32mf*.

```
subrutina(float A[1000], float B[1000]){
    int i;
    int contador = 0;
    float acumulador = 0;
    for (i=0; i<1000; i++){
        A[i] += B[i];
        if (A[i] > 100) { contador++; }
        else { contador--; }
        acumulador += A[i];
    }
}
```

Para la generación de código, se puede utilizar 32 registros de enteros R_x ($x=0,\dots,31$), 32 registros de punto flotante F_x ($x=0,\dots,31$), todos ellos de 32 bits, el registro de estado *FPcond*, así como las siguientes instrucciones:

Nemónico	Nombre	Descripción RTL	Latencia en ciclos por dependencia de ...		Tipo de instrucción	Observaciones
			Datos	Control		
lwc1 Fd, Rs, inm	Carga dato punto flotante 32 bits	$Fd \leftarrow M[inm + (RS)]$	1	0	Enteros(ENT), Registro-Inmediato	
swc1 Fd, Rs, inm	Almacenar dato punto flotante 32 bits	$Fd \rightarrow M[inm + (RS)]$	0	0	Enteros(ENT), Registro-Inmediato	
add.s Fd, Fs, Ft	Suma datos punto flotante 32 bits	$Fd \leftarrow Fs + Ft$	3	0	Flotante (FP), Registro – Registro	
addi Rt, Rs, inm	Suma con inmediato	$Rt \leftarrow Rs + inm$	0	0	Enteros(ENT), Registro - Registro	
subi Rt, Rs, inm	Resta con inmediato	$Rt \leftarrow Rs - inm$	0	0	Enteros(ENT), Registro – Registro	
c.lt.s Fs, Ft	Comparación menor-que de registros flotantes de 32 bits con inicialización del registro estado <i>FPcond</i>	$FPcond = (Fs < Ft) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <i>FPcond</i> se inicializa en la etapa EX
c.gt.s Fs, Ft	Comparación mayor-que de registros flotantes de 32 bits con inicialización del registro estado <i>FPcond</i>	$FPcond = (Fs > Ft) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <i>FPcond</i> se inicializa en la etapa EX
j inm	Salto incondicional	$PC \leftarrow inm$	0	1	Enteros (ENT), Saltos	Tiene un ciclo de salto retardado
bne Rs, Rt, inm	Salto condicional	If ($Rs \neq Rt$), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
beq Rs, Rt, inm	Salto condicional	If ($Rs = Rt$), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
bc1f inm	Salto condicional utilizando registro de estado <i>FPcond</i>	If ($!FPcond$), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Inmediato	El estado del registro <i>FPcond</i> se consulta en la etapa ID
bc1t inm	Salto condicional utilizando registro de estado <i>FPcond</i>	If ($FPcond$), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Inmediato	El estado del registro <i>FPcond</i> se consulta en la etapa ID
nop	No-Operación	No modifica el estado de registros	0	0		Equivale a un ciclo de penalización

Ayudas.

- Suponer que los registros involucrados en el cómputo del bucle `for` que requieren una inicialización previa están inicializados utilizando las instrucciones apropiadas, las cuales no tienen que coincidir con las que aparecen en la tabla. Indicar qué registros se inicializan y cuáles son los valores iniciales.
- Utilizar una tabla como la que aparece a continuación para codificar las instrucciones *DLX32mf*

Ciclos	Etiqueta	Instrucción
1		

- Utilizar una tabla como la que aparece a continuación para codificar las instrucciones *DLX32vliw*

Ciclos	Etiqueta	Instrucción MEM-1	Instrucción MEM-2	Instrucción FP-1	Instrucción FP-2	Instrucción INT
1						

PREGUNTA 2-1-8 (Examen de Julio 2011). Aplica la técnica de desenrollamiento de bucles y la de planificación de trazas utilizando la arquitectura *DLX32vliw* a la generación de código ensamblador de la subrutina que se muestra más abajo. Suponer que la vía “if” de la primera sentencia “if-then-else” es la que se ejecuta en el 95% de las iteraciones, y que la vía “else” de la segunda sentencia “if-then-else” es la que también se ejecuta en el 95% de las iteraciones. Analiza de forma cuantitativa las prestaciones del procesador *DLX32vliw* con respecto al procesador *DLX32mf*.

```
subrutina(float A[1000], float B[1000]){
    int i;
    int contador = 0;
    float acumulador = 0;

    for (i=0; i<1000; i++){
        A[i] += B[i];

        if (A[i] > 100) { contador++; }
        else if ((A[i] >= 50) && (A[i] <= 100)) { contador--; }
        else { contador += 2; }

        if (B[i] > 100) { contador += 3; }
        else { contador--; }

        acumulador += A[i];
    }
}
```

Para la generación de código, se puede utilizar 32 registros de enteros R_x ($x=0,\dots,31$), 32 registros de punto flotante F_x ($x=0,\dots,31$), todos ellos de 32 bits, el registro de estado *FPcond*, así como las siguientes instrucciones:

Nemónico	Nombre	Descripción RTL	Latencia en ciclos por dependencia de ...		Tipo de instrucción	Observaciones
			Datos	Control		
lwcl Fd, Rs, inm	Carga dato punto flotante 32 bits	$Fd \leftarrow M[inm + (Rs)]$	1	0	Enteros(ENT), Registro-Inmediato	
swcl Fd, Rs, inm	Almacenar dato punto flotante 32 bits	$Fd \rightarrow M[inm + (Rs)]$	0	0	Enteros(ENT), Registro-Inmediato	
add.s Fd, Fs, Ft	Suma datos punto flotante 32 bits	$Fd \leftarrow Fs + Ft$	3	0	Flotante (FP), Registro - Registro	
addi Rt, Rs, inm	Suma con inmediato	$Rt \leftarrow Rs + inm$	0	0	Enteros(ENT), Registro - Registro	
subi Rt, Rs, inm	Resta con inmediato	$Rt \leftarrow Rs - inm$	0	0	Enteros(ENT), Registro - Registro	
c.lt.s Fs, Ft	Comparación menor-que de registros flotantes de 32 bits con inicialización del registro estado <i>FPcond</i>	$FPcond = (Fs < Ft) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <i>FPcond</i> se inicializa en la etapa EX
c.gt.s Fs, Ft	Comparación mayor-que de registros flotantes de 32 bits con inicialización del registro estado <i>FPcond</i>	$FPcond = (Fs > Ft) ? 1 : 0$	0	0	Flotante (FP), Registro - Registro	El registro de estado <i>FPcond</i> se inicializa en la etapa EX
j inm	Salto incondicional	$PC \leftarrow inm$	0	1	Enteros (ENT), Saltos	Tiene un ciclo de salto retardado
bne Rs, Rt, inm	Salto condicional	If ($Rs \neq Rt$), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
beq Rs, Rt, inm	Salto condicional	If ($Rs == Rt$), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Saltos	El salto se resuelve en la etapa ID
bclf inm	Salto condicional utilizando registro de estado <i>FPcond</i>	If (! <i>FPcond</i>), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Inmediato	El estado del registro <i>FPcond</i> se consulta en la etapa ID
bclt inm	Salto condicional utilizando registro de estado <i>FPcond</i>	If (<i>FPcond</i>), $PC \leftarrow PC + 4 + inm$	1	1	Enteros (ENT), Inmediato	El estado del registro <i>FPcond</i> se consulta en la etapa ID
nop	No-Operación	No modifica el estado de registros	0	0		Equivale a un ciclo de penalización

Ayudas.

- Suponer que los registros involucrados en el cómputo del bucle `for` que requieren una inicialización previa están inicializados utilizando las instrucciones apropiadas, las cuales no tienen que coincidir con las que aparecen en la tabla. Indicar qué registros se inicializan y cuáles son los valores iniciales.
- Utilizar una tabla como la que aparece a continuación para codificar las instrucciones *DLX32mf*

Ciclos	Etiqueta	Instrucción
1		

- Utilizar una tabla como la que aparece a continuación para codificar las instrucciones *DLX32vliw*

Ciclos	Etiqueta	Instrucción MEM-1	Instrucción MEM-2	Instrucción FP-1	Instrucción FP-2	Instrucción INT
1						

SOLUCIÓN PREGUNTA 2-1-8Código DLX32mf sin optimizar

Tiempo: 28 ciclos (ruta más frecuente: if + else)

Ciclos	Etiqueta	Instrucción	Comentarios
		R1 ← 0	Prólogo: inicialización del puntero de memoria
		R2 ← 1000	Prólogo: inicialización del número de iteraciones
		F0 ← 100	Prólogo: inicialización de la constante utilizada en comparación
		F1 ← 0	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		F2 ← 50	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		F3 ← 0	Prólogo: variable "acumulador"
		R3 ← 0	Prólogo: variable "contador"
1	INICIO:	lwcl F4, R1, 0	F4 ← A[i], i ≡ R4
2		lwcl F5, R1, 4000	F5 ← B[i], i ≡ R4
3		add.s F4, F4, F5	F4 ← A[i] + B[i]; A[i] += B[i]
4,5,6		3 x nop	3 ciclos de penalización por la dependencia de F4
7	IF:	c.gt.s F4, F0	FPcond = (F4 > 100) ? 1 : 0; if (A[i] > 100)
8		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
9		bclf ELSEIF	Salto condicional sii "FPcond == 0"
10		1 x nop	Salto retardado
11		addi R3, R3, 1	contador++
12		j IF2	Salto incondicional que evita el código de las vías ELSEIF y ELSE
13		1 x nop	Salto retardado
11	ELSEIF:	c.gt.s F4, F2	FPcond = (F4 > 50) ? 1 : 0
12		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
13		bclf ELSE	Salto condicional sii "FPcond == 0" que evita la parte ELSEIF
14		1 x nop	Salto retardado
15		subi R3, R3, 1	contador--
16		j IF2	Salto incondicional que evita el código de la vía ELSE
17		1 x nop	Salto retardado
15	ELSE:	addi R3, R3, 2	contador += 2
14 (IF), 18 (ELSEIF), 16 (ELSE)	IF2:	c.gt.s F5, F0	FPcond = (F5 > 100) ? 1 : 0; if (B[i] > 100)
15 (IF)		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
16 (IF)		bclf ELSE2	Salto condicional sii "FPcond == 0"
17 (IF)		1 x nop	Salto retardado
		addi R3, R3, 3	contador += 3
		j ACUMULADOR	Salto incondicional que evita el código de la vía ELSE2
		1 x nop	Salto retardado
18 (IF+ELSE2)	ELSE2:	subi R3, R3, 1	contador--
19 (IF+ELSE2)	ACUMULADOR:	add.s F3, F3, F4	acumulador += A[i]
20 (IF+ELSE2)		swcl F4, R1, 0	Almacena A[i] en memoria
21,22 (IF+ELSE2)		2 x nop	2 ciclos de penalización por la dependencia de F3
23 (IF+ELSE2)		swcl F3, R0, 12000	Almacena acumulador en memoria
24 (IF+ELSE2)		addi R1, R1, 4	Aumenta el puntero de memoria de los dos vectores A, B
25 (IF+ELSE2)		subi R2, R2, 1	Se reduce el número de iteraciones que restan por realizar
26 (IF+ELSE2)		1 x nop	Penalización porque subi se ejecuta en etapa EX y bne se resuelve en etapa ID
27 (IF+ELSE2)		bne R2, R0, INICIO	Salto condicional al inicio de la iteración sii R2 != 0
28 (IF+ELSE2)		1 x nop	Salto retardado

Código DLX32mf con Planificación de Trazas

Tiempo: 26 ciclos (ruta más frecuente: if + else), Speed-Up = $2 / 28 = 7\%$

Ciclos	Etiqueta	Instrucción	Comentarios
		R1 ← 0	Prólogo: inicialización del puntero de memoria
		R2 ← 1000	Prólogo: inicialización del número de iteraciones
		F0 ← 100	Prólogo: inicialización de la constante utilizada en comparación
		F1 ← 0	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		F2 ← 50	Prólogo: inicialización de la constante utilizada para inicializar registros de punto flotante
		F3 ← 0	Prólogo: variable "acumulador"
		R3 ← 0	Prólogo: variable "contador"
1	INICIO:	lwcl F4, R1, 0	F4 ← A[i], i ≡ R4
2		lwcl F5, R1, 4000	F5 ← B[i], i ≡ R4
3		add.s F4, F4, F5	F4 ← A[i] + B[i]; A[i] += B[i]
4,5,6		3 x nop	3 ciclos de penalización por la dependencia de F4
7	IF:	c.gt.s F4, F0	FPcond = (F4 > 100) ? 1 : 0; if (A[i] > 100)
8		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
9		bclf ELSEIF	Salto condicional sii "FPcond == 0"
10		1 x nop	Salto retardado
11		addi R3, R3, 1	contador++
12	IF2:	c.gt.s F5, F0	FPcond = (F5 > 100) ? 1 : 0; if (B[i] > 100)
13		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
14		bclt IF2compens	Salto condicional sii "FPcond == 1", ruta menos frecuente
15		1 x nop	Salto retardado
16	ELSE2:	subi R3, R3, 1	contador--
17	ACUMULADOR:	add.s F3, F3, F4	acumulador += A[i]
18		swcl F4, R1, 0	Almacena A[i] en memoria
19, 20		2 x nop	2 ciclos de penalización por la dependencia de F3
21		swcl F3, R0, 12000	Almacena acumulador en memoria
22		addi R1, R1, 4	Aumenta el puntero de memoria de los dos vectores A, B
23		subi R2, R2, 1	Se reduce el número de iteraciones que restan por realizar
24		1 x nop	Penalización porque subi se ejecuta en etapa EX y bne se resuelve en etapa ID
25		bne R2, R0, INICIO	Salto condicional al inicio de la iteración sii R2 != 0
26		1 x nop	Salto retardado
		J FIN	
	ELSEIF:	c.gt.s F4, F2	FPcond = (F4 > 50) ? 1 : 0
		1 x nop	Penalización porque c.gt.s se ejecuta en etapa EX y bclf se resuelve en etapa ID
		bclf ELSE	Salto condicional sii "FPcond == 0"
		1 x nop	Salto retardado
		subi R3, R3, 1	contador--
		j IF2	Salto incondicional que vuelve al bucle para evaluar el segundo IF-THEN-ELSE
		1 x nop	Salto retardado
	ELSE:	addi R3, R3, 2	contador += 2
		j IF2	Salto incondicional para evaluar el segundo IF-THEN-ELSE
		1 x nop	Salto retardado
	IF2compens:	addi R3, R3, 3	contador += 3
		j ACUMULADOR	Salto incondicional que vuelve al bucle para evaluar el acumulador
		1 x nop	Salto retardado
	FIN:		

TEMA 2-2: Planificación Dinámica de Instrucciones. Ejecución Fuera de Orden

Problema 2-2-1 (HP96, ex.4.6, pp.363). Considerar la siguiente secuencia de código:

```

MULTD      F0 , F6 , F4
SUBD       F8 , F0 , F2
ADDD       F2 , F10 , F2

```

La instrucción SUBD depende de MULTD (dependencia RAW o verdadera), y por tanto la instrucción MULTD debe terminar antes de SUBD. Existe una dependencia WAR entre ADDD y SUBD, y por tanto, ADDD no debe terminar de ejecutarse antes de que SUBD haya leído sus operandos. Describir cómo la técnica de Tomasulo evita este problema en el procesador DLX32000 y mostrar los respectivos valores de los registros físicos para el código anterior, suponiendo que ADDD ha terminado de ejecutarse y no ha almacenado el resultado. Suponer que la unidad funcional de multiplicaciones tiene 3 etapas de segmentación, y la unidad funcional de suma/resta tiene 1 etapa de segmentación.

Solución.

La forma que el algoritmo de Tomasulo propone para solventar el problema de las dependencias verdaderas (RAW) es bloqueando a la instrucción que depende (SUBD en este caso) en la estación de reserva hasta que la instrucción de la que depende (MULTD en el ejercicio) ha terminado de actualizar el estado del procesador. En la estación de reserva correspondiente a la resta se apuntará en el campo Qj cuál es la estación de reserva que contiene la instrucción que le proporciona el dato que le falta. Observemos en la siguiente tabla los ciclos en los que una instrucción se encuentra en las distintas etapas.

Instrucción	Etapa		
	IS	EX	WB
MULTD F0 , F6 , F4	1	2-4	5
SUBD F8 , F0 , F2	2	6	7
ADDD F2 , F10 , F2	3	4-5	6

Vemos cómo la instrucción de resta se envía a ejecutar en el ciclo 2 (IS) y durante 3 ciclos (ciclos 3-5) se queda parada en la estación de reserva ADD1/SUB1 esperando el resultado de la instrucción MULTD de la cual tiene una dependencia verdadera a través de F0. Por eso, no se envía a ejecutar a su unidad funcional aritmética hasta el ciclo 6, un ciclo después que la multiplicación haya escrito el resultado en el banco de registros y el bus de datos común mientras se encuentra en la etapa WB en el ciclo 5.

También se puede observar cómo la suma ADDD está durante 2 ciclos en la etapa de ejecución (EX, ciclos 4-5). ¿Por qué ocurre esto si el número de tapas de la unidad funcional de sumas y restas en punto flotante es de 1 ciclo? La respuesta es que habría una colisión entre la primera (MULTD) y tercera instrucción (ADDD) cuando se transmiten los resultados por el bus de datos común. En este caso se prioriza la instrucción más antigua que es la multiplicación.

Adicionalmente, se puede observar claramente la ejecución y terminación/retirado fuera de orden de instrucciones con el algoritmo de Tomasulo, ya que la resta (más antigua) termina/retira un ciclo después que lo hace la suma (más nueva).

Observar a continuación el estado de las estaciones de reserva durante los ciclos 4º y 5º, durante los cuales la suma se encuentra ejecutándose en EX, pero aún no ha escrito el resultado en el banco de registros.

Ciclo 4: estaciones de reserva

Tiempo restante	<i>ER 4º ciclo</i>	Ocupado	Instrucción	Vj	Vk	Qj	Qk
0	<i>MUL1/DIV1</i>	Sí	MULTD	F6	F4		
	<i>ADD1/SUB1</i>	Sí	SUBD		F2	MUL1/DIV1	
0	<i>ADD2/SUB2</i>	Sí	ADDD	F10	F2		

En el ciclo 4 la estación de reserva de la resta ADD1/SUB1 indica que la instrucción de multiplicación (MULTD) que se encuentra en la estación de reserva MUL1/DIV1 es la que produce el resultado que debe disponer la instrucción SUBD para enviarse a ejecutar en la unidad funcional.

El registro de estado del banco de registros tiene el siguiente contenido, indicando las estaciones de reserva de las instrucciones que están en vuelo. En estas estaciones de reserva se encuentran las instrucciones que producen el resultado a actualizar.

Ciclo 4: registro de estado del banco de registros

F0	F2	F4	F6	F8
MULT1/DIV1	ADD2/SUBD2			ADD1/SUBD1

Ciclo 5: estaciones de reserva

Tiempo restante	<i>ER 5º ciclo</i>	Ocupado	Instrucción	Vj	Vk	Qj	Qk
	<i>MUL1/DIV1</i>	No	MULTD	F6	F4		
1	<i>ADD1/SUB1</i>	Sí	SUBD	MULTD	F2		
0	<i>ADD2/SUB2</i>	Sí	ADDD	F10	F2		

Ciclo 5: registro de estado del banco de registros

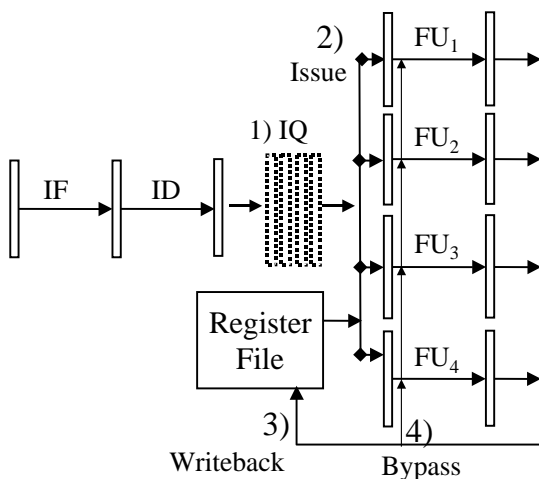
F0	F2	F4	F6	F8
	ADD2/SUBD2			ADD1/SUBD1

Como conclusión, se ha podido observar la forma en la que se evitan las dependencias verdaderas (RAW) con el algoritmo de Tomasulo. La técnica consiste en bloquear en una estación de reserva la instrucción dependiente hasta que esta estación de reserva observa por el bus de datos común el resultado de la operación que espera. Este resultado es copiado en la estación de reserva donde se encuentra la instrucción dependiente.

La técnica para evitar las dependencias WAR que se utiliza consiste en copiar en la estación de reserva de la instrucción más antigua el valor de los registros disponibles antes que otras instrucciones posteriores los sobrescriban, aunque la instrucción antigua todavía no pueda ser ejecutada en su unidad funcional.

En la siguiente tabla se puede observar la evolución temporal de las instrucciones ciclo a ciclo.

		<i>Ciclo</i>						
		1	2	3	4	5	6	7
MULTD	F0, F6, F4	IS	EX	EX	EX	WB		
SUBD	F8, F0, F2		IS	EX!	EX!	EX!	EX	WB
ADDD	F2, F10, F2			IS	EX	EX!	WB	

Problema 2-2-3. Unidades Funcionales Múltiples: Ejecución Fuera de Orden

Se pretende analizar el procesador de la figura anexa para que las instrucciones se ejecuten fuera de orden. El procesador dispone de 5 etapas: IF, ID, Issue-IQ, EX, WB. Suponemos que existen 4 unidades funcionales FU1=ALU, FU2=FAD, FU3=FMUL, FU4=Mem. Para especificar de forma completa el funcionamiento del procesador es necesario determinar algunos parámetros, que se muestran en la figura y se describen a continuación:

1) Número de instrucciones que “esperan” temporalmente a ser ejecutadas, el cual coincide con el tamaño de la Cola de Instrucciones IQ. Si la cola IQ se llena, la etapa ID se bloquea (y por tanto, la etapa IF también). En el ejemplo de este problema supondremos

que la cola IQ tiene tamaño 2 (sólo pueden alojarse hasta 2 instrucciones).

2) Para poder lanzar a ejecutar (etapa *Issue*) a la siguiente etapa EX una instrucción que está en la Cola de Instrucciones IQ es necesario que *no haya dependencias RAW pendientes*; es decir, que los operandos de la instrucción estén disponibles en el momento de la ejecución, y que *todos los recursos hardware necesarios estén disponibles*. La lectura de los operandos fuente desde el banco de registros (Register File) se realiza en esta etapa Issue-IQ. Restricciones en los recursos pueden limitar el número de instrucciones que se pueden lanzar a ejecutar simultáneamente a EX. En este caso, supondremos que *sólo se pueden lanzar a ejecutar hasta dos instrucciones por ciclo*.

3) El número de Buses de Postescritura (*Writeback*) determina cuántas unidades funcionales pueden escribir su resultado simultáneamente al banco de registros (supondremos que hay un único bus de postescritura).

4) La existencia o no de caminos de anticipación (*Bypass*) determina si es posible enviar los valores que están en los Buses de Writeback directamente a las unidades funcionales o si, por el contrario, hay que esperar a que los resultados se escriban en el banco de registros para poder usarlos como operandos. Nosotros asumiremos que sí hay caminos de realimentación.

Tenemos 4 Unidades Funcionales segmentadas: ALU, Mem, FAD y FML. En este caso, puede haber más de una instrucción en ejecución en la misma unidad funcional; eso sí, en diferentes etapas de la ejecución. La unidad funcional que se utiliza para cada tipo de instrucción y su latencia efectiva son las siguientes:

Instrucción	Semántica	Unidad Funcional	Latencia de la unidad funcional (etapa EX)
OP Rx,Ry,Rz	$(Rx \text{ op } Ry) \rightarrow Rz$	ALU	1
LDQ Rx,off(Ry)	$\text{Mem}(Ry+\text{off}) \rightarrow Rx$	Mem	2
LDF Fx,off(Ry)	$\text{Mem}(Ry+\text{off}) \rightarrow Fx$	Mem	2
STQ Rx,off(Ry)	$Rx \rightarrow \text{Mem}(Ry+\text{off})$	Mem	2
STF Fx, off(Ry)	$Fx \rightarrow \text{Mem}(Ry+\text{off})$	Mem	2
ADDF Fx,Fy,Fz	$(Fx + Fy) \rightarrow Fz$	FAD	3
MULF Fx,Fy,Fz	$(Fx \cdot Fy) \rightarrow Fz$	FML	4

1. Realizar el Diagrama de Ejecución Temporal y la Tabla de Reserva para el siguiente programa.

Solución:

En el programa que se analiza aquí tenemos las siguientes instrucciones:

- 1- Una carga al registro F7.
- 2- Una suma de punto flotante, F0 se inicializa con $F7 + F1$. Observar que este repertorio de instrucciones es un poco distinto al MIPS que hemos usado previamente. El registro destino no es el primero que aparece sino es el último.
- 3- Una multiplicación, inicializa F7 con $F7 \cdot F3$.
- 4- Una suma de registro con inmediato, que tenemos como cuarta instrucción.

La cola IQ debe guardar qué instrucciones están en vuelo y a qué registros escriben. F7 da problemas porque genera dependencias de datos. Por ello, se requiere tener en algún sitio anotado que en la unidad funcional 4 hay una carga que escribe en el registro F7. Esta información es la que se utilizaría en la decodificación de la siguiente instrucción para tenerlo en cuenta. Además hay que saber cuánto le queda a la instrucción de carga para generar el registro F7, para que se haga coincidir el envío a ejecutar de la instrucción suma con el camino de realimentación de la salida de la carga.

Tabla de Ejecución Temporal

	Ciclo									
	1	2	3	4	5	6	7	8	9	10
1.LDF F7,0(R7) ;F7←MEMORY(R7)	IF	ID	IQ	M1	M2	WB				
2.ADDF F7,F1,F0 ;F0←F7+F1		IF	ID	IQ!	IQ	FA1	FA2	FA3	WB	
3.MULF F7,F3,F7 ;F7←F7·F3			IF	ID	IQ	FM1	FM2	FM3	FM4	WB
4.ADDQI R7,8,R7 ;R7←R7+8				IF	ID	IQ	ALU	WB		
1.LDF F7,0(R7) ;F7←MEMORY(R7)					IF	ID	IQ	M1	M2	WB

Para entender la Tabla de Ejecución Temporal se requiere considerar lo siguiente:

- 1- Sólo puede entrar una instrucción por ciclo. Entra la primera instrucción en el cauce y va evolucionando observando que los recursos están libres: búsqueda, decodificación, cola IQ (es decir, etapa Issue), acceso a la memoria. El enunciado nos dice que el acceso a memoria tarda dos ciclos en la unidad funcional: M1, M2. Finalmente, cuando se tiene el dato lo escribe en el banco de registros en la etapa de la postescritura en el ciclo 6.
- 2- En la segunda instrucción tenemos una dependencia de tipo WAR. Entra a la cola IQ porque la cola está libre. Fijarse que en el ciclo 4 la primera instrucción se ha ido de la cola, pero ahora resulta que no puede ejecutarse porque no tiene uno de los operandos que es F7. Por eso aparece "IQ!", es decir, se está esperando por los operandos. En el ciclo 6 podemos anticipar el operando desde el bus de Writeback a la entrada del registro de segmentación de la unidad funcional para que empiece a ejecutarse la instrucción. Si no existiera camino de anticipación, tendría que retrasarse un ciclo, porque hay que esperar hasta que se guarde el dato en el banco de registros para luego leer del banco de registro en el siguiente ciclo.
- 3- La operación de multiplicación de la instrucción 3 entra en la cola IQ porque permite alojar hasta dos instrucciones posibles. Hay renombramiento de registros implícito en el cual los operandos que están disponibles los copia localmente. F3 estaría en el banco de registro ya que no existe ninguna instrucción en vuelo que genere F3, por lo que lo leería del banco de registro y lo copiaría y lo colocará en la cola IQ junto con la instrucción indicando que le queda pendiente obtener el registro F7 desde una instrucción que está en vuelo. Cuando F7 se haya generado se anticipa desde la instrucción de carga a

través del camino de anticipación. Fijarse en que la multiplicación se realiza en paralelo con la suma porque van a unidades funcionales independientes.

- 4- La última instrucción no depende de las anteriores; fijarse que trabaja con registros de enteros. Cuando le toca entrar en el ciclo 4 evolucionará si tiene los recursos disponibles. En el ciclo 6 entra en la cola IQ, ya que tiene espacio suficiente. La unidad aritmético lógica tampoco está ocupada con lo cual entra a ejecutarse durante un ciclo y finalmente se escribe fuera de orden en el ciclo 8. Tenemos que tener anotado qué instrucción está en vuelo por si existiese una instrucción posterior que va a utilizar el resultado de esa instrucción. Una opción es anotar esa información en estaciones de reserva (registros físicos internos) que no se especifican en el enunciado del problema.

La tabla de reserva que es lo que aparece a continuación proporciona una visión un poco complementaria a la Tabla de Ejecución Temporal, ya que representa qué instrucción se encuentra en cada etapa. En la Tabla de Reserva no incluye ninguna información nueva, pero permite detectar fácilmente si en una etapa aparecen más instrucciones de lo que se puede. La ALU requiere un ciclo, la operación de memoria 2 ciclos, la suma y resta en punto flotante tres ciclos, la multiplicación 4 etapas de segmentación, y la etapa de postescritura tiene una sola etapa.

Tabla de Reserva

Ciclo	1	2	3	4	5	6	7	8	9	10
IF	1	2	3	4						
ID		1	2	3	4					
Issue-IQ (2)			1	2	3	4				
ALU							4			
Mem1				1						
Mem2					1					
FAD1						2				
FAD2							2			
FAD3								2		
FML1						3				
FML2							3			
FML3								3		
FML4									3	
WB						1		4	2	1



En el ciclo 3, la cola IQ dispone de la primera instrucción. En el ciclo 4 entra en IQ la segunda instrucción que le añadimos una admiración indicando que está parada. En el ciclo 5, IQ tiene las instrucciones 2 y 3. En el ciclo 6 entra la instrucción 4, que en el ciclo 7 pasa a la ALU. En el ciclo 5, la etapa MEM proporciona los datos desde la instrucción 1 a las instrucciones 2 y 3. Observar que en la etapa de postescritura (WB) no aparece ninguna colisión entre instrucciones. Da la casualidad que cuando terminan las instrucciones, ellas lo hacen en distintos ciclos de reloj.

2. ¿Por qué se bloquea la instrucción 2 al final del ciclo 4, y se queda en la cola IQ?

Solución:

Porque existe una dependencia de datos RAW a través del registro F7, el cual es inicializado por la instrucción 1 y utilizado por la instrucción 2 para realizar una operación de suma.

3. Si estas cuatro instrucciones se ejecutaran de forma repetida 100 veces, ¿cuál sería el tiempo total de ejecución? ¿Cuál sería el IPC? ¿Cuántos ciclos de penalización se producen respecto a una ejecución ideal sin problemas? ¿A qué son debidos?

Solución:

Fijarse que estas 4 instrucciones entran en 4 ciclos consecutivos. No se producen paradas porque una instrucción dependa de las anteriores. Todas entran en IF, pasan a ID, evolucionan sin problemas. Por lo tanto, de las 100 iteraciones, las 99 primeras tardan 4 ciclos, y la última iteración tarda 10 ciclos, desde que entra la primera instrucción LDF en la etapa IF hasta que la instrucción MULF llega a la etapa WB.

	Ciclo																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1.LDF F7,0(R7) ;F7←MEMORY(R7)	IF	ID	IQ	M1	M2	WB											
2.ADDF F7,F1,F0 ;F0←F7+F1		IF	ID	IQ!	IQ	FA1	FA2	FA3	WB								
3.MULF F7,F3,F7 ;F7←F7·F3			IF	ID	IQ	FM1	FM2	FM3	FM4	WB							
4.ADDQI R7,8,R7 ;R7←R7+8				IF	ID	IQ	ALU	WB									
1.LDF F7,0(R7) ;F7←MEMORY(R7)					IF	ID	IQ	M1	M2	WB							
2.ADDF F7,F1,F0 ;F0←F7+F1						IF	ID	IQ!	IQ	FA1	FA2	FA3	WB				
3.MULF F7,F3,F7 ;F7←F7·F3							IF	ID	IQ	FM1	FM2	FM3	FM4	WB			
4.ADDQI R7,8,R7 ;R7←R7+8								IF	ID	IQ	ALU	WB					
1.LDF F7,0(R7) ;F7←MEMORY(R7)									IF	ID	IQ	M1	M2	WB			
2.ADDF F7,F1,F0 ;F0←F7+F1										IF	ID	IQ!	IQ	FA1	FA2	FA3	WB
3.MULF F7,F3,F7 ;F7←F7·F3											IF	ID	IQ	FM1	FM2	FM3	FM4
4.ADDQI R7,8,R7 ;R7←R7+8												IF	ID	IQ	ALU	WB	
1.LDF F7,0(R7) ;F7←MEMORY(R7)													IF	ID	IQ	M1	M2
4 ciclos				4 ciclos				4 ciclos				4 ciclos					

99 Iteraciones (1ª; 2ª .. 99ª): $99 \times 4 \text{ ciclos} = 396 \text{ ciclos}$

Última Iteración: 10 ciclos

Tiempo Total de Ejecución = 406 ciclos

Lo siguiente que se nos pide es el IPC (instrucciones por ciclo): se realizan 100 iteraciones, y en cada una de ellas se ejecutan 4 instrucciones, lo que hace un total de 400 instrucciones.

$IPC = 100 \text{ iteraciones} \times 4 \text{ instrucciones/iteración} / 406 \text{ ciclos} = 0.99 \text{ instrucciones/ciclo}$

Fijarse que el IPC es inferior a 1 que es el máximo que se podría obtener, debido a las penalizaciones.

¿Cuántos ciclos de penalización se producen? Para que el IPC fuera 1, las instrucciones se tienen que ejecutarse en 400 ciclos, y por lo tanto:

Penalización = $406 - 400 = 6 \text{ ciclos}$ debidos el vaciado del cauce en la última iteración.


4. Suponer que se introduce un registro de segmentación en la etapa WB para acortar la trayectoria de anticipación (bypass). ¿Cómo se modifican los resultados de los apartados anteriores?

Solución:

Aquí se añade un registro de segmentación en la etapa WB y en el siguiente ciclo de reloj se realiza la realimentación. Fijarse que ahora la primera instrucción de carga escribe en la etapa WB pero no se puede anticipar directamente en el mismo ciclo a la unidad funcional. Después del último registro de segmentación de la unidad funcional el dato pasa al registro de segmentación WB, y luego lo guarda en el banco de registros y se anticipan a las unidades funcionales.

Tabla de Ejecución Temporal de la Primera Iteración

	Ciclo											
	1	2	3	4	5	6	7	8	9	10	11	12
1.LDF F7,0(R7) ;F7←MEMORY(R7)	IF	ID	IQ	M1	M2	WB						
2.ADDF F7,F1,F0 ;F0←F7+F1		IF	ID	IQ!	IQ!	IQ	FA1	FA2	FA3	WB		
3.MULF F7,F3,F7 ;F7←F7·F3			IF	ID	IQ!	IQ	FM1	FM2	FM3	FM4	WB	
4.ADDQI R7,8,R7 ;R7←R7+8				IF	ID!	ID	IQ	ALU	WB			
1.LDF F7,0(R7) ;F7←MEMORY(R7)						IF	ID	IQ!	IQ	M1	M2	WB

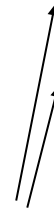


5 ciclos

Lo que ocurre ahora es que la última instrucción ADDQI cuando quiere entrar en la cola IQ en el ciclo 6 no puede porque ya existen dos instrucciones ocupando la cola. Entonces se tiene que parar un ciclo en la etapa ID. Eso ocasiona que las iteraciones a partir de la segunda hasta la penúltima tarden un ciclo más, de 5 ciclos pasan a tardar 6 ciclos.

Tabla de Reserva

Ciclo	1	2	3	4	5	6	7	8	9	10	11
IF	1	2	3	4							
ID		1	2	3	4!	4					
IQ (2)			1	2!	2!3!	2 3	4				
ALU								4			
Mem1				1							
Mem2					1						
FAD1							2				
FAD2								2			
FAD3									2		
FML1							3				
FML2								3			
FML3									3		
FML4										3	
WB						1			4	2	3



99 Iteraciones (1ª; 2ª .. 99ª): $5 + 98 \times 6$ ciclos = 5 + 588 ciclos

Última Iteración: 12 ciclos

Tiempo Total de Ejecución = 605 ciclos

IPC = $100 \text{ iteraciones} \times 4 \text{ instrucciones/iteración} / 605 \text{ ciclos} = 0.66 \text{ instrucciones/ciclo}$

Penalización= 1 ciclo en 1ª iteración, 2 ciclos en cada iteración siguiente (2ª-100ª) debido a RAW (LDF → ADDF,MULF; ADDQI → LDF), y 6 ciclos en la última iteración debido al vaciado del cauce.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1.LDF F7,0(R7) ;F7←MEMORY(R7)	IF	ID	IQ	M1	M2	WB																	
2.ADDF F7,F1,F0 ;F0←F7+F1		IF	ID	IQ!	IQ!	IQ	FA1	FA2	FA3	WB													
3.MULF F7,F3,F7 ;F7←F7·F3			IF	ID	IQ!	IQ	FM1	FM2	FM3	FM4	WB												
4.ADDQI R7,8,R7 ;R7←R7+8				IF	ID!	ID	IQ	ALU	WB														
1.LDF F7,0(R7) ;F7←MEMORY(R7)					IF!	IF	ID	IQ!	IQ	M1	M2	WB											
2.ADDF F7,F1,F0 ;F0←F7+F1	← 5 ciclos →						IF	ID	IQ!	IQ!	IQ!	IQ	FA1	FA2	FA3	WB							
3.MULF F7,F3,F7 ;F7←F7·F3								IF	ID	IQ!	IQ!	IQ	FM1	FM2	FM3	FM4	WB						
4.ADDQI R7,8,R7 ;R7←R7+8									IF	ID!	ID!	ID	IQ	ALU	WB								
1.LDF F7,0(R7) ;F7←MEMORY(R7)										IF!	IF!	IF	ID	IQ!	IQ	M1	M2	WB					
2.ADDF F7,F1,F0 ;F0←F7+F1						← 6 ciclos →						IF	ID	IQ!	IQ!	IQ!	IQ	FA1	FA2	FA3	WB		
3.MULF F7,F3,F7 ;F7←F7·F3														IF	ID	IQ!	IQ!	IQ	FM1	FM2	FM3	FM4	WB
4.ADDQI R7,8,R7 ;R7←R7+8															IF	ID!	ID!	ID	IQ	ALU	WB		
1.LDF F7,0(R7) ;F7←MEMORY(R7)																IF!	IF!	IF	ID	IQ!	IQ	M1	M2
2.ADDF F7,F1,F0 ;F0←F7+F1																			IF	ID	IQ!	IQ!	IQ!
3.MULF F7,F3,F7 ;F7←F7·F3																				IF	ID	IQ!	IQ!
4.ADDQI R7,8,R7 ;R7←R7+8																					IF	ID!	ID!
	5 ciclos					6 ciclos						6 ciclos					6 ciclos						

Las instrucciones 1 de la segunda iteración y 3 de la primera iteración escriben en el mismo registro destino F7. La más nueva es la que prevalece en el banco de registros

Problema 2-2-4. En este ejercicio se analiza cómo un bucle vectorial se ejecuta en la versión del procesador DLX32000, el cual es un procesador escalar y realiza planificación dinámica de instrucciones utilizando el algoritmo de Tomasulo. El que aparece a continuación es un código DLX, donde se supone que el inmediato **done** y los registros **R1**, **R2**, y **F0** se han inicializado antes de comenzar a ejecutarse el bucle:

```

ini:      LD      F2,0(R1)
          MULTD   F4,F2,F0
          LD      F6,0(R2)
          ADDD    F6,F4,F6
          SD      0(R2),F6
          ADDI    R1,R1,#8
          ADDI    R2,R2,#8
          SGTI    R3,R1,done
          BEQZ    R3,ini
  
```

Suponer que se dispone de una unidad funcional de enteros (denominada “E”), otra para cargas (denominada “C”), y otras para almacenamientos (denominada “A”). Todas estas unidades funcionales tardan 1 ciclo de reloj en terminar la operación desde que los operandos están disponibles. Adicionalmente, existe otra unidad funcional que realiza las operaciones de multiplicación en punto flotante (denominada “M”) que tarda 4 ciclos de reloj, y una unidad funcional que realiza las sumas en punto flotante (denominada “S”) que tarda 3 ciclos de reloj en realizar la operación desde que los operandos están disponibles. Los saltos se resuelven en la unidad de enteros. Suponer que en cada una de las estaciones de reserva de las unidades funcionales se pueden alojar hasta tres instrucciones, a excepción de E que puede alojar cuatro instrucciones. Suponer que las unidades funcionales pueden empezar a ejecutar la operación correspondiente tan pronto como se disponga de los datos en la estación de reserva. Rellena la siguiente tabla, mostrando el estado de las estaciones de reserva que estén ocupadas cuando la instrucción SGTI escribe por primera vez el resultado en el CDB.

ID	Instrucción	Etapa		
		IS	EX	WB
1	ini: LD F2,0(R1)			
2	MULTD F4,F2,F0			
3	LD F6,0(R2)			
4	ADDD F6,F4,F6			
5	SD 0(R2),F6			
6	ADDI R1,R1,#8			
7	ADDI R2,R2,#8			
8	SGTI R3,R1,done			
9	BEQZ R3,ini			

Solución.

Se pide mostrar el estado de las estaciones de reserva, en el ciclo que la instrucción SGTI se esté ejecutando. A continuación, se completa la **tabla de evolución temporal** de las instrucciones a lo largo del cauce del procesador DLX000. No es exactamente lo que se pide en el ejercicio, pero lo necesitamos para saber qué ha sucedido en cada ciclo y llegar a conocer el estado de las estaciones de reserva en el ciclo que la instrucción SGTI se ejecuta en su unidad funcional.

Seguidamente, se describen los eventos que le ocurren a cada instrucción en el orden secuencial del programa.

Instrucción				IS	EX	WB	Observaciones
1	ini:	LD	F2, 0(R1)	1	2	3	
2		MULTD	F4, F2, F0	2	4-7	8	
3		LD	F6, 0(R2)	3	4	5	
4		ADDD	F6, F4, F6	4	9-11	12	
5		SD	0(R2), F6	5	13		
6		ADDI	R1, R1, #8	6	7-8	9	1 ciclo esperando desocuparse CDB
7		ADDI	R2, R2, #8	7	9	10	1 ciclo esperando desocuparse E
8		SGTI	R3, R1, done	8	10	11	
9		BEQZ	R3, ini	9	12		

Instrucción 1. La primera instrucción se envía a ejecutar y evoluciona sin paradas.

Instrucción 2. En el ciclo 3 puede tomar el dato del registro F2 que genera la instrucción anterior a través del bus de datos común (CDB). A partir del ciclo 4 se ejecuta en la unidad funcional de multiplicaciones y su ejecución tarda 3 ciclos de reloj en la unidad funcional. Cuando escribe el resultado de la multiplicación en la etapa WB, otras instrucciones dependientes copian ese dato desde el bus de datos común a su estación de reserva, además de almacenar el valor del registro en el banco de registro.

Instrucción 3. Evoluciona sin paradas a partir del ciclo 3 debido a que no tiene dependencias de datos con las anteriores instrucciones y su unidad funcional está disponible. En el ciclo 5 no tiene impedimentos de realizar la actualización del banco de registros porque el bus de datos común está libre.

Instrucción 4. Necesita los operandos F4 y F6. El operando que más va a tardar es F4, que se escribe en el ciclo 8 en la estación de reserva y banco de registros. Por ello, se podrá operar sobre él a partir del ciclo 9. F6 habrá sido generado desde el ciclo 5 (se puede utilizar este resultado a partir del ciclo 6). Por lo tanto, la ejecución de la instrucción ADDD en su unidad funcional ocurre desde el ciclo 9 al 11 (3 ciclos de ejecución por ser una instrucción en punto flotante).

Instrucción 5. El almacenamiento necesita el valor de F6, el cual es accesible después del ciclo 12.

Instrucción 6. La suma de valores enteros con un inmediato dispone de todos los operandos y al menos una unidad funcional libre para poder ejecutarse. Se produce un ciclo de penalización al intentar utilizar el bus de datos común para escribir el resultado. Una instrucción más antigua, la instrucción MULTD, hace uso del bus de datos común para escribir su resultado en el ciclo 8. Por ello, la suma debe esperar a que esté el bus libre, que en este caso será un ciclo después, en el ciclo 9.

Instrucción 7. La parada de la instrucción anterior en su unidad funcional a la espera que el bus se desocupe ocasiona que esta instrucción no pueda utilizar la unidad funcional que le corresponde. Por ello, hasta el ciclo 9 no puede empezar a realizar la operación de suma. Luego evoluciona sin paradas.

Instrucción 8. Esta instrucción tiene una dependencia con la instrucción 6 a través del registro R1. Por ello, debe esperar en su estación de reserva hasta el ciclo 10, en el cual se ejecuta en la unidad funcional de saltos. Escribe el resultado en el ciclo 11 ya que ninguna otra unidad funcional utiliza el bus de dato común.

Seguidamente se ejecutaría la instrucción de salto, pero ya sabemos lo que nos pide en el enunciado: la instrucción SGTI finaliza en el ciclo 11. Ahora rellenaremos la tabla que nos pide el problema, en la que se visualiza el estado de las estaciones de reserva.

CICLO 11: estaciones de reserva

Estación de reserva	Ocupada?	Operación	V _j	V _k	Q _j	Q _k
E1	No					
E2	No					
E3	Si	SGTI	R1	done		
E4	Si	BEQZ	R3	ini		
S1	No					
S2	Si	ADDD	F4	F6		
S3	No					
A1	Si	SD	0(R2)			S2
A2 y A3	No					
C1, C2, C3	No					
M1,M2,M3	No					

El estado de las estaciones de reserva indica lo siguiente:

- La instrucción de almacenamiento en la estación A1 indica en Q_k que la estación de reserva de la instrucción de la cual depende (ADDD) es S2.
- ADDD se encuentra en el tercer y último ciclo de su ejecución ocupando la estación S2.
- La instrucción SGTI ha finalizado ya que en el ciclo 11 escribe el resultado en el CDB y en el banco de registros. En el ciclo siguiente se desocupa la estación donde se encuentra: E3.
- La instrucción de salto condicional esta parada, esperando el resultado que genera la instrucción SGTI. En el ciclo 11 los copia desde el bus de dato común, con lo cual en el ciclo siguiente el salto puede empezar a ejecutarse.
- Las cuatro estaciones de reserva de enteros se ocupan hasta el ciclo 9. A partir del ciclo 10, sólo están ocupadas tres estaciones de reserva, y a partir del ciclo 11, existe dos estaciones de reserva E ocupadas.

En la siguiente tabla de evolución temporal se puede observar la etapa en que se encuentra cada instrucción para cada ciclo de reloj. Adicionalmente, la anticipación de datos se representa con flechas desde unas instrucciones a otras, la cual se realiza a través del copiado de datos desde el bus de datos común a las estaciones de reserva donde se alojan las instrucciones dependientes.

			CICLOS de RELOJ												
Instrucción			1	2	3	4	5	6	7	8	9	10	11	12	13
1	ini:	LD F2,0(R1)	IS	EX	WB										
2		MULTD F4,F2,F0		IS	EX!	EX	EX	EX	WB						
3		LD F6,0(R2)			IS	EX	WB								
4		ADDD F6,F4,F6				IS	EX!	EX!	EX!	EX!	EX	EX	EX	WB	
5		SD 0(R2),F6					IS	EX!	EX!	EX!	EX!	EX!	EX!	EX!	EX
6		ADDI R1,R1,#8						IS	EX	EX!	WB				
7		ADDI R2,R2,#8							IS	EX!	EX	WB			
8		SGTI R3,R1,done								IS	EX!	EX	WB		
9		BEQZ R3,ini									IS	EX!	EX!	EX	

Observar que la instrucción 6 (ADDI) se retrasa 1 ciclo en el ciclo 8 porque la instrucción 2 (MULTD) está ocupando el bus CDB durante su etapa de WB.

TEMA 2-3: Ejecución Especulativa de Instrucciones

Problema 2-3-1 (HP96, ex.4.10, pp.365). Suponer que se dispone de un procesador altamente segmentado para el que se implementa un "Branch Target Buffer (BTB)" sólo para los saltos condicionales. Suponer que la penalización por fallo en la predicción del salto es siempre de 4 ciclos de reloj y la *penalización por fallo* de la BTB es siempre de 3 ciclos de reloj. Suponer que se tiene una *tasa de acierto* en la BTB del 90% (de cada 10 accesos a la BTB por parte de una instrucción de salto condicional, en 9 de ellos existe información de ese salto), una *precisión* del 90% (de cada 10 predicciones, se acierta en 9 de ellas), y una *frecuencia de salto* del 15% (de cada 100 instrucciones ejecutadas, 15 de ellas son saltos condicionales). ¿Cuánto más rápido es el procesador con la BTB en relación a un procesador que tiene una penalización fija de ciclos en los saltos? Suponer un CPI ideal sin paradas por saltos de 1. Suponer que se produce una penalización de 2 ciclos de reloj por dependencias de control en el procesador que no tiene BTB.

Solución:

Se comparan procesadores SIN y CON BTB para saltos condicionales:

$$Speed - Up = \frac{CPI_{noBTB}}{CPI_{BTB}} = \frac{CPI_{base} + Penalizaciones_{noBTB}}{CPI_{base} + Penalizaciones_{BTB}}$$

Según enunciado: $CPI_{base} = 1$

$$Penalizaciones = \sum_i Frecuencia(instrucción_i) \times Penalizaciones(instrucción_i)$$

$$Penalizaciones_{noBTB} = 15\% \times 2 = 0.3 \text{ ciclos / instrucción}$$

$$Penalizaciones_{BTB} = fallo - BTB + acierto - BTB(predicción correcta) + acierto - BTB(predicción incorrecta)$$

Acierto/Fallo BTB	Predicción BTB	frecuencia(instrucción)	Penalización (ciclos)
Fallo		$15\% \times 10\% = 1.5\%$	3
Acierto	correcta	$15\% \times 90\% \times 90\% = 12.1\%$	0
Acierto	incorrecta	$15\% \times 90\% \times 10\% = 1.3\%$	4

$$Penalizaciones_{BTB} = 1.5\% \times 3 + 12.1\% \times 0 + 1.3\% \times 4 = 0.097 \text{ ciclos / instrucción}$$

$$Speed - Up = \frac{1 + 0.3}{1 + 0.097} = 1.2X \quad (15.6\% \text{ más rápido})$$

Problema 2-3-2 (HP96, ex.4.11, pp.365). Determinar la mejora introducida por la técnica "Branch Folding" para saltos incondicionales en el procesador DLX32p que resuelve los saltos en la etapa ID y que no acepta saltos retardados. Esta técnica consiste en una BTB en la que se almacena la dirección de destino del salto y además la instrucción almacenada en la dirección de destino del salto. Suponer una frecuencia de acierto en la BTB del 90%, un CPI base de 1 sin paradas por saltos incondicionales, y una frecuencia de saltos incondicionales del 5%. ¿Cuál es la ganancia de rendimiento en relación a un procesador cuyo CPI efectivo es de 1.1?

Solución:

Esta técnica nos dice que cuando la BTB falla en un salto, la penalización es de 1 ciclo de reloj, debido a que no entra una instrucción en la etapa IF, ya que hay que desecharla. En caso de acierto de predicción, la instrucción destino del salto aparece en la etapa ID directamente sin pasar por la etapa IF. Esto equivale a una reducción de 1 ciclo de reloj, es decir, la penalización es negativa: -1 ciclo.

Considerando estos datos, calculamos el CPI del procesador:

$$\text{CPI} = \text{CPI base} + \text{CPI penalización}$$

$$\text{CPI por BTB} = \text{Penalización por acierto} + \text{Penalización por fallo}$$

$$\text{Penalización por acierto} = 0.9 (\% \text{ acierto BTB}) \times 0.05 (\% \text{ saltos}) \times -1 (\text{ciclo de penalización negativa})$$

$$\text{Penalización por acierto de BTB y por instrucción} = -0.045 \text{ ciclos}$$

$$\text{Penalización por fallo} = (1 - 0.9) (\% \text{ fallo BTB}) \times 0.05 (\% \text{ saltos}) \times 1 (\text{ciclo de penalización})$$

$$\text{Penalización por fallo de BTB y por instrucción} = 0.005 \text{ ciclos}$$

$$\text{CPI por BTB} = 1 - 0.045 + 0.005 = 0.96 \text{ ciclos}$$

La ganancia de prestaciones frente a una máquina de referencia con CPI de 1.1 ciclos será:

$$\text{Speed-Up} = \text{CPI referencia} / \text{CPI por BTB} = 1.1 / 0.96 = 1.15X$$

En conclusión, el procesador con BTB es un 12.7% mejor: $12.7\% = 100 \times (1.1 - 0.96) / 1.1$

Problema 2-3-3 (HP96, ex.4.14, pp.365). En este ejercicio se analizará cómo un bucle vectorial se ejecuta en distintas versiones del DLX^{32p}. El bucle es el llamado SAXPY y es la operación central del algoritmo de Eliminación Gaussiana. El bucle implementa la operación vectorial $Y = a * X + Y$ para un vector de longitud 100. Este es su código DLX:

```
foo:  LD      F2,0(R1)      ; carga X(i)
      MULTD   F4,F2,F0     ; a X(i)
      LD      F6,0(R2)     ; carga Y(i)
      ADD     F6,F4,F6     ; a X(i) + Y(i)
      SD      0(R2),F6     ; almacena Y(i)
      ADDI    R1,R1,#8     ; incrementa índice de X
      ADDI    R2,R2,#8     ; incrementa índice de Y
      SGTI    R3,R1,done   ; R3=1 si R1 > done
      BEQZ    R3,foo       ; vuelve a foo si R3=0
```

Para los siguientes apartados de la (a) a la (e) suponer que las operaciones sobre enteros se envían a ejecución y se completan en 1 ciclo de reloj (incluyendo las cargas) y que los resultados están disponibles a las restantes unidades funcionales tan pronto como se obtengan. Ignorar el retardo de los saltos. Utilizar las latencias del problema 3. Suponer que la unidad de coma flotante está completamente segmentada.

- a. DLX^{32p} segmentado con **anticipación y envío a ejecución de 1 instrucción por ciclo**. Mostrar el número de ciclos de parada para cada instrucción y en qué ciclo de reloj cada instrucción comienza su ejecución en la etapa EX para la primera iteración del bucle. ¿Cuántos ciclos de reloj tarda cada iteración del bucle?

Solución.

Se suponen estas latencias,

FP ALU op	→	Another FP ALU op	: 3 ciclos
FP ALU op	→	Store double	: 2 ciclos
Load double	→	FP ALU op	: 1 ciclos
Load double	→	Store double	: 0 ciclos

Se suponen unidades funcionales de coma flotante completamente segmentadas

La evolución de los ciclos en los que cada instrucción comienza la etapa EX es la siguiente:

			1er ciclo en EX
foo:	LD	F2,0(R1)	; 1
			; 2, MULT se para por dependencia F2
	MULTD	F4,F2,F0	; 3
	LD	F6,0(R2)	; 4
			; 5, se para ADD por dependencia con LD
			; 6, se para ADD 2 ciclos por dependencia de F4
	ADD	F6,F4,F6	; 7
			; 8
			; 9, se para por dependencia con F6
	SD	0(R2),F6	; 10
	ADDI	R1,R1,#8	; 11
	ADDI	R2,R2,#8	; 12
	SGTI	R3,R1,done	; 13
	BEQZ	R3,foo	; 14
			; 15, ciclo de espera por salto retardado

Supongamos que en todos los casos que vamos a comparar no se considera la penalización por saltos retardados. Por lo tanto, el número de ciclos por iteración es: 14 (9 de ejecución y 5 de parada)

- b. Desarrollar el código DLX para SAXPY y realizar 4 copias del cuerpo del bucle y planificar su ejecución para la ruta segmentada del DLX^{32p} con una unidad FPU segmentada con las latencias mencionadas anteriormente. ¿Cuántos ciclos de reloj tarda cada iteración?

Solución.

```

inicio: LD      F2,0(R1)      ; X[0]
        LD      F4,8(R1)     ; X[1]
        LD      F6,16(R1)    ; X[2]
        LD      F8,24(R1)    ; X[3]
        LD      F10,0(R2)    ; Y[0]
        LD      F12,8(R2)    ; Y[1]
        LD      F14,16(R2)   ; Y[2]
        LD      F16,24(R2)   ; Y[3]
        MULTD   F18,F2,F0     ; a*X
        MULTD   F20,F4,F0     ; a*X
        MULTD   F22,F6,F0     ; a*X
        MULTD   F24,F8,F0     ; a*X
        ADDD    F2,F18,F10     ; a*X[0]+Y[0]
        ADDD    F4,F20,F12     ; a*X[1]+Y[1]
        ADDD    F6,F22,F14     ; a*X[2]+Y[2]
        ADDD    F8,F24,F16     ; a*X[3]+Y[3]
        ADDI    R1,R1,#32      ; R1=R1+32
        ADDI    R2,R2,#32      ; R2=R2+32
        SD      -32(R2),F2     ; Y[0]= a*X[0]+Y[0]
        SD      -24(R2),F4     ; Y[1]= a*X[1]+Y[1]
        SD      -16(R2),F6     ; Y[2]= a*X[2]+Y[2]
        SD      -8(R2),F8      ; Y[3]= a*X[3]+Y[3]
        SGTI    R3,R1, 800     ;
        BNEZ    R3, inicio     ;
800:

```

Ciclos por iteración: 24 clks / 4 iteraciones = 6 clks/iteración

Registros: 16 registros (13 FP + 3 enteros)

- d. Considerar ahora el algoritmo de Tomasulo para un hardware que dispone de una unidad de enteros con 1 ciclo de ejecución para todas las operaciones de enteros. Mostrar el estado de las estaciones de reserva y las tablas de estado de los registros cuando el salto condicional es ejecutado por segunda vez.

Solución.

Latencias en la correspondiente unidad funcional:

LD/SD: 1 clk

ADDD/MULTD: 4 clks (latencia+1)

Enteros: 1 clk

WB en Estaciones de reserva: 1 clk

Suponemos un único bus de datos común (no puede haber dos instrucciones que terminen en WB en el mismo ciclo), y una unidad de carga/almacenamiento independiente.

		IS	EX	WB	Observaciones
LD	F2,0(R1)	1	2	3	
MULTD	F4,F2,F0	2	4-7	8	
LD	F6,0(R2)	3	4	5	
ADDD	F6,F4,F6	4	9-12	13	
SD	0(R2),F6	5	14		
ADDI	R1,R1,#8	6	7	9	+1clk: conflicto en CDB
ADDI	R2,R2,#8	7	8	10	+1clk: conflicto en CDB
SGTI	R3,R1,800	8	10	11	+1clk: dependencia R1
BEQZ	R3,inicio	9	12		
LD	F2,0(R1)	10	11	12	LD y SD en clk 11
MULTD	F4,F2,F0	11	13-16	17	
LD	F6,0(R2)	12	13	14	
ADDD	F6,F4,F6	13	18-21	22	
SD	0(R2),F6	14	23		
ADDI	R1,R1,#8	15	16	18	+1clk: conflicto en CDB
ADDI	R2,R2,#8	16	17	19	+1clk: conflicto en CDB
SGTI	R3,R1,800	17	19	20	+1clk: dependencia R1
BEQZ	R3,inicio	18	21		

Si observamos los ciclos de IS, se tarda 9 clk/iteración.

Estado de las estaciones de reserva

Ciclo 21: BEQZ ejecutado por segunda vez

	Ocupada	Operación	Vj	Vk	Qj	Qk
Ent1	si	BEQZ	RR2			
FP1	si	ADDD	MUL1	LD1		
Load1	no					

Tablas de estado: en el ciclo 21 todos los registros están actualizados con los valores verdaderos generados por el programa.

f. Considerar ahora el algoritmo de Tomasulo con una unidad de enteros y otra de punto flotante completamente segmentadas. Tomar las latencias siguientes

FP ALU op	→	Another FP ALU op	: 3 ciclos
FP ALU op	→	Store double	: 2 ciclos
Load double	→	FP ALU op	: 1 ciclos
Load double	→	Store double	: 0 ciclos
Op sobre enteros	→	cualquiera	: 0 ciclos

Mostrar el estado de las estaciones de reserva y las tablas de estado de los registros cuando el salto condicional es ejecutado en la segunda ocasión. Suponer que el salto fue correctamente predecido. ¿Cuántos ciclos de reloj tarda cada iteración?

Solución.

PENDIENTE DE ACTUALIZAR

- g. Suponer una arquitectura superescalar que puede enviar a ejecutar 2 operaciones independientes en un ciclo de reloj (incluyendo 2 operaciones sobre enteros). Realiza cuatro copias del cuerpo del bucle SAXPY y planifica su ejecución suponiendo las latencias descritas en el problema 3. Suponer que se tiene una copia completamente segmentada de cada unidad funcional (sumador FP, multiplicador FP, etc.) y 2 unidades funcionales sobre enteros con latencia 0. ¿Cuántos ciclos de reloj requiere cada iteración del bucle original?. ¿Cuál es el speedup respecto al código original?.

Solución.

Se supone que la instrucción MULTD tiene asociada 4 ciclos en la etapa EX, y la instrucción ADDD, 2.

		ISSUE	EX
top:	LD F2,0(R1)	; X[0], 1	2
	LD F4,8(R1)	; X[1], 1	2
	LD F6,16(R1)	; X[2], 2	3
	MULTD F18,F2,F0	; a*X, 2	3 .. 6
	LD F8,24(R1)	; X[3], 3	4
	MULTD F20,F4,F0	; a*X, 3	4 .. 7
	LD F10,0(R2)	; Y[0], 4	5
	MULTD F22,F6,F0	; a*X, 4	5 .. 8
	LD F12,8(R2)	; Y[1], 5	6
	MULTD F24,F8,F0	; a*X, 5	6 .. 9
	LD F14,16(R2)	; Y[2], 6	7
	ADDD F2,F18,F10	; 6	7 .. 9
	LD F16,24(R2)	; Y[3], 7	8
	ADDD F4,F20,F12	; 7	8 .. 10
	ADDI R1,R1,#32	; 8	9
	ADDD F6,F22,F14	; 8	9 .. 11
	ADDI R2,R2,#32	; 9	10
	ADDD F8,F24,F16	; 9	10 .. 12
	SGTI R3,R1, done	; 10	11
	SD -32(R2),F2	; 10	11
	SD -24(R2),F2	; 11	12
	SD -16(R2),F2	; 11	12
	SD -8(R2),F2	; 12	13
	BNEZ R3, top	; 12	13

done:

Se realizan 4 iteraciones en 12 ciclos, lo que equivale a 3 ciclos/iteración.

Con respecto al apartado (a), Speed-Up = $14/3 = 4.7$

□

h. En un procesador supersegmentado, en vez de tener múltiples unidades funcionales, se segmentan todas las unidades. Suponer que se diseña un DLX supersegmentado que tendría el doble de frecuencia de reloj que el DLX estándar y que puede enviar a ejecutar 2 instrucciones no dependientes a la vez. Si la segunda instrucción depende de la primera, sólo se envía a ejecutar la primera instrucción. Desarrolla el código SAXPY del DLX para hacer 4 copias del cuerpo del bucle y planifica su ejecución para un procesador supersegmentado suponiendo que las latencias de las unidades funcionales son las que aparecen a continuación. Suponer que la latencia de las instrucciones de carga es de 1 ciclo del procesador segmentado (2 ciclos del supersegmentado), y que el resto de instrucciones sobre enteros tiene latencia 0. ¿Cuántos ciclos de reloj requiere cada iteración del bucle?. Utilizar la siguiente tabla para los ciclos de latencia del procesador supersegmentado.

Instrucción produce resultado		Instrucción leyendo resultado	Ciclos latencia
FP MULT	→	FP ALU op	: 6 ciclos
FP ADD	→	FP ALU op	: 4 ciclos
FP MULT	→	FP store	: 5 ciclos
FP ADD	→	FP store	: 3 ciclos
Op sobre enteros	→	cualquiera	: 0 ciclos

Solución.

PENDIENTE DE ACTUALIZAR

- i. Utilizar ahora el procesador VLIW y el bucle SAXPY. Desarrolla el bucle SAXPY cuatro veces y realiza optimizaciones simples. Suponer que las latencias de las unidades funcionales sobre enteros tienen latencia 0 y que las latencias de las unidades funcionales FP son las del problema 3. Planifica el código para la arquitectura VLIW. . ¿Cuántos ciclos de reloj requiere cada iteración del bucle?.

Solución.

Referencia Memoria 1	Referencia Memoria 2	Operación FP 1	Operación FP 2	Op. Enteros/ Salto	Ciclo Reloj
LD F2,0(R1)	LD F8,8(R1)				1
LD F14,16(R1)	LD F20,24(R1)				2
LD F6,0(R2)	LD F12,8(R2)				3
LD F18,16(R2)	LD F24,24(R2)			ADDI R1,R1,#32	4
		MULTD F4,F2,F0	MULTD F10,F8,F0	ADDI R2,R2,#32	5
		MULTD F16,F14,F0	MULTD F22,F20,F0	SGTI R3,R1,done	6
					7
					8
		ADDD F6,F4,F6	ADDD F12,F10,F12		9
		ADDD F18,F16,F18	ADDD F24,F22,F24		10
					11
SD -32(R2),F6	SD -24(R1),F12				12
SD -16(R2),F18	SD -8(R1),F24			BEQZ R3,foo	13

Se realizan 4 iteraciones en 13 ciclos, lo que equivale a 3.25 ciclos/iteración.

Observar que se ejecutan 24 instrucciones, lo que equivale a un poco menos de 2 instrucciones por ciclo de reloj.

□

j. Suponer ahora que se dispone de un procesador especulativo, y que se dispone de las siguientes latencias de las unidades funcionales: Multiplicación, 4 ciclos; Sumas en punto flotante, 3 ciclos; operaciones sobre enteros, 1 ciclo. Mostrar el estado del procesador cuando el salto condicional se envía a ejecutar por segunda vez. Suponer que se acierta en la predicción del salto y que necesitó 1 ciclo de reloj en su unidad funcional. ¿Cuántos ciclos de reloj requiere cada iteración del bucle?.

Solución:

Resumen de características:

- Procesador escalar especulativo: planificación dinámica de saltos, con ejecución fuera de orden y búfer de reordenamiento para retirado en orden de las instrucciones.
- Los saltos se predicen de forma correcta y tardan 1 ciclo. Se supone que se utiliza una tabla BTB.
- Latencias en unidades funcionales:
 - o 4 ciclos Multiplicación.
 - o 3 ciclos Sumas en punto flotante.
 - o 1 ciclo operaciones en entero.
- Se supone que en la etapa de retirado sólo se puede retirar 2 instrucciones cualquiera cada ciclo excepto los saltos, debido al riesgo de retirar una instrucción mal especulada.
- Se supone que el procesador cuenta con capacidad suficiente en la cola de instrucciones y estaciones de reserva para albergar todas las instrucciones en vuelo.
- Se supone que las instrucciones de almacenamiento, por eficiencia, se resuelven en la etapa de retirado (donde se calcula la dirección y se accede a la cache de datos), y por lo tanto carecen de etapas EX (el sencillo cálculo de la dirección puede llevarse a la etapa de retirado) y WB.

A continuación se muestra el diagrama de ciclos donde la primera columna de la izquierda muestra las dos iteraciones del bucle y en las sucesivas columnas se mostrará el estado en cada ciclo de las instrucciones. Como se ve, se duplica el código dos veces debido a la planificación dinámica del procesador.

[illegible]

1. LD F2, 0(R1):

Entra en el cauce en el ciclo 1 y al ser la primera instrucción avanza sin problema por las etapas del procesador. Atraviesa las etapas de búsqueda, decodificación, envío a las estaciones de reserva y ejecución (ciclos 1, 2, 3, 4). En el ciclo 5 entra en writeback y escribe el resultado en el búfer de reordenamiento y en las estaciones de reserva que se encontraran esperando por su valor, en este caso, la multiplicación que le sucede. En el ciclo 6 se retira del búfer y se actualiza el registro F2 con el valor de la carga en el banco de registros.

2. MULTD F4, F2, F0:

Entra en el cauce en el ciclo 2 y avanza sin problema por las etapas IF, ID, IS en los ciclos consecutivos. En el ciclo 5 debería entrar a ejecutarse pero existe una dependencia de tipo verdadero con el registro F2 cargado en la instrucción anterior. Por ello, debe esperar a que la instrucción anterior actualice el valor de F2 en la estación de reserva de la multiplicación en el ciclo 5, para que posteriormente pueda comenzar a ejecutarse en el ciclo 6. La multiplicación necesita 4 ciclos (ciclos 6, 7, 8, y 9). En el ciclo 10 escribirá en el búfer de reordenamiento y en las estaciones de reserva pertinentes, en este caso en la estación de reserva de la suma en punto flotante.

3. LD F6, 0(R2):

Entra en el cauce en el ciclo 3 y avanza sin problema por las etapas IF, ID, IS, EX en los ciclos consecutivos. En el ciclo 7 escribe en el búfer de reordenamiento, adelantándose a la multiplicación anterior, pero no se puede retirar hasta el ciclo 11 (se realiza ejecución fuera de orden pero el retirado debe hacerse en orden). La etapa de retirado permite retirar hasta dos instrucciones en un mismo ciclo.

4. ADDD F6, F4, F6:

Entra en el cauce en el ciclo 4 y avanza sin problema por las etapas IF, ID e IS. Existe una dependencia de tipo verdadero con la instrucción de carga anterior debido al registro F6, pero además tiene otra dependencia del mismo tipo con la multiplicación anterior a través del registro F4. Por ello, la ejecución de la suma debe retrasarse hasta el ciclo 11, justo después de que la multiplicación actualice el valor en la estación de reserva y por fin se disponga de todos los operandos. La suma permanece ejecutándose en la unidad funcional durante 3 ciclos (ciclos 11, 12 y 13). En el ciclo 14 escribe en el búfer de reordenamiento y en la estación de reserva del almacenamiento que le sucede.

5. SD 0(R2), F6:

Entra en el cauce en el ciclo 5 y avanza sin problema por las etapas IF, ID e IS. Como se ha comentado, se supone que por sencillez los almacenamientos se resuelven directamente en la etapa de retirado. El cálculo de dirección que se realizaría en EX se puede mover fácilmente y sin apenas coste a la etapa de retirado, y al ser un almacenamiento, tampoco genera dependencias y no adelantaría el dato a nadie. De esta forma se pasa directamente a la etapa Ret. Sin embargo, la retirada debe hacerse en orden; podría provocar un riesgo por ambigüamiento de la memoria si no se hiciera así. Por ello, no es hasta el ciclo 15 cuando puede almacenarse el dato en la memoria.

6. ADDI R1, R1, #8:

Entra en el cauce en el ciclo 6 y avanza sin problema por las etapas IF, ID, IS, EX en los ciclos sucesivos. La suma en enteros tiene una latencia de 1 ciclo. En el ciclo 10 intenta escribir en el búfer de reordenamiento y en las estaciones de reserva que proceda, en este caso, en la instrucción SGTI posterior, pero el Bus de Datos Común se encuentra ocupado en ese momento por la multiplicación y debe esperar al ciclo 11. Aunque la instrucción haya terminado y se haya escrito en el búfer de reordenamiento en el ciclo 11, debe permanecer en espera y no puede retirarse hasta llegado el ciclo 16.

7. ADDI R2, R2, #8:

Entra en el cauce en el ciclo 7 y procede de igual forma que la homónima instrucción precedente. De igual manera termina su ejecución, pero no puede escribir en el bus de datos común debido a que la instrucción anterior lo está ocupando. Debe esperar al ciclo 12 para escribir en el búfer de reordenamiento y hasta el ciclo 16 para poder ser retirada.

8. SGTI R3, R1, done:

Entra en el cauce en el ciclo 8 y avanza por las etapas de IF, ID e IS sin problemas en los ciclos consecutivos. Existe una dependencia verdadera RAW a través del registro R1 con la instrucción de suma anterior que hace detenerse un ciclo. Finalmente, la suma actualiza su valor en la estación de reserva en el ciclo 11 y la instrucción puede entrar a ejecutarse en el ciclo 12. En el ciclo 13 escribe en el búfer de reordenamiento y en la estación de reserva del salto posterior. Como en los casos anteriores debe esperar hasta ser retirada. El retirado se lleva a cabo en el ciclo 17.

9. BEQZ R3, foo:

Entra en el cauce en el ciclo 9 y avanza por las etapas de IF, ID e IS sin problemas en los ciclos consecutivos. Estando en la etapa IF, la tabla BTB reconoce la instrucción y predice el salto correctamente (como se supone), y por ello, la instrucción destino del salto puede entrar en el cauce en el ciclo 10. Como tiene una dependencia de tipo verdadera con la instrucción anterior, no puede entrar en EX hasta el ciclo 14. En el ciclo 15 escribe, pero no puede retirarse hasta el ciclo 18. Las etapas de WB y Ret son necesarias dado que no se sabe si la especulación es correcta hasta la retirada. En caso de no ser correcta habría que borrar las instrucciones posteriores. Por razones de seguridad, las instrucciones de salto deben retirarse solas, ya que sería un error retirar una instrucción mal especulada a la vez que se comprueba la predicción del salto.

Finalizada la primera iteración se procede a ejecutar la segunda, la cual evoluciona a lo largo del cauce de idéntica manera que la primera iteración. La única diferencia es un retraso encadenado de un ciclo que se propaga por las instrucciones, provocado por la competencia por el bus de datos común. La evolución completa puede verse en el diagrama anterior. Como puede observarse, entre la primera instrucción de la primera iteración y la primera instrucción de la segunda pasan 9 ciclos (del ciclo 1 al 9), con lo que se puede lanzar a ejecutar una iteración completa cada 9 ciclos. Se observa también que la última iteración tarda 10 ciclos en terminar la ejecución (del ciclo 19 al 28), con lo que la última iteración tiene una duración de $9+10=19$ ciclos.

Un detalle a tener en cuenta es el ritmo de salida de instrucciones del cauce. Hemos concluido que podemos lanzar una iteración cada 9 ciclos. Si nos fijamos en el tiempo que transcurre entre el ciclo de la última instrucción en ejecutarse de la primera iteración y el ciclo de la última instrucción en ejecutarse de la segunda, podemos ver que son necesarios 10 ciclos para que una iteración abandone el cauce. Por lo tanto, podemos lanzar una iteración cada 9 ciclos, pero se necesitan 10 ciclos para que lo abandone. Esto provocará a la larga una saturación de los registros de la cola de instrucciones y las estaciones de reserva. Sin embargo, se ha supuesto que el procesador posee capacidad suficiente para albergar todas las instrucciones en vuelo.

- j. Suponer que se dispone de un procesador superescalar especulativo que puede buscar, decodificar y enviar a ejecutar dinámicamente una combinación compuesta por 1 instrucción de carga/almacenamiento, una operación sobre enteros, y una operación FP en cada ciclo de reloj, y que las latencias de las unidades funcionales son :

FP ALU op	→	FP ALU op	: 3 ciclos
FP ALU op	→	Store double	: 2 ciclos
Load double	→	FP ALU op	: 1 ciclos
Load double	→	Store double	: 0 ciclos

Mostrar el estado del procesador cuando el salto condicional se envía a ejecutar por segunda vez. Suponer que el salto es correctamente especulado y que necesitó 1 ciclo de reloj en su unidad funcional. ¿Cuántos ciclos de reloj requiere cada iteración del bucle?

Solución:

Resumen de características:

- Procesador superescalar especulativo (planificación dinámica de saltos, y con ejecución fuera de orden y buffer de reordenamiento) que puede lanzar hasta 3 instrucciones por ciclo, cuya combinación sea del tipo: carga/almacenamiento, operación en enteros y operación en FP.
- Los saltos se predicen de forma perfecta y tardan 1 ciclo. Se supone el uso de BTB como modelo.
- Existe una única unidad funcional completamente segmentada para las operaciones en punto flotante.

Las latencias en la unidad funcional son:

FP ALU op	→	FP ALU op	: 3 ciclos
FP ALU op	→	Store double	: 2 ciclos
Load double	→	FP ALU op	: 1 ciclos
Load double	→	Store double	: 0 ciclos

- Existe una unidad funcional independiente para las operaciones en enteros. Latencia de la unidad funcional de enteros (incluidas cargas) es 1 ciclo.
 - Se supone que en la etapa de retirado sólo se puede retirar 2 instrucciones cada vez (excepto saltos, debido al riesgo de retirar una instrucción mal especulada).
 - Se supone que el procesador cuenta con capacidad suficiente en la cola de instrucciones y estaciones de reserva para albergar todas las instrucciones en vuelo.
 - Se supone que la instrucción MULTD tiene asociada 3 ciclos en la etapa EX, y la instrucción ADDD, 2.
- A continuación se muestra el diagrama de ciclos donde la primera columna de la izquierda muestra las dos iteraciones del bucle y en las sucesivas columnas se mostrará el estado de las instrucciones en cada ciclo. Como se ve, se duplica el código dos veces dado que no existe ningún tipo de planificación estática por parte del compilador, y corresponde a la traza en tiempo de ejecución de dos iteraciones del código. Con el código completo se comienza el estudio de su evolución en el procesador:

1. LD F2, 0(R1):

Entra en el cauce en el ciclo 1 y al ser la primera instrucción avanza sin problema por las etapas del procesador. Avanza por las etapas de búsqueda, decodificación, envío a las estaciones de reserva y ejecución (ciclos 1, 2, 3, 4). Tras el ciclo 4 el dato producido puede usarse por la multiplicación en el ciclo siguiente que tuvo que detenerse 1 ciclo en la unidad funcional. En el ciclo 5 entra en WB y escribe el resultado en el búfer de reordenamiento. En el ciclo 6 se retira del búfer y se actualiza el registro F2 con el valor de la carga en el banco de registros.

2. MULTD F4, F2, F0:

Entra en el cauce en el ciclo 1 y avanza sin problema por las etapas IF, ID, IS en los ciclos consecutivos. Existe una dependencia de tipo verdadero con el registro F2 cargado en la instrucción anterior, pero el valor le es adelantado nada más terminar la ejecución de la carga (1 ciclo) y puede

comenzar su ejecución en el ciclo 5. La multiplicación necesita 3 ciclos (ciclos 5, 6, 7). Al final, la fase de ejecución en el ciclo 7 adelanta el dato a la suma posterior por lo que puede comenzar a ejecutarse en el ciclo 8. En el ciclo 8, la multiplicación escribirá en el búfer de reordenamiento, y en el ciclo 9 se retirará actualizando el banco de registros.

3. LD F6, 0(R2):

Entra en el cauce en el ciclo 2 y avanza sin problema por las etapas IF, ID, IS, EX en los ciclos consecutivos. En el ciclo 6 escribe en el búfer de reordenamiento (adelantándose a la multiplicación anterior) pero no se puede retirar hasta el ciclo 9, junto a la instrucción anterior (ejecución fuera de orden pero el retirado debe hacerse en orden hasta dos instrucciones en un mismo ciclo).

4. ADD F6, F4, F6:

Entra en el cauce en el ciclo 2 y avanza sin problema por las etapas IF, ID e IS. Existe una dependencia de tipo verdadero con la instrucción de carga anterior debido al registro F6, pero además tiene otra dependencia del mismo tipo con la multiplicación anterior a través del registro F4. Por ello, la ejecución de la suma debe retrasarse hasta el ciclo 8, justo después de que la multiplicación le adelante el valor. La suma permanece ejecutándose en la unidad funcional durante 2 ciclos (ciclos 8, 9). En el ciclo 10 escribe en el búfer de reordenamiento, y en el ciclo 11 se retira.

5. SD 0(R2), F6:

Entra en el cauce en el ciclo 3 y avanza sin problema por las etapas IF, ID e IS. En el ciclo 6 debe detenerse debido a la dependencia tipo RAW con la suma anterior (latencia de 2 ciclos). En el ciclo 10 le es adelantado el dato desde la salida de la unidad funcional, por lo que puede ejecutarse y almacenar el dato en memoria. Se han conservado las etapas WB y Ret, al no especificarse ningún comportamiento especial para esta instrucción aunque no genera datos. El mayor problema de esta convención es que ocupa el bus de datos común de forma improductiva e impidiendo que otra instrucción acceda a él.

6. ADDI R1, R1, #8:

Entra en el cauce en el ciclo 3 y avanza sin problema por las etapas IF, ID, IS, EX en los ciclos sucesivos. La suma en enteros tiene una latencia de 1 ciclo. En el ciclo 7 el dato está disponible para adelantarse a la instrucción SGTI posterior. En el ciclo 7 escribe en el búfer de reordenamiento a través el Bus de Datos Común que se encuentra libre en ese momento. Sin embargo, debe esperar hasta el ciclo 12 para ser retirada junto a la instrucción anterior.

7. ADDI R2, R2, #8:

Entra en el cauce en el ciclo 4 y procede de igual forma que la homónima instrucción precedente. De igual manera, termina su ejecución pero no puede escribir en el bus de datos común debido a que una instrucción anterior lo está ocupando. Debe esperar al ciclo 9 para escribir en el búfer de reordenamiento, y en el ciclo 13 es retirada.

8. SGTI R3, R1, done:

Entra en el cauce en el ciclo 5 (sólo puede entrar 1 operación en enteros en cada ciclo y por ello, no es emparejable con la anterior) y avanza por las etapas de IF, ID e IS sin problema en los ciclos consecutivos. Existe una dependencia verdadera RAW por el registro R1 con la instrucción de suma anterior, pero el dato se encuentra disponible cuando se requiere en la unidad funcional, por lo que no sufre ningún retraso ejecutándose en el ciclo 8. Igualmente, el salto siguiente que tiene una dependencia de tipo RAW a través del registro R3 puede comenzar a ejecutarse en el ciclo 9 dado que dispone del valor nada más finalizar la ejecución. El bus de datos común se encuentra

ocupado hasta que puede escribir en él en el ciclo 12 en el búfer de reordenamiento. El retirado se lleva a cabo en el ciclo 13.

9. BEQZ R3, foo:

Entra en el cauce en el ciclo 6 (se considera entera y no es emparejable con la anterior) y avanza por las etapas de IF, ID e IS sin problema en los ciclos consecutivos. Estando en la etapa IF, la BTB reconoce la instrucción y predice el salto correctamente (se supone predicción perfecta), y por ello, la instrucción destino del salto puede entrar en el cauce en el ciclo 7. Tiene una dependencia de tipo verdadero con la instrucción anterior, pero dispone del dato al entrar en EX en el ciclo 9. Las etapas de WB y Ret son necesarias dado que no se sabe si la especulación es correcta hasta ser retirada. En caso de no ser correcta habría que borrar las instrucciones posteriores. Por razones de seguridad las instrucciones de salto deben retirarse solas, ya que sería un error retirar otra instrucción a continuación que pudiera estar mal especulada.

Finalizada la primera iteración se procede a ejecutar la segunda. Se desenvuelve a lo largo del cauce de igual que la primera iteración. La única diferencia es la progresión del perceptible encadenamiento de accesos de escritura en el bus de datos común, que provoca que de ahí en adelante sólo se pueda retirar una instrucción por ciclo. La evolución completa puede verse en el diagrama siguiente:

Instrucciones / Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21			
1 LD F2,0(R1)	IF	ID	IS	EX	WB	Ret																		
2 MULTD F4,F2, F0	IF	ID	IS	EX!	EX	EX	EX	WB	Ret															
3 LD F6,0(R2)		IF	ID	IS	EX	WB	Ret!	Ret!	Ret															
4 ADD F6,F4,F6		IF	ID	IS	EX!	EX!	EX!	EX	EX	WB	Ret													
5 SD 0(R2),F6			IF	ID	IS	EX!	EX!	EX!	EX!	EX	WB	Ret												
6 ADDI R1,R1,#8			IF	ID	IS	EX	WB	Ret!	Ret!	Ret!	Ret!	Ret												
7 ADDI R2,R2,#8				IF	ID	IS	EX	WB!	WB	Ret!	Ret!	Ret!	Ret											
8 SGTI R3,R1,done					IF	ID	IS	EX	WB!	WB!	WB!	WB	Ret											
9 BEQZ R3,foo						IF	ID	IS	EX	WB!	WB!	WB!	WB	Ret										
1 LD F2,0(R1)							IF	ID	IS	EX	WB!	WB!	WB!	WB	Ret									
2 MULTD F4, F2, F0							IF	ID	IS	EX!	EX	EX	EX	WB!	WB	Ret								
3 LD F6,0(R2)								IF	ID	IS	EX	WB!	WB!	WB!	WB!	WB	Ret							
4 ADD F6,F4,F6								IF	ID	IS	EX!	EX!	EX!	EX	EX	WB!	WB	Ret						
5 SD 0(R2),F6									IF	ID	IS	EX!	EX!	EX!	EX!	EX	WB!	WB	Ret					
	1ª Iteración						2ª Iteración																	
Instrucciones / Ciclo	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25							
6 ADDI R1,R1,#8	IF	ID	IS	EX	WB!	WB!	WB!	WB!	WB!	WB!	WB	Ret												
7 ADDI R2,R2,#8		IF	ID	IS	EX	WB!	WB!	WB!	WB!	WB!	WB!	WB	Ret											
8 SGTI R3,R1,done			IF	ID	IS	EX	WB!	WB!	WB!	WB!	WB!	WB!	WB	Ret										
9 BEQZ R3,foo				IF	ID	IS	EX	WB!	WB!	WB!	WB!	WB!	WB!	WB	Ret									
1 LD F2,0(R1)					IF	ID	IS	...																
	... 2ª Iteración ->					Fin de la 2ª iteración																		

Como puede verse, entre la primera instrucción lanzada de la primera iteración y la primera de la segunda pasan 6 ciclos (del ciclo 1 al 6), con lo que se puede lanzar a ejecutar una iteración completa cada 6 ciclos. Se ve también que la última iteración tarda 11 ciclos en terminar la ejecución (del ciclo 13 al 23), con lo que la última iteración tiene una duración de 6+11=17 ciclos.

Un detalle a tener en cuenta es el ritmo de salida de instrucciones del cauce. Hemos concluido que podemos lanzar una iteración cada 6 ciclos. Si nos fijamos en el tiempo que transcurre entre el ciclo de la última instrucción en ejecutarse de la primera iteración y el ciclo de la última instrucción en ejecutarse de la segunda, podemos ver que son necesarios 9 ciclos para que una iteración abandone el cauce. Por lo

tanto podemos lanzar una iteración cada 6 ciclos, pero se necesitan 9 ciclos para que lo abandone. Esto provocará a la larga una saturación de los registros de la cola de instrucciones y las estaciones de reserva. Sin embargo, se ha supuesto que el procesador posee capacidad suficiente para albergar todas las instrucciones en vuelo

A continuación, se muestra el estado de las estaciones de reserva, búfer de reordenamiento y banco de registros en el ciclo 14, momento en que el salto de la segunda iteración se lanza a ejecutar.

Estaciones de reserva:

Estación de reserva	Ocupada?	Operación	V_j / Address	V_k	Q_j	Q_k
E1	Sí	ADDI	R1	#8		
E2	Sí	ADDI	R2	#8		
E3	Sí	SGTI	R1	done	(E1 ya adelantado)	
E4	Sí	BEQZ	R3	foo		
E5, ...	No					
FP1	Sí	MULTD	F2	F0		
FP2	Sí	ADD	F4	F6	(FP1 ya adelantado)	(L1 ya adelantado)
L1	No					
L2	Si	LD	0+R2			
S1	Sí	SD	F6		FP2	

Estado del Banco de registros:

	0	1	2	3	4	5	6	...
R		E1	E2	E3				
F			BR1		FP1		FP2	

Búfer de Reordenamiento:

<<	BR1	BR2	BR3	BR4	BR5	BR6	BR7 ...	<<
	$F2 = L1 = M(0+R1)$	$F4=FP1$	$F6=L1$	$F6=FP2$	$R1=E1$	$R2=E2$	$R3=E3$	
	OK	NOK	NOK	NOK	NOK	NOK	NOK	

Problema 2-3-4. Suponer que se dispone de un procesador altamente segmentado para el que se implementa un "Branch Target Buffer (BTB)" sólo para los saltos condicionales. Suponer que la penalización por fallo en la predicción del salto es siempre de 4 ciclos y la penalización por fallo del búfer es siempre de 2 ciclos. Suponer que se tiene una tasa de acierto del 80% y una precisión del 90%, y una frecuencia de instrucciones de saltos condicionales del 20%. ¿Cuánto más rápido es el procesador que dispone de un búfer BTB en relación a otro procesador sin BTB que tiene una penalización fija de ciclos en los saltos? Suponer que se produce una penalización de 3 ciclos de reloj por dependencias de control en el procesador que no tiene BTB. Suponer un CPI ideal sin paradas por saltos de 1. (Respuesta: 40%)

Solución.

BTB (Branch Target Buffer) hace referencia a una técnica para emplear en la predicción del comportamiento de instrucciones de saltos en procesadores que introducen planificación dinámica de saltos. El principal objetivo de la técnica BTB consiste en conseguir que la penalización por dependencias de control sea lo menor posible. Esta técnica consigue no introducir ningún ciclo de penalización cuando se acierta en la BTB y la predicción es correcta.

Suponiendo que:

$CPI_{ideal} = 1$, comparamos las dos máquinas (con BTB y sin BTB) con una máquina ideal.

Con BTB

$CPI_{BTB} = 1 + \text{penalización por predicción de salto} + \text{penalización por predicción del búfer BTB}$.

Salto condicional que aciertan en la BTB pero fallan en la predicción corresponden al 10% de los saltos (Precisión= 90%) del 80% de los saltos que corresponden al 20% de las instrucciones. En estos casos, siempre se introducen 4 ciclos de penalización. Por lo tanto, la penalización por instrucción es $= 0.1 \times 0.8 \times 4 = 0.064$ ciclos por instrucción

Salto condicional que fallan en la BTB corresponden al 20% (1-80%). Las instrucciones de salto corresponden al 20% de las instrucciones, y siempre introducen 2 ciclos de penalización. Por lo tanto, la penalización por instrucción es $= 0.2 \times 0.2 \times 2 = 0.08$ ciclos por instrucción

$CPI_{BTB} = 1 + 0.064 + 0.08 = 1.144$ ciclos.

Sin BTB

$CPI_{sin-BTB} = 1 + \text{penalización por dependencias de control}$.

Salto condicional que se engloban en las dependencias de control: corresponde al 20% de las instrucciones, y siempre introducen 3 ciclos de penalización; penalización $= 0.2 \times 3$

$CPI_{sin-BTB} = 1 + 3 \times 0.2 = 1.6$ ciclos.

El tiempo de ejecución del programa en un procesador con BTB es un $28.5\% = 100 \times (CPI_{sin-BTB} - CPI_{BTB}) / CPI_{sin-BTB}$ más largo que el tiempo de ejecución en una máquina sin BTB.

Problema 2-3-5 (Examen Septiembre 2007, 14/9/2007). Suponer que se dispone de un procesador parecido a DLX32p cuya microarquitectura es de 5 etapas de segmentación (IF, ID, EX, MEM, WB) y permite una sola instrucción durante cada ciclo de reloj en cada una de las etapas de segmentación. Este procesador ejecuta el siguiente código:

```
Bucle:    LW    R3, 0(R0)
          LW    R1, 0(R3)
          ADDI  R1, R1, #1
          SUB   R4, R3, R2
          SW    R1, 0(R3)
          BNZ   R4, Bucle
```

Todas las operaciones se realizan en 1 ciclo de reloj excepto LW y SW, las cuales tardan 1 + 2 ciclos en MEM (1 ciclo para el envío de la dirección al controlador de cache, y 2 ciclos para acceder a la cache de datos), y los saltos condicionales, los cuales tardan 1 + 1 ciclos en ID (1er ciclo para decodificar los campos del formato de la instrucción, y 2º ciclo para leer el banco de registros, comprobar que la condición de salto se cumple y generar la dirección destino del salto). No existe anticipación, y el banco de registros puede ser leído y escrito en el mismo ciclo de reloj.

1. Mostrar en cada ciclo de reloj la etapa de segmentación en la que se encuentra cada instrucción durante una iteración del bucle. Suponer que no existe ningún hardware para realizar predicción de saltos condicionales.
2. ¿Cuántos ciclos de reloj se pierden en cada iteración debidos a las penalizaciones del salto condicional? Justifica la respuesta.
3. Suponer que ahora se dispone de una predictor estático de saltos condicionales que es capaz de reconocer en la etapa ID si los saltos condicionales lo hacen hacia atrás. En este caso, predice que el salto es tomado. ¿Cuántos ciclos de reloj se pierden debido a las penalizaciones del salto condicional? Justifica la respuesta.
4. Suponer que ahora se dispone de una predictor dinámico de saltos condicionales en la etapa IF del procesador. En este caso, ¿cuántos ciclos de reloj se pierden cuando la predicción es correcta? Justifica la respuesta.

Solución.

1.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	LW R3, 0(R0)	IF	ID	EX	M	M	M	WB													
	LW R1, 0(R3)		IF	ID!	ID!	ID!	ID!	ID	EX	M	M	M	WB								
	ADDI R1,R1,#1			IF!	IF!	IF!	IF!	IF	ID!	ID!	ID!	ID!	ID	EX	M	WB					
	SUB R4, R3, R2							IF!	IF!	IF!	IF!	IF!	IF	ID	EX	M	WB				
	SW R1, 0(R3)													IF	ID!	ID	EX	M	M	M	WB
	BNZ R4, Bucle														IF!	IF	ID	ID	EX	M	WB
(1,2)	LW R3, 0(R0)																2 cicl	IF	ID	EX	
(3)	LW R3, 0(R0)																1 c	IF	ID	EX	M
(4)	LW R3, 0(R0)																	IF	ID	EX	M

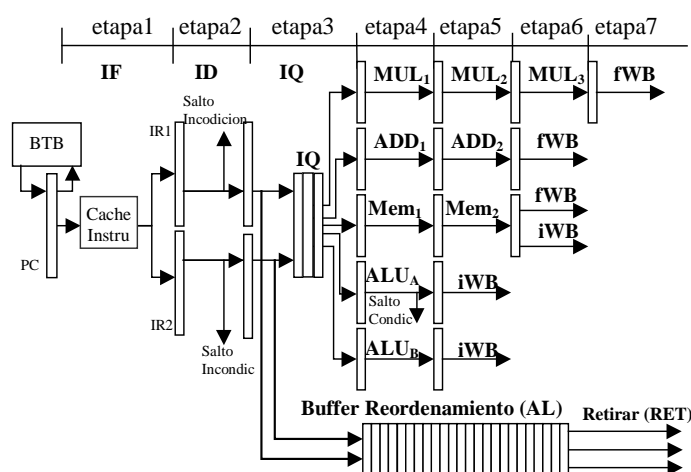
2. 2 ciclos

3. 1 ciclo

4. ningún ciclo

TEMA 2-4: Procesadores Superescalares

Problema 2-4-1 (examen Febrero 2004). Procesador Superescalar con Ejecución Fuera de Orden. En la figura se muestra el esquema de un procesador **superescalar** con ejecución fuera de orden y renombramiento de registros. Se desea ejecutar el programa que aparece a la derecha.



Programa

```

1. R0 ← MEM(R1)
2. R1 ← R1+8
3. go to 10
4. F2 ← MEM(R1)
5. F3 ← MEM(R1+8)
6. F2 ← F2·F3
7. F5 ← F2+F5
8. R1 ← R1+16
9. R0 ← R0-1
10. if (R0>0) go to 4
11. if (R0==0) go to 1
12. R1 ← R1+10
  
```

La tabla siguiente muestra qué unidad funcional es utilizada por cada tipo de instrucción, en qué etapa se resuelve la instrucción y cuál es su latencia efectiva. Todas las instrucciones tienen que esperar a tener todos sus operandos disponibles (en un registro o por anticipación) antes de comenzar su ejecución. Las unidades funcionales están segmentadas, así que varias instrucciones pueden estar en ejecución simultáneamente en la misma unidad funcional (en diferentes etapas de su ejecución). Fijarse en que sólo la ALU_A puede ejecutar instrucciones de salto condicional.

Semántica		Resolución	W-back	Latencia
Salto incondicional	go to N	ID	-	-
Salto condicional	if cc(RX) go to N	ALU_A	-	1
Operación Entera	$R_z \leftarrow R_x \text{ op } R_y$	ALU_A, ALU_B	iWB	1
Carga desde Memoria a Rx	$R_x \leftarrow \text{MEM}(R_y)$	Mem_2	iWB	2
Carga desde Memoria a Fx	$F_x \leftarrow \text{MEM}(R_y)$	Mem_2	fWB	2
Almacenamiento en Memoria desde Rx	$\text{MEM}(R_y) \leftarrow R_x$	Mem_2	-	-
Almacenamiento en Memoria desde Fx	$\text{MEM}(R_y) \leftarrow F_x$	Mem_2	-	-
Suma punto-flotante	$F_z \leftarrow F_x + F_y$	ADD_2	fWB	2
Multiplicación punto-flotante	$F_z \leftarrow F_x \cdot F_y$	MUL_3	fWB	3

La **etapa IF** predice saltos usando una tabla BTB. En cada acceso a la cache de instrucciones se leen dos instrucciones almacenadas en direcciones de memoria consecutivas. En paralelo, se accede a la BTB utilizando el contenido del registro PC como índice. De esta forma, se obtiene la dirección de la siguiente instrucción (el nuevo contenido del PC), tanto para instrucciones de salto como cualquier otra. Supondremos que la BTB acierta siempre su predicción, incluso para los saltos condicionales y para los saltos que se decodifican por primera vez (predicción perfecta).

En la **etapa ID** se realiza lo siguiente: (1) decodifica hasta 2 instrucciones por ciclo, (2) renombrar los registros en estaciones de reserva, y (3) bloquea las instrucciones sólo si la cola de instrucciones (IQ) o la lista de instrucciones activa en el Búfer de Reordenamiento (AL) están llenas o faltan registros de renombrado. Si la 1ª instrucción de la pareja se ha de bloquear, la 2ª también se bloquea, excepto si es un salto incondicional, que se puede ejecutar fuera de orden. Si sólo se necesita bloquear la 2ª instrucción, ésta se copia de IR2 a IR1, y en el siguiente ciclo la etapa IF podrá enviar una sola instrucción a IR2.

La **etapa IQ** se usa para lo siguiente: (1) copiar las instrucciones recién decodificadas y renombradas a la cola de instrucciones IQ (con espacio para tres instrucciones), (2) copiar las instrucciones al Búfer de Reordenamiento (AL) (con espacio para quince instrucciones), (3) comprobar dependencias RAW, (4) comprobar si existen unidades funcionales libres, (5) comprobar si todos los operandos están disponibles en registros o por anticipación, (6) leer operandos fuente de los bancos de registros y (7) enviar a ejecutar hasta tres instrucciones a las unidades funcionales. Las instrucciones se empiezan a ejecutar fuera de orden, dando siempre prioridad a las más antiguas.

En la etapa “Ret” las instrucciones se retiran en orden del Búfer de Reordenamiento (AL). Puede existir cualquier número de instrucciones retiradas por ciclo. La retirada de instrucciones puede ocurrir como muy pronto en el ciclo siguiente después de finalizar todas sus tareas.

Hay dos bancos de registros: uno de enteros (registros R0 á R31) y otro de punto flotante (registros F0 á F31). Se permiten seis lecturas y dos escrituras simultáneas por cada banco de registros. Si se detecta que habrá más de dos escrituras en la correspondiente etapa Ret, la instrucción menos antigua se frena en la etapa IQ. A la vez que se escribe un resultado en el Búfer de Reordenamiento (etapa WB), el dato se puede enviar por anticipación a la entrada de cualquier unidad funcional.

	1	2	3	4	5	6	7
IF (2)	1,2	3,4	10,11		...		
ID (2)		1,2	3, -	10, -	...		
IQ (3)			1,2		10	...	
AL (12)			1,2	1,2,3	1,2,3, 10	1,2,3 10,
Add₁							
Add₂							
Mul₁							
Mul₂							
Mul₃							
Addr				1			
Mem					1		
ALU_A				2		10	
ALU_B							
fWB (2)							
iWB (2)					2	1	

CICLOS	1	2	3	4	5	6	7	8
1. R0 ← MEM(R1)	IF	ID	IQ	M1	M2	iW	Ret	
2. R1 ← R1+8	IF	ID	IQ	Aa	iW	-	Ret	
3. goto 10		IF	ID	-	-	-	Ret	
4. F2 ← MEM(R1)		IF						
10.if R0>0 goto 4			IF	ID	IQ	Aa	Ret	
11.if R0==0 goto 1			IF					

Supondremos que: (1) las dependencias de datos con posiciones de memoria (almacenamientos escribiendo y/o cargas leyendo datos de la misma posición de memoria) no generan conflictos ni retardos, (2) que no hay fallos de cache (caches de datos e instrucciones perfecta), (3) que existe un número ilimitado de registros físicos para renombrar, (4) que la cache de datos tiene un solo puerto.

Se ejecuta el fragmento de código de la página anterior, donde la secuencia de instrucciones ejecutadas es: 1, 2, 3, 10, 4, 5, 6, 7, 8, 9, 10, 4, 5, 6, 7, 8, 9, 10, 11, 1, 2, 3, 10, 11, 12. La tabla de reserva de la ejecución se muestra a la derecha de este párrafo. Suponemos que las instrucciones 1 y 2 entran juntas en la ruta de datos al comenzar la ejecución del programa. Fijarse en la fila AL de la tabla, donde se muestran en negrita las instrucciones ya finalizadas. En el ciclo 7, las instrucciones 1, 2, 3 y 10 se retiran de AL y de la ruta de datos. En la tabla de evolución temporal que se muestra debajo se muestra el comportamiento de los primeros ciclos.

Ayuda: No hace falta hacer la tabla para un millón de iteraciones del bucle (☺). Es necesario, en cambio, hacer la tabla para 2, 3 o más iteraciones, hasta que se encuentre una pauta de repetición que nos permita saber cada cuántos ciclos se repite el mismo estado (la misma columna), y cuántas iteraciones (instrucciones de la 4 a la 10) se ejecutan (entran en la ruta de datos) en esos ciclos. A partir de aquí se calcula el número de ciclos necesarios para que entre una nueva iteración en la ruta de datos. Y luego, el número de ciclos necesario para que entre una nueva instrucción en la ruta de datos.

1. Completar la tabla de reserva y la tabla de evolución temporal de las instrucciones hasta que la última instrucción (instrucción 12) completa la última etapa de ejecución (RET). Indicar el uso de caminos de anticipación (mediante una flecha entre las etapas de la ruta de datos que realizan la anticipación).

Solución.

TABLA DE RESERVA

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
IF (2)	1, 2	3, 4	10, 11	4, 5	6, 7	8, 9	10, 11	4, 5	5, 6	7, 8	9, 10	10, 11	11, 1	2, 3	3, 10	10, 11	12, -	-	-	-	-
ID (2)		1, 2	3, -	10, -	4, 5	6, 7	8, 9	9, 10	10, 4	5, 6	7, 8	8, 9	9, 10	11, 1	1, 2	2, 3	10, 11	12, -	-	-	-
IQ (3)			1, 2		10	4, 5	5, 6, 7	6, 7, 8	6, 7, 9	7, 10, 4	7, 5, 6	7, 6, 7	6, 7, 8	7, 9, 10	7, 10, 11	7, 11, 1	2	10, 11	11, 12	-	-
AL (12)			1, 2	1, 2, 3	1, 2, 3, 10	1, 2, 3, 10, 4, 5	4, 5, 6, 7	4, 5, 6, 7, 8	4, 5, 6, 7, 8, 9	5, 6, 7, 8, 9, 10, 4	6, 7, 8, 9, 10, 4, 5, 6	6, 7, 8, 9, 10, 4, 5, 6, 7	6, 7, 8, 9, 10, 4, 5, 6, 7, 8	7, 8, 9, 10, 4, 5, 6, 7, 8, 9, 10	7, 8, 9, 10, 4, 5, 6, 7, 8, 9, 10, 11	5, 6, 7, 8, 9, 10, 11, 1	5, 6, 7, 8, 9, 10, 11, 1, 2, 3	7, 8, 9, 10, 11, 1, 2, 3, 10, 11	7, 8, 9, 10, 11, 1, 2, 3, 10, 11, 12	11, 12	12
Add ₁													7				7				
Add ₂														7				7			
Mul ₁										6				6							
Mul ₂											6				6						
Mul ₃												6				6					
Mem ₁				1			4	5			4	5					1				
Mem ₂					1			4	5			4	5					1			
ALU _A				2		10			8	→	10			8	→	10	11	2	10	11	
ALU _B																				12	
fWB (2)								4	5				4, 6	5	7		6		7		
iWB (2)					2	1				8	9				8				1, 2		12

Traza Dinámica: 1, 2, 3, 10, 4, 5, 6, 7, 8, 9, 10, 4, 5, 6, 7, 8, 9, 10, 11, 1, 2, 3, 10, 11, 12.

**TABLA DE EVOLUCIÓN
TEMPORAL**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1. R0 ← MEM(R1)	IF	ID	IQ	M1	M2	iW	Ret															
2. R1 ← R1+8	IF	ID	IQ	Aa	iW	-	Ret															
3. go to 10		IF	ID	-	-	-	Ret															
4. F2 ← MEM(R1)		IF																				
10.if R0>0 go to 4			IF	ID	IQ	Aa	Ret															
11.if R0==0 go to 1			IF																			
4. F2 ← MEM(R1)				IF	ID	IQ	M1	M2	fW	Ret												
5. F3 ← MEM(R1+8)				IF	ID	IQ	IQ!	M1	M2	fW	Ret											
6. F2 ← F2·F3					IF	ID	IQ!	IQ!	IQ!	IQ!	IQ!	ML	ML	ML	fW	Ret						
7. F5 ← F2+F5					IF	ID	IQ!	IQ!	IQ!	IQ!	IQ!	FA	FA	FA	fW	Ret						
8. R1 ← R1+16						IF	ID	IQ	Aa	iW	-	-	-	-	-	Ret						
9. R0 ← R0-1						IF	ID!	ID	IQ	Aa	iW	-	-	-	-	Ret						
10.if R0>0 go to 4						IF	ID!	ID	IQ!	Aa	-	-	-	-	-	Ret						
11.if R0==0 go to 1						IF																
4. F2 ← MEM(R1)								IF	ID	IQ	M1	M2	fW	-	-	Ret						
5. F3 ← MEM(R1+8)								IF!	IF	ID	IQ	M1	M2	fW	-	-	Ret					
6. F2 ← F2·F3									IF	ID	IQ!	IQ!	IQ!	IQ!	IQ!	FA	FA	fW	Ret			
7. F5 ← F2+F5										IF	ID!	ID	IQ	Aa	iW	-	-	-	-	Ret		
8. R1 ← R1+16											IF	ID!	ID	IQ	Aa	iW	-	-	-	-	Ret	
9. R0 ← R0-1												IF	ID!	ID	IQ	Aa	iW	-	-	-	-	Ret
10.if R0>0 go to 4												IF!	IF	ID	IQ!	IQ!	Aa	-	-	-	-	Ret
11.if R0==0 go to 1												IF!	IF	ID	IQ!	IQ!	Aa	-	-	-	-	Ret
12.R1 ← R1+10												IF										
1. R0 ← MEM(R1)													IF	ID!	ID	IQ	M1	M2	iW	-	Ret	
2. R1 ← R1+8													IF	ID!	ID	IQ	Aa	iW	-	-	Ret	
3. go to 10													IF!	IF	ID	-	-	-	-	-	Ret	
4. F2 ← MEM(R1)																						
10.if R0>0 go to 4																IF!	IF	ID	IQ	Aa	-	Ret
11.if R0==0 go to 1																IF	ID	IQ	IQ	Aa	Ret	
12.R1 ← R1+10																IF	ID	IQ	Ab	iW	Ret	

Suponer ahora que en el programa anterior, las instrucciones del bucle interno se ejecutan 1000 veces por cada ejecución del bucle externo, y que el bucle externo se ejecuta 1000 veces. Ello supondría un total de 7,005,001 instrucciones (instrucciones bucle interno: 7 [4,...,10], instrucciones bucle externo: 1 [11], última instrucción: 1 [12]). Para simplificar el cálculo, ignoraremos la ejecución de las instrucciones fuera del bucle interno (que representan menos del 0,1%), y supondremos que sólo se ejecuta el bucle interno 1 millón de veces (es decir, se ejecutan 7,000,000 instrucciones).

2. Calcular el tiempo de ejecución y el IPC.

Solución.

Considerando sólo 1,000,000 iteraciones internas:

Ciclos = 1,000,000 iteraciones × 4 ciclos/iteración = 4,000,000 ciclos

IPC = 7,000,000 / 4,000,000 = 1.75

Ya que hay 7 instrucciones por iteración interna, el procesador superescalar podría ejecutar cada iteración en 3.5 ciclos. Como realmente se ejecuta en 4 ciclos, Penalización = 4 – 3.5 = 0.5 ciclos

3. ¿Cuál sería la penalización, en ciclos por iteración del bucle interno si la predicción de la instrucción 10 (salto condicional) siempre fallase?

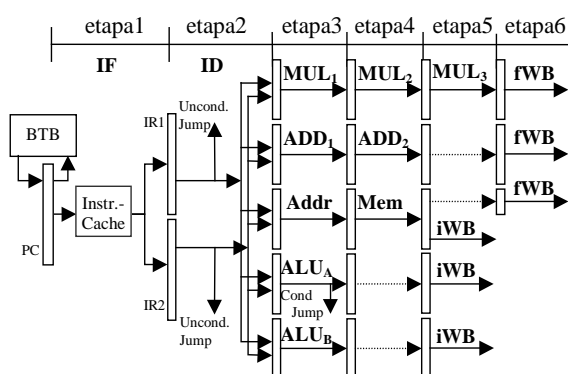
Solución.

La instrucción de salto condicional se resuelve en la etapa ALU_A , y se observa en el diagrama de ejecución temporal que la instrucción 10 llega a la etapa ALU_A 4 ciclos después de entrar la instrucción. Por ello, el número de ciclos por iteración interna se eleva a 8 ($4 + 4$). Esto hace que,

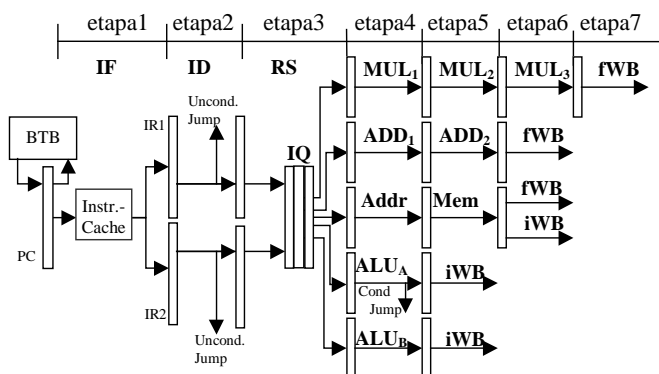
Penalización = $8 - 3.5 = 4.5$ ciclos.

Problema 2-4-2 (examen febrero 2005). En la figura se muestra el esquema de dos **procesadores superescalares** con diferentes políticas de lanzamiento de instrucciones. La etapa IF es idéntica en ambos y realiza predicción de hasta 2 instrucciones de salto por ciclo usando una tabla BTB. La cache de instrucciones dispone de doble puerto, lo cual permite acceder a dos instrucciones por ciclo en direcciones de memoria cualquiera. La etapa ID decodifica hasta 2 instrucciones por ciclo y puede bloquear una instrucción en las siguientes situaciones:

- **Procesador con Ejecución en Orden:** si hay dependencia RAW con una instrucción que aún no ha generado su resultado o la unidad funcional que se necesita está ocupada.
- **Procesador con Ejecución Fuera de Orden:** si la cola de instrucciones IQ está llena.



Procesador con Ejecución-en-Orden



Procesador con Ejecución-Fuera-de-Orden

Instrucción	Resol.	WB	Lat
BR label	ID	-	-
Bcc Rx, label	ALU _A	-	-
OP Rx,Ry,Rz	ALU _A / ALU _B	iWB	1
LDQ Rx,off(Ry)	Mem	iWB	2
LDF Fx,off(Ry)	Mem	fWB	2
STQ Rx, off(Ry)	Mem	-	-
STF Fx, off(Ry)	Mem	-	-
ADDF Fx,Fy,Fz	ADD ₂	fWB	2
MULF Fx,Fy,Fz	MUL ₃	fWB	3

1. LDQ R0,(R1)	R0 ← MEM(R1)
2. ADDQI R1,8,R1	R1 ← R1+8
3. BR +28	salta a instr. 11
4. LDF F2,(R1)	F2 ← MEM(R1)
5. LDF F3,8(R1)	F3 ← MEM(R1+8)
6. MULF F2,F3,F2	F2 ← F2·F3
7. BFLE F2, +4	if F2≤0 salta a 9
8. ADDF F2,F5,F5	F5 ← F2+F5
9. ADDQI R1,16,R1	R1 ← R1+16
10. ADDQI R0,-1,R0	R0 ← R0-1
11. BGT R0,-32	si R0>0 salta a 4
12. BEQ R0,-48	if R0==0 salta a 1

En ambos casos, si la 1ª instrucción se ha de bloquear, la 2ª también se bloquea. Si sólo se ha de bloquear la 2ª instrucción, ésta se copia de IR2 a IR1, y en el siguiente ciclo la etapa IF podrá enviar una instrucción a IR2.

En el procesador *fuera de orden*, la etapa RS se utiliza para: (1) copiar las instrucciones recién decodificadas en una entrada de la cola de instrucciones IQ (con **4 entradas**), (2) comprobar dependencias RAW, (3) obtener unidades funcionales libres, (4) leer operandos fuente de los bancos de registros, y (5) enviar **hasta 4** instrucciones listas a las unidades funcionales. Las instrucciones pueden comenzar a ejecutarse en cualquier orden, dando siempre prioridad a las más antiguas.

En ambos procesadores se pueden realizar **cuatro lecturas** y **dos escrituras** simultáneas a cada banco de registros (tipo entero y tipo punto flotante). En el procesador *en orden*, la **actualización** de registros enteros se realiza siempre en la 5ª etapa, iWB, y la de registros de punto flotante se realiza siempre en la 6ª etapa, fWB. En el procesador *en orden* **existen todos** los caminos de **anticipación** posibles. En el procesador fuera de orden pueden realizar anticipación las (hasta 4) instrucciones que van a

realizar la escritura en un banco de registros. Si hay conflicto en la escritura de resultados, la instrucción menos antigua se bloquea.

La tabla que aparece más arriba muestra qué unidad funcional es utilizada por cada tipo de instrucción, en qué etapa se resuelve la instrucción y cuál es su latencia efectiva. Todas las instrucciones tienen que esperar a tener todos sus operandos disponibles (en un registro o por anticipación) antes de comenzar su ejecución. Supondremos que el desambigüamiento de memoria se resuelve sin problema, que no hay fallos de cache (datos e instrucciones), que la predicción de saltos es ideal, y que existe un número ilimitado de estaciones de reserva.

Se ejecuta el fragmento de código siguiente, donde la secuencia de instrucciones ejecutadas es: 1, 2, 3, 11, 4, 5, 6, **7, 9**, 10, 11, 4, 5, 6, 7, **8**, 9, 10, 11, 12.

Suponer que la tabla BTB permite almacenar junto con la dirección PC destino del salto la primera instrucción destino del salto.

- 1. Rellenar la tabla de reserva de recursos para ambos procesadores. Suponer que al comenzar la ejecución del programa las instrucciones 1 y 2 entran juntas en el procesador. Calcular el IPC.**
- 2. Indicar en cada caso el uso de caminos de anticipación con una flecha entre las etapas de la microarquitectura que realizan la anticipación.**

Solución:

Procesador con Ejecución en Orden

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
IF (2)	1, 2	3, 11	4, 5!	5!, 6!	5!, 6!	6, 7!	7, 9!	9!, 10!	9!, 10!	9!, 10!	9, 10!	10, 11!	11, 4!	4!, 5!	4, 5!	5, 6	7, 8!	8!, 9!
ID (2)		1, 2	3, 11!	4, 11!	4!, 11	4, 5!	5, 6!	6!, 7!	6!, 7!	6!, 7!	6, 7!	7!, 9	7!, 10	7!, 11!	7, 11!	4, 11	5, 6	6!, 7!
Add₁																		
Add₂																		
fWB																		
Mul₁																		
Mul₂													6					
Mul₃														6				
fWB															6			
Addr			1				4	5									4	5
Mem				1				4	5									4
fWB										4	5							
iWB					1				4	5								
ALU_A			2				11						9				7	11
iWB				2!	2!	2								9!	9			
ALU_B														10				
iWB															10!	10		

Suponemos predicción perfecta de los saltos: BFLE (instrucción 9) en ciclo 11

Instrucciones 3 en ciclo 4, 9 en ciclo 14, y 10 en ciclo 15 atraviesan una etapa de segmentación de sincronización equivalente a Mem

Instrucción 2 se para en ciclo 5 porque el bus iWB está ocupado con la instrucción 1

	19	20	21	22	23	24	25	26	27	28	29	30	31	32			
IF (2)	8!,9!	8!,9!	8!,9!	9!,10!	9!,10!	9!,10!	9,10	11,12!	12								
ID (2)	6!,7!	6!,7!	6,7!	7!,8!	7!,8!	7!,8!	7,8	9,10!	10,11!	11!,12!	11,12!	12!	12				
Add₁								8									
Add₂									8								
fWB										8							
Mul₁				6													
Mul₂					6												
Mul₃						6											
fWB							6										
Addr																	
Mem	5																
fWB		4	5														
iWB	4	5															
ALU_A								7	9	10			11	12			
iWB										9!	9, 10!	10					
ALU_B																	
iWB																	

El número de instrucciones ejecutadas, coincide con el número de instrucciones de la traza dada en el enunciado, que son 20, y el número de ciclos es 32, por lo tanto: $IPC : 20 \text{ instrucciones} / 32 \text{ ciclos} = 0.625$

Procesador con Ejecución Fuera de Orden

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
IF (2)	1,2	3,11	4,5	6,7	9,10	11,4	5,6	7,8	7,8	9,10	11,12	11,12	11,12	-						
ID (2)		1,2	3,11	4,5	6,7	9,10	11,4	5,6	5,6	7,8	9,10	9,10	9,10	11,12						
RS (4)			1,2	11!	11!,4, 5!	11,5, 6!,7!	9, 10, 6!, 7!	4!, 11!, 6!, 7!	11, 4, 6!, 7!	5, 6!, 7!	7!,7!, 6!, 8!	7!,7!,2, 6, 8	7!,7!,2, 6, 8	9,10,7!, 8!	11, 12!, 7!,8!	12!, 7	12			
Mul₁										6				6						
Mul₂											6				6					
Mul₃												6				6				
fWB													6				6			
Addr				1		4	5			4	5									
Mem					1		4	5			4	5								
i/fWB						1		4	5			4	5							
ALU_A				2			11	9		11				7	10		11	7	12	
iWB					2				9							10				
ALU_B								10							9					
iWB									10							9				
ADD1																		8		
ADD2																			8	
fWB																				8

Hasta el ciclo 20, se realiza una carga de trabajo efectivo en el procesador. El número de instrucciones ejecutadas, coincide con el número de instrucciones de la traza dada en el enunciado, que son 20, por lo tanto:

$$\text{IPC} : 20 \text{ instrucciones} / 20 \text{ ciclos} = 1$$

Se estudian alternativas para mejorar las prestaciones del procesador fuera de orden en el programa anterior:

3. Suponiendo que se realizaran muchas iteraciones del bucle interno y que la instrucción nº 7 alternara sus resultados como en el ejemplo (la primera vez salta, la segunda no salta, la tercera salta, etc.) ¿cuál sería el IPC si se aumentara a 100 el tamaño de la cola IQ?

Solución.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
IF (2)	1,2	3,11	4,5	6,7	9,10	11,4	5,6	7,8	9,10	11,4	5,6	7,9	10,11	4,5	6,7	8,9	10,11	12	
ID (2)		1,2	3,11	4,5	6,7	9,10	11,4	5,6	7,8	9,10	11,4	5,6	7,9	10,11	4,5	6,7	8,9	10,11	12
RS (100)			1,2	11!	11!,4, 5!	11,5, 6!,7!	9, 10, 6!, 7!	7!, 11!, 6!, 4!	7!,11, 4, 5,6!, 6!²	7!¹, 6!², 8!, 7!²	7!¹,7!² ,9, 10, 6!², 8!	7!¹,7!², 6!², 8,11!,4!	4, 7!¹, 5, 6!³, 11, 8,6²,7!²	6!³,5,11, 7!³,9,7!²	11!⁴,10, 6!³, 7!³, 7!²	4,5,11!⁴, 6!³,5, 7!³,7!²	6!³,5,11!⁴, ,6!⁴,5, 7!³,7!²,7⁴	8,9,11!⁴, 6!⁴, 7!³,7!⁴	8,11!⁵, 6!⁴,10 7!³,7!⁴
Mul₁										6¹				6²				6³	
Mul₂											6¹				6²				6³
Mul₃												6¹				6²			
fWB																			
Addr				1		4	5			4	5			4	5		4	5	
Mem					1		4	5			4	5			4	5		4	5
i/fWB						1		4	5			4	5			4	5		4
ALU_A				2			11	9		11		9		7¹	11		7²		11
iWB					2				9				9						
ALU_B								10				10			9	10			9
iWB									10				10			9	10		
Add1														8					
Add2															8				
fWB																8			

	20	21	22	23	24	25	26	
IF (2)								
ID (2)								
RS (100)	12, 8, 11 ¹⁵ , 6 ¹⁴ , 7 ¹³ , 7 ¹⁴	12, 8, 11 ¹⁵ , 6 ¹⁴ , 7 ¹⁴	12, 11 ¹⁵ , 7 ¹⁴	12, 7 ¹⁴	7 ¹⁴	7 ¹⁴		
Mul₁			6 ⁴					
Mul₂				6 ⁴				
Mul₃	6 ³				6 ⁴			
fWB		6 ³				6 ⁴		
Addr								
Mem								
i/fWB	5							
ALU_A			7 ³	11 ⁵	12		7 ⁴	
iWB								
ALU_B	10							
iWB	9	10						
Add1			8					
Add2				8				
fWB					8			

Se itera 4 veces, alternando los saltos de la instrucción 7. Se observa que la ocupación máxima es de 8 instrucciones en espera de ser lanzadas (ciclo 13). Por tanto, no se llegará a ocupar la cola IQ. Considerando que la primera iteración del bucle interno comienza en la instrucción 4, tarda 3 ciclos: ciclos 3-5. La siguiente iteración en la que se ejecuta la instrucción 8, tarda 4 ciclos: 6-9. La siguiente iteración tarda 4 ciclos: 10-13, y la última, 13 ciclos: 14-26. Se han ejecutado 35 instrucciones, por lo tanto: $IPC = 35 \text{ instrucciones} / 26 \text{ ciclos} = 1.35$.

4. Con el mismo programa que antes y con la cola IQ de tamaño 100, si pudieras añadir una nueva unidad funcional, ¿de qué tipo sería? Razónalo.

Solución:

Observar los ciclos 6-7, 10-11, 17-18; en todos ellos existe un ciclo de parada de la segunda instrucción de carga debido al riesgo estructural de que existe un único puerto de acceso a memoria, por lo que la instrucción 5 debe esperarse un ciclo mientras la instrucción 4 está utilizando la primera etapa de la unidad funcional de carga/almacenamiento. Por lo tanto, añadiría una unidad funcional de memoria o equivalentemente un segundo puerto de acceso a la memoria. Con esta mejora se adelantaría la ejecución de la instrucción 6 en un ciclo de reloj, la cual se ejecuta una vez en cada iteración del bucle interno del código.

5. ¿Cuánto penalizaría un error en la predicción del salto para la instrucción nº 7? ¿Penaliza lo mismo tanto si el salto se toma como que no se toma?

Solución:

En el ciclo 4 se introduce por primera vez el salto que corresponde a la instrucción 7. Hasta el ciclo 14 no se resuelve el salto. Por lo tanto, han transcurrido 10 ciclos de reloj, los cuales corresponde a los ciclos de penalización si existiera un fallo de predicción. Para la segunda vez que se ejecuta la instrucción 7, se

introduce en el ciclo 8 y se resuelve en el 18. Para la tercera vez que se ejecuta la instrucción, los ciclos desde que entra hasta que se resuelve son 10 también: 12-22. Para la cuarta vez la penalización es de 11 ciclos: 15-26. Por lo tanto, la penalización no es la misma cuando el salto se toma que cuando no se toma.

6. Si se aumentase la capacidad de las etapas IF e ID a 4 instrucciones/ciclo, con una cola IQ suficientemente grande y suficientes unidades funcionales y puertos de lectura/escritura en los bancos de registro, ¿cuál sería aproximadamente el CPI?

Solución:

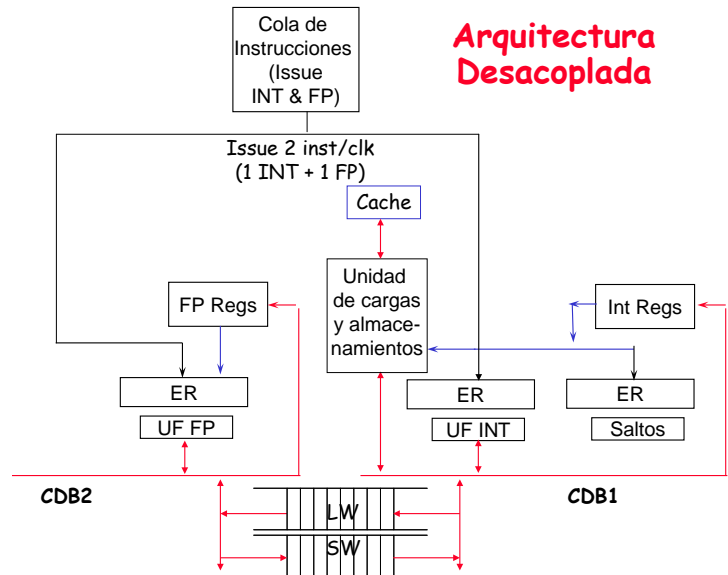
Para calcular esto, analizamos la ruta crítica de instrucciones. Si analizamos la tabla del apartado 3, observamos que cada iteración se podría realizar durante el tiempo que tarda la multiplicación (instrucción 6). Esto significa que cada iteración tarda 4 ciclos de reloj, y se ejecutan bien 8 instrucciones cuando el salto BFLE es no tomado, o 7 instrucciones cuando el salto es tomado. Para el primer caso, $CPI = 4/8 = 0.5$ ciclos ($IPC=2$ instrucciones/ciclo), y para el segundo caso, $CPI = 4/7 = 0.57$ ciclos ($IPC=1.75$ instrucciones/ciclo).

Problema 2-4-3 (examen Parcial 2005). Utilizando el diagrama de ciclos que aparece más abajo, describe la evolución de las instrucciones de dos iteraciones del bucle que se muestra a continuación en un procesador superescalar de dos vías desacoplado que realiza ejecución fuera de orden a través del Algoritmo de Tomasulo.

```

1   Loop:      LD    F0, 0(R1)
2              ADDD  F4, F0, F2
3              SD    0(R1), F4
4              SUBI  R1, R1, 8
5              BNEZ  R1, Loop

```



La latencia de las instrucciones en las respectivas unidades funcionales son las siguientes: 2 ciclos (LD, SD), 3 ciclos (ADDD), 1 ciclo (resto de instrucciones). La condición de emparejamiento de las instrucciones consiste en que una de ellas sea de enteros y la otra de punto flotante. Las instrucciones de carga y almacenamiento se consideran instrucciones de enteros. Las instrucciones de salto no son emparejables y se predicen de forma perfecta. Suponer que en el funcionamiento de la microarquitectura del procesador se distinguen las siguientes etapas: búsqueda (IF), decodificación (ID), renombramiento de registros (IS), ejecución en la unidad funcional (EX), escritura de resultados en los bancos de registros (WB). Suponer que la cola de instrucciones y las estaciones de reserva nunca se saturan completamente. Adicionalmente, suponer que existen las siguientes unidades funcionales en el cauce para enteros: ALU para enteros, saltos, cargas, almacenamientos. Y además existe un cauce de punto flotante con una unidad funcional de suma/resta en punto flotante. La anticipación de resultados a las instrucciones dependientes se realiza a través de las estaciones de reserva o del banco de registros. A partir del gráfico, calcula el número de ciclos promedio en los que se ejecuta cada iteración.

Solución:

Resumen de características:

- Procesador superescalar de 2 vías: Se puede mandar a ejecutar hasta 2 instrucciones a la vez.
- La condición de emparejamiento consiste en que una instrucción debe ser de tipo entero y la otra de tipo flotante (las instrucciones de carga/almacenamiento se consideran de tipo entero), a excepción de la instrucción de salto condicional que no es emparejable.
- Los saltos se predicen de forma perfecta, y no existen fallos de predicción.
- El procesador implementa Tomasulo con ejecución y terminación Fuera de Orden.
- La característica de “Desacoplado” consiste en que existen cauces independientes para los dos tipos de instrucciones, uno para enteros y otro para punto flotante. Igualmente existe un Bus de Dato Común para las instrucciones de tipo entero (CDB1) y otro para las de tipo flotante (CDB2), comunicados entre sí a través de un búfer de desacoplamiento.

- Existe una unidad funcional para las cargas/almacenamientos, otra para la resta en enteros, otra para saltos, y una última para la suma en punto flotante.
- La cola de Issue y las Estaciones de Reserva no se saturan (nunca se llenan).
- La anticipación de resultados se realiza a través de las estaciones de reserva o del banco de registros (no existe un camino directo entre las unidades funcionales).
- Latencias:
 - 2 ciclos cargas/almacenamientos.
 - 3 ciclos suma en punto flotante.
 - 1 ciclo resto de instrucciones (resta en enteros y salto condicional).
- Etapas del procesador: IF (búsqueda), ID (decodificación), IS (renombramiento de registro y envío a las estaciones de reserva), EX (ejecución), WB (escritura). Como existen 2 bancos de registros independientes entre sí distinguiremos un WB_i para enteros y un WB_f para flotante. En este procesador no existe una etapa de retirado de instrucciones, ya que como puede verse en el diagrama no hay ningún búfer de reordenamiento, por lo que los datos se actualizan fuera de orden directamente en el banco de registros.
- El diseño del cauce tiene una peculiaridad importante respecto a las cargas/almacenamientos. Como se ha comentado estas operaciones son consideradas de tipo entero, lo que quiere decir que se ejecutarán en el cauce de enteros. Sin embargo, estas operaciones pueden cargar o almacenar un dato en punto flotante. Esto hace necesario el traspaso del mismo de un lado al otro a través del búfer de desacoplamiento. Esta peculiaridad se explicará más a fondo al analizar las instrucciones más adelante.

A continuación, se muestra el diagrama de ciclos donde la primera columna de la izquierda muestra las dos iteraciones del bucle y en las sucesivas columnas se mostrará el estado de las instrucciones en el procesador en cada ciclo de reloj. Como se puede observar, la traza en tiempo de ejecución corresponde a dos iteraciones del código.

Instrucción	IF	ID	IS	EX	WB _i	WB _f	Observaciones
1. LD F0,0(R1)	1	2	3	4-5	6	7	
2. ADDD F4,F0,F2	1	2	3	4-10	12	11	
3. SD 0(R1),F4	2	3	4	5-14			
4. SUBI R1,R1,8	3	4	5	6	7		
5. BNEZ R1,Loop	4	5	6	7-8			
1. LD F0,0(R1)	5	6	7-8	9-10	11	12	→LD y ADDD se paran en IS para dar tiempo a confirmar que la predicción de salto es correcta
2. ADDD F4,F0,F2	5	6	7-8	9-15	17	16	
3. SD 0(R1),F4	6	7	8	9-19			
4. SUBI R1,R1,8	7	8	9	10	13		→SUBI se para porque el CDB1 está ocupado con las instrucciones 1 y 2
5. BNEZ R1,Loop	8	9	10	11-14			

Trazas de Instrucciones	Ciclo																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1. LD F0,0(R1)	IF	ID	IS	EX	EX	WB _i	WB _f												
2. ADDD F4,F0,F2	IF	ID	IS	EX!	EX!	EX!	EX!	EX	EX	EX	WB _f	WB _i							
3. SD 0(R1),F4		IF	ID	IS	EX!	EX!	EX!	EX!	EX!	EX!	EX!	EX!	EX	EX					
4. SUBI R1,R1,8			IF	ID	IS	EX	WB _i												
5. BNEZ R1,Loop				IF	ID	IS	EX!	EX											
1. LD F0,0(R1)					IF	ID	IS!	IS	EX	EX	WB _i	WB _f							
2. ADDD F4,F0,F2					IF	ID	IS!	IS	EX!	EX!	EX!	EX!	EX	EX	EX	WB _f	WB _i		
3. SD 0(R1),F4						IF	ID	IS	EX!	EX!	EX!	EX!	EX!	EX!	EX!	EX!	EX!	EX	EX
4. SUBI R1,R1,8							IF	ID	IS	EX!	EX!	EX	WB _i						
5. BNEZ R1,Loop								IF	ID	IS	EX!	EX!	EX!	EX					
	1ª iteración				2ª iteración														
	Última iteración																		

WB_i: Escritura en las estaciones de reserva de enteros.

WBf: Escritura en las estaciones de reserva de punto flotante.

Ciclos promedio de cada iteración = 4 ciclos; Última iteración = 15 ciclos.

Las dos primeras instrucciones son la carga, considerada como de tipo entero, y la suma en punto flotante. Como son emparejables ambas pueden entrar a la vez en el cauce.

1. LD F0, 0(R1):

Entra en el cauce en el ciclo 1 y al ser la primera instrucción avanza sin problema por las etapas del procesador. Avanza por las etapas de búsqueda, decodificación e issue (ciclos 1, 2 y 3). En la etapa de ejecución tarda dos ciclos (ciclos 4 y 5) en la unidad de cargas/almacenamientos situada en el cauce de enteros, pero debe notarse que el dato que se lee es de tipo punto flotante y se debe guardar en el banco de registros de punto flotante (registro F0), y que éste se encuentra en el otro cauce de punto flotante.

Para poder guardar el dato correctamente, la instrucción debe pasar por dos etapas de escritura: primero por una etapa de escritura WBi, donde se escribe el dato en el búfer de desacoplamiento a través del bus de datos común del cauce de enteros CDB1; para luego en el siguiente ciclo, pasar por una etapa de escritura WBf, donde el dato se guarda, a través del bus de datos común del cauce de punto flotante CDB2, en el registro F0 en el banco de registros de punto flotante y en las estaciones de reserva que estuvieran esperando por este dato (en este caso en la estación de reserva de la suma ADDD). Recordar que sólo puede escribirse un dato cada vez en cada bus de datos común, por lo que si queremos escribir en un bus que se encuentra en uso la instrucción debe pararse hasta que el bus vuelva a estar disponible.

2. ADDD F4, F0, F2:

Entra a la vez que la instrucción anterior y evolucionan juntas por las etapas de búsqueda, decodificación e issue. Sin embargo, existe una dependencia de tipo verdadero RAW entre la suma que lee del registro F0 y la carga previa que escribe en él. Debido a esto, la instrucción no puede comenzar a ejecutarse y permanecerá bloqueada esperando a que el dato se actualice en la estación de reserva. Esto sucede en el ciclo 7. En este ciclo el dato se escribe en el bus de datos común CDB2 y en los registros que esperaban por él. Entre ellos la estación de reserva de la suma. Así en el ciclo 8 puede comenzar a ejecutarse la instrucción ADDD en su unidad funcional. Según la tabla de latencias, esta operación tarda 3 ciclos (ciclos 8, 9 y 10). En el ciclo 11, cuando se encuentra en la etapa WBf, se escribe el dato en el registro F4 del banco de registros a través del bus de datos común. También debe escribirse en el búfer de desacoplamiento dado que la próxima instrucción es un almacenamiento (tipo entero) y requiere el dato. De esta forma, en el ciclo 12, cuando se encuentra en la etapa WBi, el dato proveniente de la suma se guarda en la estación de reserva del almacenamiento, pasando por el bus CDB1.

En el ciclo 2 se buscan 2 nuevas instrucciones en la cache de instrucciones, pero ambas instrucciones son de tipo entero (un almacenamiento y una resta), y por ello, no son emparejables. De esta forma, en este ciclo sólo puede lanzarse una instrucción, la primera de ellas: el almacenamiento.

3. SD 0(R1), F4:

Entra en la etapa de búsqueda en el ciclo 2, y avanza sin problemas por las etapas de decodificación e issue (ciclos 3 y 4). Recordar que el enunciado del problema garantiza que la cola de issue y las estaciones de reserva tienen capacidad suficiente para soportar todas las instrucciones en vuelo, y por lo tanto, nunca se producirán penalizaciones por esta causa. Sin embargo, el almacenamiento requiere el dato resultado de la suma, registro F4, que se escribe por la instrucción anterior (dependencia verdadera RAW). Así que no podrá comenzar a ejecutarse en la unidad funcional hasta que dicho dato esté disponible y se actualice en su estación de reserva. Como se comentó al final de la instrucción anterior, este dato llega al bus de datos común CDB1 y se escribe

en la estación de reserva correspondiente al almacenamiento en el ciclo 12. De esta forma, la ejecución en la unidad funcional que se hallaba parada en la etapa EX desde el ciclo 5 puede comenzar en el ciclo 13 durando hasta el 14 (según la latencia especificada).

Notar que la instrucción de almacenamiento no necesita una etapa de escritura WB dado que el almacenamiento en memoria ya se produce en la etapa de ejecución. Debido a esto, podemos decir que termina en el ciclo 14, en el cual finaliza la etapa EX.

En el ciclo 3 vuelven a buscarse 2 instrucciones, sin embargo, la segunda de ellas es un salto y no se puede emparejar. Por ello, se lanza sólo la resta.

4. SUBI R1, R1, 8:

Entra en el ciclo 3 y avanza por todas las etapas sin problema al no tener ningún tipo de dependencia con instrucciones anteriores, y además encontrarse todos los recursos disponibles. La resta tiene una latencia de 1 ciclo y tras ejecutarse pasa por una etapa de escritura WBi. En ella se transmite el resultado por el bus de datos común de enteros CDB1, se actualiza el registro R1 del banco de registros y las estaciones de reserva que estuvieran esperando por dicho dato (en este caso el salto condicional y la carga siguientes).

En el ciclo 4 vuelven a buscarse 2 instrucciones. Sin embargo, la primera de ellas es un salto y no se puede emparejar. Por ello, se lanza sólo el salto condicional.

5. BNEZ R1, Loop:

Entra en el cauce en el ciclo 4 y avanza por las etapas de decodificación e issue en los ciclos 5 y 6. En el ciclo 7 no puede ejecutarse dado que depende del resultado de la operación anterior, que en ese mismo ciclo se encuentra en etapa de escritura actualizando el registro en el banco de registros y en la estación de reserva de nuestro salto. No es hasta el ciclo 8 cuando puede comenzar. El salto tiene una latencia en la unidad de saltos de 1 ciclo y dado que se dispone de un predictor perfecto nunca fallará, por lo que nunca se tendrá que vaciar el cauce por fallo de predicción de saltos.

Ha finalizado la primera iteración. Gracias a nuestro predictor perfecto, en el ciclo 5 vuelven a buscarse 2 instrucciones, que corresponden con las dos primeras instrucciones de la segunda iteración. En este caso, disponemos de una instrucción de carga (tipo entero) y de una suma en punto flotante, por lo que son emparejables y pueden lanzarse en el mismo ciclo de reloj.

La segunda iteración se ejecutará de forma idéntica a la primera con una única salvedad: cuando la primera instrucción de la segunda iteración (la carga en F0) quiere escribir en el bus de datos común de punto flotante CDB2 para guardar el dato en el registro F0 en el ciclo 11, se encuentra con que dicho bus se encuentra ocupado en ese momento por la instrucción de suma de la primera iteración. Esto provoca una penalización de 1 ciclo de reloj y el dato de la carga deberá esperar en el búfer de desacoplamiento hasta el siguiente ciclo en el que el dato podrá escribirse en CDB2, en el banco de registro, y en las estaciones de reserva que esperasen por él. De esta forma, este ciclo de penalización se propaga a la suma de la segunda iteración debido a la dependencia RAW con la carga anterior, y a su vez el almacenamiento siguiente que depende de que la suma termine de escribir para poder tomar el dato y poder comenzar a ejecutarse.

En consecuencia, debido a que el almacenamiento era la última instrucción en ejecutarse del programa, la duración del mismo se ve perjudicada debido a este ciclo de penalización. Como puede apreciarse, entre la primera instrucción lanzada de la primera iteración y la primera de la segunda pasan 4 ciclos (del ciclo 1 al 4), con lo que se puede lanzar a ejecutar una iteración completa cada 4 ciclos. Se aprecia también que la última iteración tarda 11 ciclos en terminar la ejecución (del ciclo 9 al 19), con lo que la última iteración tiene una duración de $11+4=15$ ciclos.

Un detalle a tener en cuenta es el ritmo de salida de instrucciones del cauce. Hemos concluido que podemos lanzar una iteración cada 4 ciclos. Sin embargo, si nos fijamos en el tiempo que transcurre entre el ciclo en que termina la última instrucción de la primera iteración y el ciclo en que termina la última instrucción de la segunda iteración, podemos observar que son necesarios 5 ciclos para que una iteración abandone el cauce. Por lo tanto, podemos lanzar una iteración cada 4 ciclos, pero se necesitan 5 ciclos para que lo abandone. Esto va a provocar una acumulación progresiva de instrucciones en vuelo.

Generalmente, este problema provocaría penalizaciones de tipo estructural, por saturación de recursos, haciendo que las instrucciones deban retrasarse, aumentando así el promedio de ciclos necesarios por iteración de nuestro programa. Sin embargo, recordemos que el procesador con el que se trabaja garantiza una capacidad suficiente para soportar todas las instrucciones en vuelo y por lo tanto, no se producirán penalizaciones por este motivo.

PROBLEMA 2-4-4 (Examen Parcial 27-1-2006). (a) **Mostrar la evolución temporal de las instrucciones de 2 iteraciones del siguiente programa en un procesador superescalar de 2 vías con predicción perfecta de hasta 1 salto por ciclo, cache de primer nivel perfecta y bloqueante, y ejecución fuera de orden. Suponer que no existen restricciones en el emparejamiento de instrucciones. En el caso de que la primera instrucción de una pareja que se introduce en la etapa IF sea un salto, la segunda instrucción es la que se encuentra en la dirección destino del salto. Cuando se retira un salto, no se puede retirar ninguna otra instrucción.**

```

1 Loop:  LD    F0,0(R1)
2         ADDD F4,F0,F2
3         SD    0(R1),F4
4         SUBI  R1,R1,8
5         BNEZ  R1,Loop

```

La ruta de datos tiene las siguientes etapas: búsqueda (IF), decodificación (ID), envío a ejecutar (IS), ejecución en las unidades funcionales (EX), actualización de las estaciones de reserva (WB), actualización del banco de registros y retirado de hasta 2 instrucciones por ciclo (Ret). Las actividades realizadas en cada una de estas etapas de segmentación son las mismas que las descritas en clases de teoría para el procesador DLX32ss. Las 4 unidades funcionales segmentadas disponibles y las etapas de ejecución correspondientes en las que están divididas son las siguientes: LW/SW (2 etapas: los almacenamientos actualizan la memoria en la etapa Ret), Saltos (2 etapas), operaciones de punto flotante (3 etapas), operaciones de enteros (2 etapas). Suponemos que la primera etapa de cada unidad funcional consiste en el copiado de operandos desde la estación de reserva hasta la unidad funcional correspondiente. Cada etapa de la unidad funcional sólo puede realizar la operación de una instrucción. Suponer que existe un único bus CDB para todas las unidades funcionales, tanto de enteros como de punto flotante.

Solución:

Resumen de características:

- Procesador superescalar de 2 vías: Se puede mandar a ejecutar hasta 2 instrucciones a la vez.
- No existen restricciones para el emparejamiento de instrucciones.
- Los saltos se predicen de forma perfecta (hasta un salto por ciclo). Si la primera instrucción de la pareja es un salto, la siguiente será la de la dirección de destino del salto y podrá entrar en IF a la vez que el salto.
- Implementa Tomasulo con ejecución fuera de orden y búfer de reordenamiento, el cual asegura que se realiza terminación de las instrucciones en orden.
- Existe un Bus de Datos Común único para todas las unidades funcionales.
- 4 unidades funcionales segmentadas, una para cada tipo de operación: carga/almacenamiento, operaciones en enteros, operaciones en punto flotante y saltos. Latencias:
 - 2 ciclos cargas/almacenamientos (los almacenamientos se actualizan en el retirado).
 - 3 ciclos operaciones en punto flotante.
 - 2 ciclos resto de instrucciones (operaciones en enteros y saltos).
- Considerar que en la primera etapa de las unidades funcionales se copia los operandos desde la estación de reserva hasta dicha unidad funcional.
- Etapas del procesador: IF (búsqueda), ID (decodificación), IS (renombramiento de registros y envío a las estaciones de reserva), EX (ejecución), WB (escritura, sólo una etapa WB dado que sólo existe un bus de datos común para todas las unidades funcionales), y Ret (actualización del estado de los registros y retirado de hasta 2 instrucciones por ciclo; excepto saltos, que se supone que deben retirarse de forma exclusiva). Los saltos deben verificar el cumplimiento de una condición especulada

para el salto. Si las instrucciones posteriores al salto son susceptibles de ser retiradas a la vez que el salto y podrían ser incorrectas al fallar la especulación. Por ello, se supone que el salto no es emparejable en la retirada de instrucciones.

A continuación, se muestra el diagrama de ciclos donde la primera columna de la izquierda muestra las dos iteraciones del bucle, y en las sucesivas columnas se mostrará el estado de las instrucciones en cada ciclo de reloj. Como se puede observar, la traza en tiempo de ejecución duplica automáticamente el código de cada iteración.

Instrucciones / Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1 LD F0,0(R1)	IF	ID	IS	EX	EX	WB	Ret												
2 ADDD F4,F0,F2	IF	ID	IS	EX!	EX!	EX!	EX	EX	EX	WB	Ret								
3 SD 0(R1),F4		IF	ID	IS	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!								
4 SUBI R1,R1,8		IF	ID	IS	EX	EX	WB	Ret!	Ret!	Ret!	Ret!	Ret							
5 BNEZ R1,Loop			IF	ID	IS	EX!	EX!	EX	EX	WB!	WB	Ret!	Ret						
1 LD F0,0(R1)			IF	ID	IS	EX!	EX!	EX	EX	WB!	WB!	WB!	Ret!	Ret					
2 ADDD F4,F0,F2				IF	ID	IS	EX!	EX!	EX!	EX!	EX!	EX!	EX	EX	EX	WB	Ret		
3 SD 0(R1),F4				IF	ID	IS	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!	Ret!	Ret		
4 SUBI R1,R1,8					IF	ID	IS	EX	EX	WB!	WB!	WB!	WB!	Ret!	Ret!	Ret!	Ret!	Ret	
5 BNEZ R1,Loop					IF	ID	IS	EX!	EX!	EX!	EX!	EX!	EX!	EX	EX	WB!	WB	Ret!	Ret

Las dos primeras instrucciones son la carga, considerada como de tipo entero, y la suma en punto flotante. Como son emparejables, ambas pueden entrar a la vez en el cauce.

1. LD F0,0(R1):

Entra en el cauce en el ciclo 1, y al ser la primera instrucción avanza sin problema por las etapas del procesador. Avanza por las etapas de búsqueda, decodificación e issue (ciclos 1, 2 y 3). En la etapa de ejecución tarda dos ciclos en la unidad funcional de carga/almacenamiento que se encuentra en el cauce de enteros (ciclos 4 y 5). En el ciclo 6 entra en la etapa de escritura WB escribiéndose el resultado en el búfer de reordenamiento, así como en las estaciones de reserva que dependen de él a través del bus de datos común. Por último, se llega a la etapa de retirado Ret en el ciclo 7 donde finalmente se actualiza el registro F0 en el banco de registros. Recordar que sólo puede escribirse un dato en cada ciclo a través del bus de datos común, por lo que si queremos escribir en el bus y éste se encuentra en uso, la instrucción debe pararse hasta que vuelva a estar disponible.

2. ADDD F4, F0, F2:

Entra a la vez que la instrucción anterior y evolucionan juntas por las etapas de búsqueda, decodificación e issue. Sin embargo, existe una dependencia de tipo verdadero RAW entre la suma que lee del registro F0 y la carga previa que escribe en él. Debido a esto, la instrucción no puede comenzar a ejecutarse y permanecerá bloqueada esperando a que el dato se actualice en la estación de reserva. Esto sucede en el ciclo 6, en el cual el dato se escribe en el bus de datos común y en la estación de reserva que esperaba por él. Así en el ciclo 7 puede comenzar a ejecutarse la instrucción de suma en su unidad funcional. Según la tabla de latencias, esta operación tarda 3 ciclos (7, 8 y 9). En el ciclo 10, cuando se encuentra en la etapa WB, se escribe en el búfer de reordenamiento y las estaciones de reserva pertinentes (en este caso en la instrucción de carga posterior). Finalmente, se retira la instrucción en el ciclo 11 y se actualiza el registro F4 en el banco de registros.

En el ciclo 2 se buscan 2 nuevas instrucciones en la cache de instrucciones, un almacenamiento y una resta. Al no haber restricciones para el emparejamiento ambas pueden entrar a la vez.

3. SD 0(R1), F4:

Entra en la etapa de búsqueda en el ciclo 2, y avanza sin problemas por las etapas de decodificación e issue (ciclos 3 y 4). Sin embargo, el almacenamiento requiere el dato resultado de la suma, F4, que se escribe en la instrucción anterior (dependencia verdadera RAW), así que no podrá ejecutarse hasta que dicho dato esté disponible. Los almacenamientos actualizan la cache de datos cuando se retiran del búfer de reordenamiento en la etapa de retirado. Por ello, la instrucción no pasa por una etapa EX. Como se comentó al final de la instrucción anterior, este dato llega al bus de datos común y se escribe en el búfer de reordenamiento en el ciclo 10. De esta forma, el almacenamiento que se hallaba parado en la etapa Ret desde el ciclo 5 puede almacenarse en el ciclo 11. Debe notarse que la etapa de retirado coincide con la anterior, ya que según el enunciado pueden retirarse hasta 2 instrucciones por ciclo.

4. SUBI R1, R1, 8:

Entra en el ciclo 2, avanzando por todas las etapas sin problema al no tener ningún tipo de dependencia con instrucciones anteriores, además de encontrarse todos los recursos disponibles. La resta tiene una latencia de 2 ciclos (5 y 6), y tras ejecutarse pasa por una etapa de escritura WB (ciclo 7), en la cual se escribe en el bus de datos común, guardando el resultado en el búfer de reordenamiento y en las estaciones de reserva que estuvieran esperando por dicho dato (en este caso el salto condicional, la carga, y la resta de la siguiente iteración). El enunciado limita la cantidad de instrucciones que se pueden retirar en 2. Como ya hay dos instrucciones en fase de retirado la resta debe esperar hasta el ciclo 12.

En el ciclo 3 vuelven a buscarse 2 instrucciones. La primera de ellas es un salto y la segunda será la instrucción destino del salto, que será la primera instrucción de la segunda iteración.

5. BNEZ R1, Loop:

Entra en el cauce en el ciclo 3 y avanza por las etapas de decodificación e issue en los ciclos 4 y 5. En el ciclo 6 no puede ejecutarse dado que depende del resultado de la operación anterior. En el ciclo 7 la resta actualiza los datos en la estación de reserva del salto, y es en el ciclo 8 cuando podrá comenzar a ejecutarse. Tiene una latencia de 2 ciclos (ciclos 8 y 9). En el siguiente ciclo tendrá que detenerse de nuevo debido a que el bus de datos común se encuentra ocupado por la instrucción de suma y no puede escribir en él hasta el ciclo 11 (etapa WB). En el ciclo siguiente se paraliza la retirada debido a que los saltos requieren retirarse solos, lo cual ocurre en el siguiente ciclo, ciclo 13. Las etapas de WB y Ret son necesarias dado que no se sabe si la especulación es correcta hasta la retirada. En caso de no ser correcta habría que borrar las instrucciones posteriores que entraron por una predicción del salto.

1. LD F0, 0(R1):

Comienza la segunda iteración. La carga entra en el cauce en el ciclo 3 junto al salto. Avanza por las etapas de búsqueda, decodificación e issue (ciclos 3, 4 y 5). En el ciclo siguiente no puede comenzar a ejecutarse debido a la dependencia RAW en R1 con la resta de la iteración anterior. La resta actualiza el dato en la estación de reserva en el ciclo 7. En el ciclo 8 puede comenzar a ejecutarse la carga y tarda 2 ciclos (8 y 9). En el ciclo 10 la etapa WB se encuentra ocupada por la suma de la iteración anterior y debe esperar. En el ciclo 11 es el salto el que tiene prioridad al ser anterior y sigue esperando. Es en el ciclo 12 cuando la carga puede escribir a través del bus de datos común en el búfer de reordenamiento y en las estaciones de reserva que se encuentren esperando. Finalmente, debido a la restricción de retirado del salto la carga debe esperar hasta el ciclo 13 para poder retirarse.

En el ciclo 4 vuelven a buscarse 2 instrucciones. La primera de ellas es la suma de la segunda iteración y la segunda el almacenamiento.

2. ADD F4, F0, F2:

Entra y evoluciona por las etapas de búsqueda, decodificación e issue en los ciclos 4, 5 y 6. Para poder entrar a ejecutarse necesita que la instrucción de carga anterior actualice el valor de F0 en su estación de reserva. Esto sucede en el ciclo 12 y entrará a ejecutarse durante los ciclos 13, 14 y 15. En el ciclo 16 el bus de datos común se encuentra libre y se lleva a cabo la etapa de escritura WB en el búfer de reordenamiento y las estaciones de reserva pertinentes. En el ciclo 17 se retira la instrucción.

3. SD 0(R1), F4:

Entra y evoluciona por las etapas de búsqueda, decodificación e issue en los ciclos 4, 5 y 6 junto con la instrucción anterior. El almacenamiento requiere el dato resultado de la suma que se escribe en la instrucción anterior (dependencia verdadera RAW con el registro F4). Así que no podrá ejecutarse hasta que dicho dato esté disponible. Como se comentó en el almacenamiento de la primera iteración, estos se saltan la etapa de ejecución y se llevan a cabo en la etapa de retirada. El valor actualizado de F4 se encuentra disponible en el ciclo 16, así en el ciclo 17 se guarda en memoria y se retira la instrucción.

En el ciclo 5 vuelven a buscarse las 2 últimas instrucciones. La primera de ellas es la resta de la segunda iteración y la segunda el salto posterior.

4. SUBI R1, R1, 8:

Entra y evoluciona por las etapas de búsqueda, decodificación e issue en los ciclos 5, 6 y 7. En este mismo ciclo es cuando la instrucción de resta de la iteración anterior (con la que tiene una dependencia de tipo de verdadera RAW con el registro R1) realiza la escritura en la estación de reserva. De esta forma la instrucción puede entrar a ejecutarse en el ciclo 8 y permanece 2 ciclos en EX (8 y 9). En el ciclo 10 debe detenerse debido a que la suma de la primera iteración se encuentra escribiendo en WB. Lo mismo sucede en el ciclo 11 con el salto de la primera iteración y en el 12 con la carga de la segunda iteración. Es en el ciclo 13 cuando la etapa está disponible y la resta escribe en el búfer de reordenamiento y en la estación de reserva del salto siguiente. Finalmente, la instrucción vuelve a detenerse hasta que puede retirarse en el ciclo 18.

5. BNEZ R1, Loop:

Entra y evoluciona por las etapas de búsqueda, decodificación e issue en los ciclos 5, 6 y 7 junto a la resta. Existe una dependencia RAW con la misma debido al registro R1, por lo que debe detenerse hasta que se actualice en su estación de reserva. Esto sucede en el ciclo 13. En el ciclo 14 entra a ejecutarse y permanece durante 2 ciclos (14 y 15). En el ciclo 16 no puede entrar en WB debido a la instrucción de suma. Entra en el ciclo 17, escribiendo en el búfer de reordenamiento. Debido a la restricción de retirada de saltos no puede retirarse junto a la resta y debe esperar un ciclo más, terminando en el ciclo 19.

(b) Calcular el número de ciclos que se requieren y el IPC promedio. Suponer que: (1) no existen penalizaciones debidas al tamaño limitado de la “Cola Issue”, las estaciones de reserva, y del Búfer de Reordenamiento; (2) la memoria cache tiene un solo puerto de lectura/escritura.

Solución:

Ciclos en los que se ejecuta el programa= 19 ciclos.

$IPC = \text{Número de instrucciones} / \text{Ciclos} = 10 \text{ instrucciones} / 19 \text{ ciclos} = 0.5 \text{ instrucciones/ciclo}$ frente al IPC ideal que tendría el procesador de 2 instrucciones por ciclo.

Agradecimientos

En el curso 2006-2007, los alumnos David López Luengo, Francisco Javier Montero Vega, Antonio José Sánchez López y Oumlfadi Bouchraoui participaron en la redacción de este bloque de problemas.