

## Segmentación básica

### La ruta de datos del MIPS

**Ejercicio 2.1.** Un procesador compatible binario con el MIPS64 posee su ciclo de instrucción segmentado en 5 fases:

**IF:** Búsqueda de la instrucción a ejecutar.

**ID:** Decodificación de la instrucción y lectura de los registros operandos.

**EX:** Operación en la U.A.L. Cálculo de la condición y escritura del PC en las instrucciones de salto.

**M:** Acceso a memoria en las instrucciones de Carga y Almacenamiento.

**WB:** Escritura del resultado sobre el registro destino.

La máquina resuelve los riesgos de control mediante el salto retardado. También posee antememorias de instrucciones y datos separadas, así como dos puertos de lectura y uno de escritura en el banco de registros.

Sobre dicho procesador se está ejecutando el siguiente bucle, compuesto de  $n$  iteraciones:

```

    . . .
L   :   ld $t1,X($t2)
        dadd $t1,$t1,$t3
        sd $t1,X($t2)
        daddi $t2,$t2,8
        daddi $t4,$t4,-1
        bnez $t4,L
        nop
        nop
    
```

1. Dibuja un diagrama en el que se indique, para cada instrucción y ciclo de reloj, qué fase de la instrucción se está completando. Considera sólo la primera iteración. Calcula los CPI y el tiempo de ejecución obtenidos en función del número de iteraciones  $n$ . Considera los siguientes casos:

- a) Los riesgos de datos se resuelven mediante la inserción de ciclos de parada.
- b) Los riesgos de datos se resuelven mediante la técnica del cortocircuito.

### **Solución:**

1. Puesto que el PC se escribe en la tercera fase del ciclo de instrucción, el *branch delay slot* es de dos instrucciones:

			PC				
d:	bnez \$t4,L	IF	ID	EX	M	WB	
d+4:	ds1		IF	ID	EX	M	WB
d+8:	ds2			IF	ID	EX	M
d+12/L:					IF	ID	EX
						M	WB

- a) Código original con ciclos de parada.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19			
begin:	ld \$t1,X(\$t2)	IF	ID	EX	M	WB																	
	dadd \$t1,\$t1,\$t3		IF	ID	ID	ID	EX	M	WB														
	sd \$t1,X(\$t2)			IF	IF	IF	ID	ID	ID	EX	M	WB											
	daddi \$t2,\$t2,8						IF	IF	IF	ID	EX	M	WB										
	daddi \$t4,\$t4,-1									IF	ID	EX	M	WB									
	bnez \$t4,begin										IF	ID	ID	ID	EX	M	WB						
	nop											IF	IF	IF	ID	EX	M	WB					
	nop														IF	ID	EX	M	WB				
begin:	ld \$t1,X(\$t2)														IF	ID	EX	M	WB				

Para una iteración, se ejecutan 8 instrucciones en 14 ciclos de reloj. Por tanto:

$$CPI = \frac{14}{8} = 1,75 \text{ ciclos}$$

Y el tiempo de ejecución para  $n$  iteraciones es  $14 \cdot n$  ciclos:

$$T_{ej} = I \cdot CPI \cdot T = 8n \cdot 1,75 \cdot T = 14 \cdot n \cdot T \text{ seg}$$

b) Código original con cortocircuitos.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14		
begin:	ld \$t1,X(\$t2)	IF	ID	EX	M	WB											
	dadd \$t1,\$t1,\$t3		IF	ID	ID	EX	M	WB									
	sd \$t1,X(\$t2)			IF	IF	ID	EX	M	WB								
	dadd \$t2,\$t2,#8					IF	ID	EX	M	WB							
	dsub \$t4,\$t4,#1						IF	ID	EX	M	WB						
	bnez \$t4,begin							IF	ID	EX	M	WB					
	NOP								IF	ID	EX	M	WB				
	NOP									IF	ID	EX	M	WB			
begin:	ld \$t1,X(\$t2)										IF	ID	EX	M	WB		

Para una iteración, se ejecutan 8 instrucciones en 9 ciclos de reloj. Por lo tanto:

$$CPI = \frac{9}{8} = 1,125$$

Y el tiempo de ejecución para  $n$  iteraciones es  $9 \cdot n$  ciclos:

$$T_{ej} = I \times CPI \times T = 8n \times 1,125 \times T = 9 \cdot n \cdot T \text{ seg}$$

□

**Ejercicio 2.2.** Se tiene el siguiente código en alto nivel:

```
typedef struct elem
    int x[1..10];
    *elem siguiente;
;

int cont;
*elem p;

...

do
    cont = cont + 1;
    p = (*p).siguiente;
while (p != NULL);

...
```

El tipo `int` y los punteros ocupan 64 bits. La constante `NULL` se representa con un 0. El compilador genera el bucle como se muestra seguidamente:

```
; cont se ubica en R2
; p se ubica en R1
...
eti: dadd r2, r2, #1
     ld r1, 80(r1)
     bnez r1, eti
     nop           ; tantos nops como haga falta
```

Dicho código se pretende ejecutar sobre distintas versiones de un procesador segmentado en 5 etapas:

**IF:** Búsqueda de la instrucción a ejecutar.

**ID:** Decodificación de la instrucción y lectura de los registros operandos (2º semiciclo del reloj).

**EX:** Operación en la U.A.L.

**ME:** Acceso a memoria en las instrucciones de Carga y Almacenamiento.

**WB:** Escritura del resultado sobre el registro destino (1er semiciclo del reloj).

El procesador resuelve los riesgos de datos mediante la técnica del cortocircuito, los de control mediante el salto retardado y funciona a 100 MHz. También emplea arquitectura *Harvard* (caches separadas para instrucciones y datos).

En cada uno de los supuestos siguientes, calcula la longitud del *branch delay slot* (o sea, a cuántas instrucciones afecta el salto retardado) y los CPI, los MIPS y el tiempo de ejecución que alcanza el procesador uponiendo que el bucle se ejecuta 10.000 veces:

1. El cálculo de la dirección efectiva, condición de salto y la escritura del PC se realiza en ID.
2. El cálculo de la dirección efectiva y condición de salto se realiza en EX, y la escritura del PC se realiza en M.

### **Solución:**

1. Escritura del PC en ID; arquitectura *Harvard*.

El *branch delay slot* es de una instrucción, por lo que se ejecutan cuatro instrucciones en cada iteración. El riesgo de datos entre *lw* y *bnez* requiere dos ciclos de parada, dando un total de seis ciclos por iteración. En consecuencia:

$$\overline{\text{CPI}} = \frac{6}{4} = 1,5 \text{ ciclos}$$

$$\text{MIPS} = \frac{I}{T_{ej} \cdot 10^6} = \frac{I}{I \cdot \overline{\text{CPI}} \cdot T \cdot 10^6} = \frac{f(\text{en MHz})}{\overline{\text{CPI}}} = \frac{100}{1,5} = 66,7 \text{ MIPS}$$

Para procesar 10.000 iteraciones harán falta 60.000 ciclos de reloj, que, a 10 ns por ciclo, son 600.000 ns = 0.6 ms. Al mismo resultado llegamos tras aplicar la ecuación del tiempo de ejecución con  $I = 4 \cdot 10^4$  instrucciones:

$$T(10000 \text{ iteraciones}) = (4 \cdot 10^4) \cdot 1,5 \cdot \frac{1}{100 \cdot 10^6} = 6 \cdot 10^{-4} \text{ s.}$$

2. Escritura del PC en ID; cache común de instrucciones y datos.

El *branch delay slot* es también de una instrucción: se ejecutan cuatro instrucciones en cada iteración. El riesgo estructural entre las instrucciones *lw* y *nop* se superpone al de datos entre *load* y *bnez*. El bucle continúa consumiendo seis ciclos por iteración. Los parámetros son los mismos que en el caso anterior.

3. Escritura del PC en M; arquitectura *Harvard*.

El *branch delay slot* es ahora de tres instrucciones: se ejecutan seis instrucciones en cada iteración. El riesgo de de datos entre *lw* y *bnez* requiere ahora un único ciclo de parada. El bucle consume siete ciclos por iteración. Se obtienen los siguientes parámetros:

$$\overline{\text{CPI}} = \frac{7}{6} = 1,17 \text{ ciclos}$$

$$\text{MIPS} = \frac{100}{1,17} = 85,7 \text{ MIPS}$$

Para procesar 10.000 iteraciones harán falta 70.000 ciclos de reloj, que, a 10 ns por ciclo, son 700.000 ns = 0.7 ms.

□

**Ejercicio 2.3.** Un computador, valorado en 2000\$, lleva un MIPS/LC, idéntico al MIPS (5 etapas de segmentación: búsqueda de la instrucción (IF), decodificación y lectura de registros (ID), Ejecución (EX), Acceso a memoria (MEM) y Escritura de registros (WB)) pero con una sola memoria cache común para instrucciones y datos, con *branch slot* de una instrucción, cortocircuitos y reloj a 80 MHz. Se tiene instalado un compilador de dominio público que compila el siguiente texto:

```
do {
  if (v[i] != 0) {
    temp = v[i];
    v[i] = w[i];
    w[i] = temp;
  }
  i = i-1;
} while (i != 0);
```

y genera el siguiente código:

```
eti1:  ld r2,v(r1)
       beqz r2,eti2
       nop
       ld r3,w(r1)
       sd r3,v(r1)
       sd r2,w(r1)
eti2:  dadd r1,r1,-8
       bnez r1,eti1
       nop
```

El bucle se aplica a vectores con un 80 % de componentes iguales a 0.

1. Calcula el CPI medio para tallas  $n$  grandes.
2. Supón que el bucle original es una buena muestra de la carga usual de dicho computador. Para mejorar el rendimiento, considera dos posibles inversiones:

- Cambiar el procesador por la versión MIPS/ST con memorias cache de instrucciones y datos separadas, con un coste de 200\$.
- Comprar un compilador comercial, valorado en 200\$, capaz de optimizar el código anterior reduciendo en 3 ciclos de reloj cada iteración en que  $v[i] \neq 0$  y en 2 ciclos cada iteración en que  $v[i] = 0$ . El número de instrucciones ejecutadas no se modifica.

Desde el punto de vista de la relación coste/prestaciones, y suponiendo que sólo podemos gastarnos 200\$, ¿sería interesante alguna de las dos mejoras anteriores? En caso afirmativo, ¿cuál convendría aplicar?

### Solución:

1. Cuando  $v(r1) \neq 0$ , en el bucle se ejecutan nueve instrucciones. Para resolver los riesgos, se introducen 6 ciclos de parada:
  - Dos ciclos que ha de esperar la instrucción `beqz r2, et12` para que la fase ID disponga del valor de `r2` generado por la instrucción previa `ld r2, v(r1)`. Recuérdese que si el *branch delay slot* es de una instrucción, el PC se escribe en la segunda etapa (ID) del ciclo de instrucción, por lo que la condición debe calcularse también en dicha etapa.
  - Tres ciclos ocasionados por las restantes instrucciones de carga y almacenamiento al producirse un riesgo estructural por tener una única cache de instrucciones y datos.
  - Un ciclo que ha de esperar la instrucción `bnez r1, et11` para que la fase ID disponga del valor de `r1` modificado por la instrucción previa `daddi r1, r1, -8`.

El total de ciclos requeridos es de  $9+2+3+1=15$ .

Si  $v(r1) = 0$ , se ejecutan seis instrucciones. Los ciclos de parada son los correspondientes a las instrucciones `beqz` y `bnez`, porque el resto de instrucciones de carga y almacenamiento no se ejecutan. El total de ciclos es, pues, de  $6+2+1=9$ .

Para calcular el CPI medio hay que tener en cuenta las proporciones de ceros en el vector  $v$ :

$$\overline{CPI} = \frac{\text{Número de ciclos}}{\text{Número de instrucciones}} = \frac{15 \cdot 0,2 + 9 \cdot 0,8}{9 \cdot 0,2 + 6 \cdot 0,8} = 1,55$$

2. Cálculo de las mejoras en tiempo medio de ejecución del bucle, medido en ciclos, en uno y otro caso:

- nuevo procesador, en el que se eliminan los tres riesgos estructurales sólo cuando  $v(r1) \neq 0$ :

$$\Delta t = \frac{15 \cdot 0,2 + 9 \cdot 0,8}{12 \cdot 0,2 + 9 \cdot 0,8} = 1,06$$

- nuevo compilador:

$$\Delta t = \frac{15 \cdot 0,2 + 9 \cdot 0,8}{12 \cdot 0,2 + 7 \cdot 0,8} = 1,28$$

Para el mismo incremento de coste (10 %), queda claro que la mejora apropiada es la adquisición del nuevo compilador. Por otra parte, la mejora obtenida cambiando el procesador es inferior al incremento del coste, por lo que, desde el punto de vista de mantener una relación coste-prestaciones constante, no sería interesante.

□

**Ejercicio 2.4.** El ciclo de instrucción de un procesador *load/store* no segmentado se descompone en las siguientes fases (se indica entre paréntesis la duración de cada una):

- LI (10 ns): lectura de instrucción.
- DI (5 ns): decodificación de la instrucción y lectura de registros fuente.
- EXE (10 ns): cálculo de direcciones efectivas en instrucciones L/S, operación en instrucciones ALU, cálculo de condición y de valor del PC en instrucciones de salto.
- EPC (5 ns): escritura de PC en instrucciones de salto
- MEM (10 ns): acceso a memoria en instrucciones L/S
- ER (5 ns): escritura de registro destino en instrucciones de almacenamiento y de ALU.

El autómata que implementa el circuito de control cableado genera las fases en función del código de operación. El reloj funciona a 200 MHz, de manera que unas fases requieren dos ciclos y otras sólo uno. Todos los ciclos de instrucción comienzan por las fases LI y DI. Según el tipo de instrucción, las restantes fases del ciclo son (se indica entre paréntesis la frecuencia de cada tipo de instrucción):

- Instrucciones de carga (20 %): EXE, MEM y ER
- Instrucciones de almacenamiento (10 %): EXE y MEM
- Instrucciones ALU (50 %): EXE y ER
- Instrucciones de salto (20 %): EXE y EPC

Se pretende segmentar el procesador, utilizando registros de 2 ns de retardo y un reloj con desfase nulo. Desaparece la fase EPC y toda la lógica de salto pasa a la etapa DI, con lo que el retardo de esta etapa es ahora de 10 ns. El procesador queda con las 5 etapas LI, DI, EXE, MEM y ER. Se toman medidas reales y se observa que el 5 % de las instrucciones de carga generan un ciclo de parada por riesgo de datos, y que el compilador rellena el 10 % de los casos de *branch delay slot* con una instrucción inútil *nop*.

Se pide:

1. Los CPI del procesador no segmentado.
2. La frecuencia del reloj del procesador segmentado.
3. El número de instrucciones ejecutadas por el procesador segmentado en relación al no segmentado.
4. Los CPI del procesador segmentado.
5. La aceleración en el tiempo de ejecución obtenido por la segmentación.

### **Solución:**

1. CPI del procesador no segmentado.

Obtengamos primero el valor de CPI para cada tipo de instrucción:

%	tipo	CPI
20	carga	8
10	almacenamiento	7
50	cálculo (ALU)	6
20	bifurcación	6

La media ponderada es:

$$\overline{CPI}_{NS} = 0,20 \cdot 8 + 0,10 \cdot 7 + 0,50 \cdot 6 + 0,2 \cdot 6 = 6,5 \text{ ciclos}$$

2. Frecuencia del reloj del procesador segmentado.

El máximo retardo de etapa es de 10 ns, y el de registro de 2 ns. Por lo tanto, el periodo de reloj es  $t_S = 10 + 2 = 12 \text{ ns}$  y la frecuencia:

$$f = \frac{1}{t_S} = \frac{1}{12 \cdot 10^{-9}} = 83,3 \cdot 10^6 \text{ Hz} = 83,3 \text{ MHz}$$

3. Relación entre número de instrucciones ejecutadas.

El programa del procesador segmentado tendrá el mismo número de instrucciones que el del no segmentado, más las instrucciones *nop* que en el 10 % de los casos siguen al 20 % de bifurcaciones. Como la lógica de salto está ubicada en la segunda etapa (DI), el *branch delay slot* es de una instrucción. Por tanto,

$$I_S = I_{NS}(1 + 0,1 \cdot 0,2) = 1,02 \cdot I_{NS} \text{ ciclos}$$

4. CPI del procesador segmentado.

Cada instrucción buscada contribuye con un ciclo al tiempo de ejecución. También hay que contabilizar los ciclos perdidos por conflictos de datos en el 5 % de casos del 20 % de instrucciones de carga. Hay que tener en cuenta que el número total de instrucciones ha aumentado debido a las instrucciones *nop*:

cantidad	tipo	CPI
20	carga	$1 + (0,05 \cdot 1) = 1,05$
10	almacenamiento	1
50	cálculo (ALU)	1
20	bifurcación	1
2 (0,1 · 20)	<i>nop</i>	1
102	total	

La media ponderada es:

$$\overline{CPI}_S = \frac{20 \cdot 1,05 + 10 \cdot 1 + 50 \cdot 1 + 20 \cdot 1 + 2 \cdot 1}{102} = 1,01 \text{ ciclos}$$

Otra forma de proceder es considerar que, al estar segmentado, el CPI será de 1 más el número medio de ciclos de parada que origina cada instrucción. En nuestro caso, solo el 5 % de las instrucciones de carga (que son 20 de cada 102) insertan un ciclo de parada. Por lo tanto:

$$\overline{CPI}_S = 1 + \frac{0,2 \cdot 0,05}{1,02} = 1,01 \text{ ciclos}$$

5. Aceleración debida a la segmentación.

Dividiendo las expresiones del tiempo de ejecución del procesador no segmentado y segmentado, obtenemos:

$$S = \frac{T_{NS}}{T_S} = \frac{I_{NS} \cdot \overline{CPI}_{NS} \cdot t_{NS}}{I_S \cdot \overline{CPI}_S \cdot t_S} = \frac{I_{NS} \cdot 6,5 \cdot 5}{1,02 \cdot I_{NS} \cdot 1,01 \cdot 12} = 2,63$$

### Ejercicio 2.5.

Se tiene un procesador segmentado en 5 etapas (IF: búsqueda de la instrucción; ID: decodificación y lectura de registros; EX: operación en la ud. aritmética; MEM: acceso a memoria y WB: escritura de registros). El procesador incorpora el juego de instrucciones del MIPS y posee cache de instrucciones y de datos separadas. La frecuencia de reloj es de 200 MHz. Los riesgos de datos y de control se resuelven mediante la técnicas del *forwarding* e insertando dos ciclos de parada cada vez que aparece una instrucción de salto, respectivamente.

Los programas ejecutan, por término medio, un 18 % de saltos, un 39 % de cargas/almacenamientos y un 43 % de instrucciones aritméticas. Las cargas son doble frecuentes que los almacenamientos. Los accesos a *bytes* y *halfwords* suponen un 20 % de los accesos a memoria. La frecuencia de riesgos de datos entre una instrucción LOAD y otra posterior que consume el dato procedente de la memoria es la siguiente:

Frecuencia: 25 %	Frecuencia: 15 %
LOAD R1, ...	LOAD R1, ...
Instrucción que lee R1	Instrucción que no lee R1
...	Instrucción que lee R1

Con el objeto de mejorar las prestaciones, se plantea realizar las siguientes modificaciones:

- Eliminar las instrucciones de acceso a *bytes* y *halfwords* del juego de instrucciones. Como consecuencia, los programas que necesiten esta funcionalidad deberán utilizar otras instrucciones del procesador:

LB R1, x	LW R1, x'	SB R1, x	LW R2, x'
o	SRL R1, R1, #pos	o	SLL R1, R1, #pos
LH R1, x	AND R1, R1, #masc	SH R1, x	5 instrucciones ALU más
			SW x', R1

- Aumentar la frecuencia de reloj, al simplificar el diseño del procesador.

Se pide:

1. Los CPI del procesador original.
2. El número de instrucciones del procesador modificado en relación al original.
3. Los CPI del procesador modificado.
4. La frecuencia de reloj que debería alcanzarse, como mínimo, para que sea interesante incorporar las modificaciones propuestas.

### Solución:

1. CPI del procesador original.

Al ser un procesador segmentado, los CPI serán 1 más los ciclos medios de parada por instrucción. En este caso, se insertan cuando aparecen saltos (18 %, 2 ciclos de parada) y cuando se producen riesgos de datos en los que interviene una instrucción de carga y otra instrucción inmediatamente a continuación que consume el dato cargado (25 % de las cargas, 1 ciclo de parada). Sabemos que hay un 39 % de carga/almacenamiento, y que las cargas son doble frecuente que los almacenamientos. Si  $c$  y  $a$  son los porcentajes de carga y almacenamiento, respectivamente, tenemos:



$$\left. \begin{array}{rcl} c + a & = & 39 \\ c & = & 2a \end{array} \right\}$$

Resolviendo,  $c = 26\%$  y  $s = 13\%$ .

Por lo tanto, los ciclos de parada son:

$$18\% \cdot 2(\text{Saltos}) + 26\% \cdot 0,25 \cdot 1(\text{Cargas}) = 0,425$$

y  $CPI = 1,43$  ciclos.

## 2. Número de instrucciones en el procesador mejorado.

El procesador mejorado ejecuta las mismas instrucciones originales, menos las que hemos sustituido, más las que equivalen a las sustituidas.

Teniendo en cuenta que los accesos a *bytes* y *halfwords* suponen el 20 % de los accesos a memoria, la frecuencias de aparición de LB/LH y SB/SH es de:

$$\text{Frecuencia LB/LH} = 0,26 \cdot 0,20 = 0,052$$

$$\text{Frecuencia SB/SH} = 0,13 \cdot 0,20 = 0,026$$

Cada instrucción LB/LH se sustituye por 3 nuevas instrucciones, y cada instrucción SB/SH se sustituye por 8 nuevas instrucciones. Por tanto, la nueva cuenta de instrucciones es:

$$I' = I - 0,052 \cdot 1 + 0,052 \cdot 3 - 0,026 \cdot 1 + 0,026 \cdot 8 = 1,286I$$

## 3. CPI del procesador mejorado.

Los ciclos de parada se modifican debido a que las sustituciones de LB/LH introducen siempre un ciclo de parada, independientemente de si antes lo producían o no. También hay que tener en cuenta que el numero total de instrucciones ha cambiado:

$$\text{Nº de ciclos de parada: } \frac{0,18}{1,29} * 2(\text{Saltos}) + \frac{0,26}{1,29} * 0,8 * 0,25 * 1(LW) + \frac{0,26}{1,29} * 0,2 * 1(LB/LH) = 0,359$$

$$CPI' = 1,36 \text{ ciclos.}$$

## 4. Compararemos el tiempo de ejecución de la mejora propuesta con la configuración original.

$$\text{Original: } Tej = I \times CPI \times T = I \cdot 1,43 \cdot 5 = 7,15I \text{ ns}$$

$$\text{Mejora: } Tej' = I' \times CPI' \times T' = 1,286I \cdot 1,359 \cdot T' = 1,75IT' \text{ ns}$$

Igualando y despejando:

$T' = 4,08 \text{ ns}$ . Por lo tanto, la frecuencia debería aumentar hasta 245 MHz para fuera interesante la propuesta.

□

**Ejercicio 2.6.** Un procesador con arquitectura registro-memoria tiene el ciclo de instrucción segmentado en 6 etapas:

- IF: Búsqueda de la instrucción e incremento del PC.
- RF: Decodificación y lectura de registros (2º semiciclo).
- ALU1: Cálculo de la dirección efectiva en accesos a memoria y saltos.
- MEM: Acceso a memoria.
- ALU2: Operaciones aritméticas, evaluación de la condición de salto y escritura del nuevo PC, en su caso.

- WB: Escritura del registro destino (1<sup>er</sup> semiciclo).

Todas las instrucciones ejecutan las 6 etapas. Hay dos tipos de instrucciones aritméticas:

**Tipo R:** ALUOp Rd, Rs, Rt

**Tipo M:** ALUOp Rd, Rs, desp(Rt)

1. Para evitar riesgos estructurales, ¿cuál es el número mínimo de sumadores necesario en esta segmentación?  
Encuentra el número mínimo de puertos de lectura/escritura tanto para el banco de registros como para memoria, a fin de evitar riesgos estructurales.
2. Si la máquina utiliza la técnica del salto retardado, ¿cuánto vale el *delay-slot*?
3. ¿Tiene interés aplicar una estrategia *predict-taken* para los saltos condicionales hacia posiciones anteriores del código en este procesador? En caso afirmativo, ¿qué penalización en ciclos de reloj conllevan los saltos correctamente predichos?

### **Solución:**

1. Recursos necesarios para evitar riesgos estructurales.

- Número de sumadores. Hay tres etapas que pueden requerir un sumador:

IF: para incrementar el PC

ALU1: para calcular la dirección efectiva de alguna referencia a memoria

ALU2: para realizar alguna operación ALU

- Banco de registros: Se leen (2 registros) en la etapa RF, y se escribe (1 registro) en la etapa WB. Por tanto, necesitamos al menos **1 puerto de escritura y 2 puertos de lectura**.  
Por otra parte, puesto que el banco de registros es de ciclo partido, una optimización sería tener **un único** puerto combinado de lectura/escritura más otro puerto sólo de lectura.
- Memoria: Se lee en la etapa IF (instrucción) y en la etapa MEM (dato), y se escribe en la etapa MEM (dato). Por tanto, se requieren dos puertos, uno de lectura y otro de lectura/escritura o bien una utilizar antememorias de instrucciones y datos separadas (arquitectura Harvard).

2. Tamaño del *delay slot*.

Puesto que el PC se escribe en la fase ALU2, es cuando la instrucción de salto está ejecutando la fase WB cuando se puede buscar la instrucción destino del salto. Por lo tanto, se buscan y comienzan a ejecutar hasta 4 instrucciones entre la del salto y la instrucción destino del mismo.

Salto	IF	RF	A1	ME	A2	WB	
instr. 1		IF	RF	A1	ME	A2	WB
instr. 2			IF	RF	A1	ME	A2
instr. 3				IF	RF	A1	ME
instr. 4					IF	RF	A1
Destino						IF	RF

3. Interés de una estrategia *predict-taken*.

Sí, ya que:

- Se sabe la dirección de salto antes que la condición.

- Los saltos condicionales hacia atrás suelen ser efectivos en un alto porcentaje de los casos.

La penalización cuando la condición se cumple es de dos ciclos de reloj:

Salto	IF	RF	A1	ME	A2	WB	
instr. 1		IF	RF				
instr. 2			IF				
Destino				IF	RF	A1	ME

□

### Ejercicio 2.7.

Los siguientes diagramas instrucciones–tiempo corresponden a la ejecución de ciertos fragmentos de código en varios procesadores. Indica, para cada uno de los casos, qué técnica se utiliza para resolver los riesgos de datos (inserción de ciclos de parada o cortocircuito) y los riesgos de control (inserción de ciclos de parada, *predict-not-taken* o salto retardado), así como la fase en la que se escribe el PC.

1.	L	LW r2,a(r1)	IF	ID	EX	M	WB												
	L+4	ADD r3,r2,r3		IF	ID	ID	EX	M	WB										
	L+8	ADD r3,r4,r3			IF	IF	ID	EX	M	WB									
	L+12	SUB r1,r1,#4					IF	ID	EX	M	WB								
	L+16	BNEZ r1, L						IF	ID	EX	M	WB							
	L+20	SW z(r0), r3							IF	ID									
	L+24	ADD r3,r0,r0								IF									
	L	LW r2,a(r1)									IF	ID	EX	M	WB				
2.	L	LW r2,a(r1)	IF	ID	EX	M	WB												
	L+4	ADD r3,r2,r3		IF	ID	ID	EX	M	WB										
	L+8	ADD r3,r4,r3			IF	IF	ID	EX	M	WB									
	L+12	SUB r1,r1,#4					IF	ID	EX	M	WB								
	L+16	BNEZ r1, L						IF	ID	EX	M	WB							
	L+20	SW z(r0), r3							IF	IF	IF								
	L	LW r2,a(r1)										IF	ID	EX	M	WB			
3.	L	LW r2,a(r1)	IF	ID	EX	M	WB												
	L+4	SUB r1,r1,#4		IF	ID	EX	M	WB											
	L+8	ADD r3,r2,r3			IF	ID	ID	EX	M	WB									
	L+12	BNEZ r1, L				IF	IF	ID	EX	M	WB								
	L+16	ADD r3,r4,r3						IF	ID	ID	EX	M	WB						
	L	LW r2,a(r1)							IF	IF	ID	EX	M	WB					

### Solución:

1. Riesgos de datos: cortocircuito; riesgos de control: *predict-not-taken*, escritura del PC en EX.
2. Riesgos de datos: cortocircuito; riesgos de control: inserción de ciclos de parada, escritura del PC en M.
3. Riesgos de datos: inserción de ciclos de parada; riesgos de control: salto retardado, escritura del PC en ID.

□

# Operadores multiciclo y gestión estática de instrucciones

## Operadores multiciclo

**Ejercicio 2.8.** Se dispone de un procesador MIPS con los siguientes operadores multiciclo:

- Sumador/Restador segmentado lineal. Lat= 2, IR= 1
- Multiplicador segmentado lineal. Lat= 3, IR= 1
- Divisor convencional. Lat= 4, IR=  $\frac{1}{4}$

Los riesgos estructurales y de datos se detectan en la fase ID, insertando tantos ciclos de parada como sean necesarios. También se utilizan cortocircuitos.

Mostrar el diagrama de ejecución del siguiente fragmento de código, representando las etapas que atraviesan cada una de las instrucciones, utilizando la siguiente notación: IF fase de búsqueda, ID fase de decodificación, EX fase de ejecución monociclo, A1, A2 fases de ejecución del sumador/restador, M1, M2, M3 fases de ejecución del multiplicador, D1, D2, D3, D4 fases de ejecución del divisor, ME fase de acceso a memoria y WB fase de escritura en registros. Las instrucciones multiciclo no realizan la fase ME.

```
L.D      F1, 0(R1)
DIV.D    F4, F0, F1
ADD.D    F2, F3, F4
L.D      F4, 4(R1)
MULT.D   F3, F4, F2
L.D      F5, 8(R1)
```

**Solución:**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L.D F1, 0(R1)	IF	ID	EX	M	WB											
DIV.D F4,F0,F1		IF	ID	ID	D1	D2	D3	D4	WB							
ADD.D F2,F3,F4			IF	IF	ID	ID	ID	ID	A1	A2	WB					
L.D F4,4(R1)					IF	IF	IF	IF	ID	EX	M	WB				
MULT.D F3,F4,F2									IF	ID	ID	M1	M2	M3	WB	
L.D F5,8(R1)										IF	IF	ID	ID	EX	M	WB

□

**Ejercicio 2.9.**

Un procesador ejecuta el siguiente bucle que calcula  $\vec{z} = A\vec{x} + B\vec{y}$ :

```
loop: l.d F0,x(r10)
      l.d F1,y(r11)
      mult.d F4,F2,F0;   F2 contiene A.
      mult.d F5,F3,F1;   F3 contiene B.
      add.d F6,F4,F5
      daddi r14,r14,-1
      daddi r10,r10,8
      daddi r11,r11,8
      s.d F6,z(r12)
      daddi r12,r12,8
      bnez r14,loop
      nop
```

El procesador cuenta con dos bancos de registros para almacenar datos enteros y de coma flotante. Además, para la ejecución de las operaciones en coma flotante, dispone de los siguientes operadores multiciclo:

- Un multiplicador segmentado con  $T_{ev} = 5$  e  $IR = 1$ , con etapas denominadas M1, M2, etc.
- Un sumador no segmentado con  $T_{ev} = 3$  e  $IR = 1/3$ , con etapas denominadas A1, A2, etc.

Las restantes instrucciones se ejecutan utilizando el pipeline clásico de 5 etapas (IF, ID, EX, ME, WB). Los riesgos de datos se resuelven con cortocircuitos e insertando ciclos de parada cuando es necesario. Los saltos condicionales utilizan la técnica *predict-not-taken*. La condición y el destino del salto se calculan durante la etapa ID del salto. Si el salto es efectivo, el PC se actualiza con la nueva dirección destino al final de esta etapa.

Se pide:

1. El diagrama instrucciones-tiempo de la primera iteración del bucle y la primera instrucción de la segunda iteración.
2. Asumiendo que todas las iteraciones son iguales, calcule el CPI medio del bucle para  $n$  iteraciones.
3. Para acelerar la ejecución del bucle, se plantean dos opciones: a) sustituir el sumador no segmentado por otro segmentado con  $T_{ev} = 3$  e  $IR = 1$ , o bien b) sustituir el multiplicador segmentado por otro no segmentado con  $T_{ev} = 2$  e  $IR = 1/2$ . ¿Cuál de ambas opciones es la más adecuada para reducir el tiempo de ejecución del bucle? Razone la respuesta.

### Solución:

1. Diagrama instrucciones-tiempo de la primera iteración del bucle:

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21
l.d F0,x(r10)	IF	ID	EX	ME	WB																
l.d F1,y(r11)		IF	ID	EX	ME	WB															
mult.d F4,F2,F0			IF	ID	M1	M2	M3	M4	M5	WB											
mult.d F5,F3,F1				IF	ID	M1	M2	M3	M4	M5	WB										
add.d F6,F4,F5					IF	ID	ID	ID	ID	ID	A1	A2	A3	WB							
daddi r14,r14,-1						IF	IF	IF	IF	IF	ID	EX	ME	WB							
daddi r10,r10,8											IF	ID	EX	ME	WB						
daddi r11,r11,8												IF	ID	EX	ME	WB					
s.d F6,z(r12)												IF	ID	EX	ME	WB					
daddi r12,r12,8													IF	ID	EX	ME	WB				
bnez r14,loop														IF	ID	EX	ME	WB			
nop															IF	X					
l.d F0,x(r10)																IF	ID	EX	ME	WB	

2. CPI medio para  $n$  iteraciones:

La segunda iteración comienza en el ciclo 17, luego una iteración dura 16 ciclos. Teniendo en cuenta que cada iteración ejecuta 11 instrucciones, el CPI medio resulta  $CPI = 16/11 = 1,45$

3. Mejora del tiempo de ejecución:

La opción a) no tendrá ningún efecto en el tiempo de ejecución puesto que el sumador no introduce ningún riesgo estructural en el bucle y la latencia del sumador alternativo es la misma.

Opción b):

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21
l.d F0,x(r10)	IF	ID	EX	ME	WB																
l.d F1,y(r11)		IF	ID	EX	ME	WB															
mult.d F4,F2,F0			IF	ID	M1	M2	WB														
mult.d F5,F3,F1				IF	ID	ID	M1	M2	WB												
add.d F6,F4,F5					IF	IF	ID	ID	A1	A2	A3	WB									
daddi r14,r14,-1						IF	IF	ID	EX	ME	WB										

daddi r10,r10,8	IF ID EX ME WB
daddi r11,r11,8	IF ID EX ME WB
s.d F6,z(r12)	IF ID EX ME WB
daddi r12,r12,8	IF ID EX ME WB
bnez r14,loop	IF ID EX ME WB
nop	IF X
l.d F0,x(r10)	IF ID EX ME WB

La opción b) mejora el tiempo de ejecución puesto que el multiplicador alternativo presenta menor latencia y permite que la suma se lance al operador en el ciclo 9, lo que reduce los ciclos de parada en 2.

□

## Gestión estática de instrucciones

**Ejercicio 2.10.** Se dispone de un procesador compatible con el juego de instrucciones del MIPS. En dicho procesador se ejecuta la siguiente secuencia de código.

```

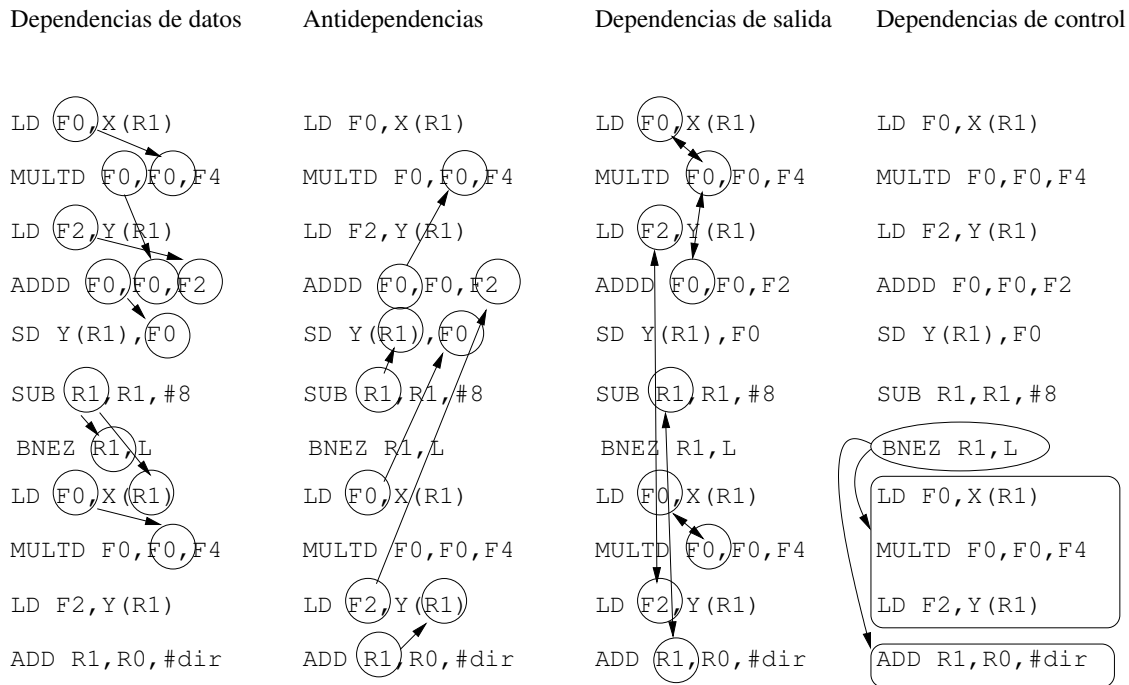
i1      L.D F0,X(R1)
i2      MULT.D F0,F0,F4
i3      L.D F2,Y(R1)
i4      ADD.D F0,F0,F2
i5      S.D F0,Y(R1)
i6      DSUB R1,R1,#8
i7      BNEZ R1,L
i8      L.D F0,X(R1)
i9      MULT.D F0,F0,F4
i10     L.D F2,Y(R1)
i11 L:   DADD R1,R0,#dir

```

Identifica al menos dos dependencias de cada tipo en el fragmento de código anterior.

### **Solución:**

La siguiente figura muestra algunas de las dependencias existentes en la secuencia de código propuesta.



□

### Ejercicio 2.11. Sea el siguiente código en ensamblador del MIPS:

```

loop:  L.D F0, 0(R1)
       MULT.D F0, F0, F10
       ADD.D F0, F0, F11
       S.D F0, 0(R1)
       DADD R1, R1, #8
       BNE R1, R3, loop

```

Dicho código se pretende ejecutar sobre un MIPS en el que las instrucciones enteras atraviesan las siguientes etapas: IF (búsqueda de la instrucción), ID (decodificación de la instrucción, lectura de registros fuente y detección de riesgos), EX (ejecución), M (acceso a memoria) y WB (writeback), mientras que las de coma flotante son: IF, ID, En (ejecución en el operador multiciclo correspondiente) y WB. Los riesgos de datos se resuelven mediante cortocircuitos, insertando en ID los ciclos de parada necesarios y los de control mediante *predict-not taken*, escribiendo el PC en la fase ID.

El procesador funciona a 4 GHz y el CPI de las instrucciones aritméticas enteras es 1.

Las características de las unidades funcionales multiciclo son:

Tipo de operador	Número	Latencia	Tipo
Multiplicación	1	3 ciclos	Segmentada
Suma/resta	1	3 ciclos	Segmentada

1. Identifica una dependencia de datos, una antidependencia, una dependencia de salida y una dependencia de control en el código original.
2. Dibuja el diagrama instrucciones–tiempo de la primera iteración del bucle. Calcula el tiempo de ejecución para  $n$  iteraciones.
3. Muestra el código obtenido tras modificarlo aplicando la técnica del *loop-unrolling*. Sin realizar nuevamente el diagrama instrucciones–tiempo, calcula el nuevo tiempo de ejecución y la aceleración (si la hubiese) con respecto al código original.

### Solución:

#### 1. Dependencias de datos identificadas

- Dependencia de datos. Entre L.D **F0**, 0(R1) y MULT.D **F0**, **F0**, F10.
- Antidependencia. MULT.D **F0**, **F0**, F10 y ADD.D **F0**, F0, F11
- Dependencia de salida. MULT.D **F0**, F0, F10 y ADD.D **F0**, F0, F11
- Dependencia de control. BNE R1, R3, loop y cualquier otra instrucción

#### 2. Diagrama instrucciones–tiempo de la primera iteración del bucle para el procesador MIPS in-order.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
loop: L.D F0, 0(R1)	IF	ID	EX	M	WB												
MULT.D F0, F0, F10		IF	ID	ID	M1	M2	M3	WB									
ADD.D F0, F0, F11			IF	IF	ID	ID	ID	A1	A2	A3	WB						
S.D F0, 0(R1)					IF	IF	IF	ID	ID	EX	M						
DADD R1, R1, #8								IF	IF	ID	EX	M	WB				
BNE R1, R3, loop										IF	ID	ID	EX	M	WB		
<desc>										IF	IF	--	--	--			
loop: L.D F0, 0(R1)												IF	ID	EX	M	WB	

Notad que si la latencia de salto es de una instrucción, la dependencia de datos DADD/BNE (via R1) provoca un ciclo de parada para poder aplicar el cortocircuito M→ID en el ciclo 12 y el fallo en la predicción provoca la cancelación de una instrucción siguiente (que aparece indicada como desc en el diagrama).

Cada iteración consume 12 ciclos de reloj, por lo que el tiempo de ejecución es  $T(n) = 12n$

#### 3. Código tras aplicar loop-unrolling.

Como se insertan dos ciclos de parada para resolver la dependencia de datos entre MULT.D y ADD.D, hay que desenrollar 3 veces. El bucle resultante quedaría:

```
loop:  L.D F0, 0(R1)
      L.D F1, 8(R1)
      L.D F2, 16(R1)
      MULT.D F0, F0, F10
      MULT.D F1, F1, F10
      MULT.D F2, F2, F10
      ADD.D F0, F0, F11
      ADD.D F1, F1, F11
      ADD.D F2, F2, F11
      S.D F0, 0(R1)
      S.D F1, 8(R1)
      S.D F2, 16(R1)
      DADD R1, R1, #24
      BNE R1, R3, loop
```

Ahora ya no es necesario insertar ciclos de parada. Como hay 15 instrucciones por iteración, teniendo en cuenta que se realizan un tercio de las iteraciones originales, el tiempo de ejecución será:

$$T(n) = \frac{15n}{3}$$

y la mejora sobre el código original es

$$S = \frac{12n}{15n/3} = 2,4$$

□



## Predicción dinámica de saltos

**Ejercicio 2.12.** Un procesador dispone de un predictor dinámico de saltos del tipo BTB (*Branch Target Buffer*) que obtiene su predicción en la fase de búsqueda de la instrucción. La dirección y condición de salto se calcula en la 3ª fase del ciclo de instrucción. La probabilidad de que un salto se encuentre en la tabla es del 80 % y de que se acierte en la predicción es del 90 %. Los saltos son efectivos en el 60 % de los casos. Se pide:

1. Mostrar las instrucciones que se buscarían después de la de salto y sus fases para las opciones siguientes:

- No hay una entrada en la tabla de predicción y el salto **no salta**.
- No hay una entrada en la tabla de predicción y el salto **salta**.
- El predictor predice que **no salta** y finalmente el salto **no salta**.
- El predictor predice que **no salta** y finalmente el salto **sí salta**.
- El predictor predice que **sí salta** y finalmente el salto **no salta**.
- El predictor predice que **sí salta** y finalmente el salto **sí salta**.

Las instrucciones se representarán como: **I.Salto**, instrucción de salto, **PC+i** ( $i = 1, 2, \dots$ ), instrucciones posteriores a la instrucción de salto, **Dest**, instrucción destino del salto y **Dest+i** ( $i = 1, 2, \dots$ ), instrucciones posteriores a la instrucción destino del salto. Las fases del ciclo de instrucción como **F1**, **F2**, etc...

2. Calcula el CPI medio de las instrucciones de salto.

3. Supóngase que se puede modificar el diseño del BTB de dos formas. La primera consiste en aumentar el número de entradas de la tabla, de forma que aloje el 90 % de los saltos ejecutados. La segunda es utilizar un predictor de dos bits de manera que se aumente la precisión de la predicción hasta el 95 %. ¿Cual de las dos es la que permite reducir los CPI de las instrucciones de salto?

### Solución:

1. Mostrar las instrucciones que se buscarían después de la de salto y sus fases para las opciones siguientes:

- No hay una entrada en la tabla de predicción y el salto **no salta**.

	p		d,c						
I. Salto	F1	F2	F3	F4	F5	F6			
PC + 1		F1	F2	F3	F4	F5	F6		
PC + 2			F1	F2	F3	F4	F5	F6	
PC + 3				F1	F2	F3	F4	F5	F6

→ 0 ciclos de penalización.

- No hay una entrada en la tabla de predicción y el salto **salta**.

	p		d,c						
I. Salto	F1	F2	F3	F4	F5	F6			
PC + 1		F1	F2	X					
PC + 2			F1	X					
Dest				F1	F2	F3	F4	F5	F6

→ 2 ciclos de penalización.

- El predictor predice que **no salta** y finalmente el salto **no salta**.

	p		d,c						
I. Salto	F1	F2	F3	F4	F5	F6			
PC + 1		F1	F2	F3	F4	F5	F6		
PC + 2			F1	F2	F3	F4	F5	F6	
PC + 3				F1	F2	F3	F4	F5	F6

→ 0 ciclos de penalización.

- El predictor predice que **no salta** y finalmente el salto **sí salta**.

	p		d,c						
I. Salto	F1	F2	F3	F4	F5	F6			
PC + 1		F1	F2	X					
PC + 2			F1	X					
Dest				F1	F2	F3	F4	F5	F6

→ 2 ciclos de penalización.

- El predictor predice que **sí salta** y finalmente el salto **no salta**.

	p		d,c						
I. Salto	F1	F2	F3	F4	F5	F6			
Dest		F1	F2	X					
Dest + 1			F1	X					
PC + 1				F1	F2	F3	F4	F5	F6

→ 2 ciclos de penalización.

- El predictor predice que **sí salta** y finalmente el salto **sí salta**.

	p		d,c						
I. Salto	F1	F2	F3	F4	F5	F6			
Dest		F1	F2	F3	F4	F5	F6		
Dest + 1			F1	F2	F3	F4	F5	F6	
Dest + 2				F1	F2	F3	F4	F5	F6

→ 0 ciclos de penalización.

## 2. Calcula el CPI medio de las instrucciones de salto.

La tabla siguiente muestra los casos posibles, así como la probabilidad de que se produzcan y su CPI (1 de la instrucción de salto más los posibles ciclos de parada:

Caso	Probabilidad	CPI
1	$0.2 \cdot 0.4$	1
2	$0.2 \cdot 0.6$	3
3	$0.8 \cdot 0.4 \cdot 0.9$	1
4	$0.8 \cdot 0.6 \cdot 0.1$	3
5	$0.8 \cdot 0.4 \cdot 0.1$	3
6	$0.8 \cdot 0.6 \cdot 0.9$	1
CPI		1.4

Casos:

1. No hay una entrada en la tabla de predicción y el salto **no salta**.
2. No hay una entrada en la tabla de predicción y el salto **salta**.
3. El predictor predice que **no salta** y finalmente el salto **no salta**.
4. El predictor predice que **no salta** y finalmente el salto **sí salta**.
5. El predictor predice que **sí salta** y finalmente el salto **no salta**.
6. El predictor predice que **sí salta** y finalmente el salto **sí salta**.

El CPI medio se calcula obteniendo la media aritmética ponderada.

3. Supóngase que se puede modificar el diseño del BTB de dos formas. La primera consiste en aumentar el número de entradas de la tabla, de forma que aloje el 90 % de los saltos ejecutados. La segunda es utilizar un predictor de dos bits de manera que se aumente la precisión de la predicción hasta el 95 %. ¿Cual de las dos es la que permite reducir los CPI de las instrucciones de salto?

Las tablas siguientes muestran los nuevo valores de la probabilidad de cada caso para cada una de las opciones:

- Se aumenta el tamaño de la tabla de predicción.

Caso	Probabilidad	CPI
1	$0.1 \cdot 0.4$	1
2	$0.1 \cdot 0.6$	3
3	$0.9 \cdot 0.4 \cdot 0.9$	1
4	$0.9 \cdot 0.6 \cdot 0.1$	3
5	$0.9 \cdot 0.4 \cdot 0.1$	3
6	$0.9 \cdot 0.6 \cdot 0.9$	1
CPI		1.3

- Se aumenta la precisión en la predicción.

Caso	Probabilidad	CPI
1	$0.2 \cdot 0.4$	1
2	$0.2 \cdot 0.6$	3
3	$0.8 \cdot 0.4 \cdot 0.95$	1
4	$0.8 \cdot 0.6 \cdot 0.05$	3
5	$0.8 \cdot 0.4 \cdot 0.05$	3
6	$0.8 \cdot 0.6 \cdot 0.95$	1
CPI		1.32

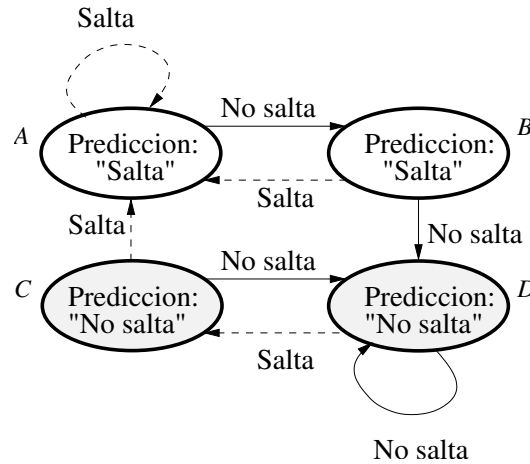
La primera opción es ligeramente mejor que la segunda.

□

**Ejercicio 2.13.** Se dispone de un procesador con un juego de instrucciones similar al MIPS con una unidad de ejecución segmentada con las siguientes etapas:

- **IF** Búsqueda de la instrucción.
- **ID** Decodificación de la instrucción y lectura de registros.
- **ALU** Cálculo de la dirección de destino del salto y de la dirección de acceso a memoria.
- **MEM** Acceso a memoria.
- **EX1** Primera fase de ejecución y cálculo de la condición de salto.
- **EX2** Segunda fase de ejecución.
- **WB** Escritura en registros.

Se desea evaluar dos esquemas de predicción de saltos para su implementación en el procesador. Los predictores que se desean evaluar son: un *Branch Prediction Buffer* y un *Branch Target Buffer*, los cuales ofrecen su predicción al final de la etapa ID. Ambos mecanismos están implementados con 4 entradas en el *buffer* y utilizan un predictor de 2 bits, cuyo funcionamiento se ilustra en la figura:



Para la evaluación de los mecanismos de predicción se utiliza un programa de prueba, del que se muestra un fragmento a continuación:

Dirección	Instrucciones	Dirección	Instrucciones
...		...	
0x03	add r1, r0, r0	0x10	add r2, r2, #1
lfor:		0x11	slt r4, r2, #3
...		0x12	bnez r4, ldo
0x05	beqz r1, lendif	lbreak:	
...		...	
lendif:		0x15	add r1, r1, #1
...		0x16	slt r6, r1, #2
ldo:		0x17	bnez r6, lfor
0x09	sub r8, r8, r2	0x18	sw z(r0), r8
0x0A	slt r3, r2, #2		
0x0B	seq r4, r1, r0		
0x0C	and r5, r3, r4		
0x0D	beqz r5, lbreak		

Inicialmente el *Branch Prediction Buffer* contiene todas las entradas en estado “D”, y el *Branch Target Buffer* tiene todas las entradas vacías. Cuando se añade una nueva entrada en el *Branch Target Buffer*, su estado será “A” si el salto ha sido efectivo y “D” si el salto no ha sido efectivo.

Las estadísticas finales de la ejecución del programa de prueba son las siguientes: el 15 % de las instrucciones son saltos condicionales, el 60 % de los saltos condicionales son efectivos (saltan), el *Branch Prediction Buffer* acierta en la predicción el 75 % de los casos, y el *Branch Target Buffer* acierta en la predicción el 90 %, incluyendo los casos en los que no hay una entrada en la tabla (los cuales se predicen como “no salta”).

Se solicita:

1. Realizar una traza de la ejecución del fragmento de código hasta que se complete la instrucción “sw z(r0), r8”. Se deberá mostrar el contenido de las entradas de ambos predictores después de cada uno de los saltos (“beqz r1, lendif”, “beqz r5, lbreak”, “bnez r4, ldo” y “bnez r6, lfor”). Se deberán utilizar las etiquetas para anotar las direcciones de destino.

Instrucción de salto beqz r1, lendif. (r1 = 0)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto beqz r5, lbreak. (r5 = 1)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto bnez r4, ldo. (r4 = 1)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto beqz r5, lbreak. (r5 = 1)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto bnez r4, ldo. (r4 = 1)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto beqz r5, lbreak. (r5 = 0)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto bnez r6, lfor. (r6 = 1)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto beqz r1, lendif. (r1 = 1)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto beqz r5, lbreak. (r5 = 0)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

Instrucción de salto bnez r6, lfor. (r6 = 0)

**BPB**

Índice	Estado
00	
01	
10	
11	

**BTB**

Índice	Dir. destino	Estado

- Analizar el comportamiento de la BTB cuando se equivoca en la predicción, indicando qué instrucciones son las que se ejecutan en los ciclos siguientes al salto. Se deberá indicar el número ciclos de ejecución perdidos. Las instrucciones canceladas se representarán con una **X** en el ciclo correspondiente. Las instrucciones siguientes al salto se representarán como **pc+1**, **pc+2**, ... y las instrucciones de destino del salto como **dest**, **dest+1**, etc.
- Calcular el número medio de ciclos por instrucción (CPI) para el programa de prueba utilizando la BTB. Suponer que las instrucciones que no son saltos se ejecutan con CPI=1.

**Solución:**

- Traza de la ejecución.

Instrucción de salto 0x05 (00000101) beqz r1, lendif. (r1 = 0) **Salta**

**BPB**

Índice	Estado
00	D
01	C
10	D
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A

Instrucción de salto 0x0D (00001101) beqz r5, lbreak. (r5 = 1) **No salta**

**BPB**

Índice	Estado
00	D
01	D
10	D
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A
0x0D	lbreak	D

Instrucción de salto 0x12 (00010010) bnez r4, ldo. (r4 = 1) **Salta**

**BPB**

Índice	Estado
00	D
01	D
10	C
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A
0x0D	lbreak	D
0x12	ldo	A

Instrucción de salto 0x0D (00001101) beqz r5, lbreak. (r5 = 1) **No salta**

**BPB**

Índice	Estado
00	D
01	D
10	C
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A
0x0D	lbreak	D
0x12	ldo	A

Instrucción de salto 0x12 (00010010) bnez r4, ldo. (r4 = 1) **Salta**

**BPB**

Índice	Estado
00	D
01	D
10	A
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A
0x0D	lbreak	D
0x12	ldo	A

Instrucción de salto 0x0D (00001101) beqz r5, lbreak. (r5 = 0) **Salta**

**BPB**

Índice	Estado
00	D
01	C
10	A
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A
0x0D	lbreak	C
0x12	ldo	A

Instrucción de salto 0x17 (00010111) bnez r6, lfor. (r6 = 1) **Salta**

**BPB**

Índice	Estado
00	D
01	C
10	A
11	C

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	A
0x0D	lbreak	C
0x12	ldo	A
0x17	lfor	A

Instrucción de salto 0x05 (00000101) beqz r1, lendif. (r1 = 1) **No salta**

**BPB**

Índice	Estado
00	D
01	D
10	A
11	C

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	B
0x0D	lbreak	C
0x12	ldo	A
0x17	lfor	A

Instrucción de salto 0x0D (00001101) beqz r5, lbreak. (r5 = 0) **Salta**

**BPB**

Índice	Estado
00	D
01	C
10	A
11	C

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	B
0x0D	lbreak	A
0x12	ldo	A
0x17	lfor	A

Instrucción de salto 0x17 (00010111) bnez r6, lfor. (r6 = 0) **No salta**

**BPB**

Índice	Estado
00	D
01	C
10	A
11	D

**BTB**

Índice	Dir. destino	Estado
0x05	lendif	B
0x0D	lbreak	A
0x12	ldo	A
0x17	lfor	B



## 2. Comportamiento de los predictores en caso de fallo en la predicción.

### ■ Para el BTB:

Predice que **no** salta y **sí** salta.

		p	d		c								
I. Salto	IF	ID	ALU	ME	EX1	EX2	WB						
PC+1		IF	ID	ALU	ME	X							
PC+2			IF	ID	ALU	X							
PC+3				IF	ID	X							
PC+4					IF	X							
DEST						IF	ID	ALU	ME	EX1	EX2	WB	

Se producen **cuatro** ciclos de parada.

Predice que **sí** salta y **no** salta.

		p	d		c								
I. Salto	IF	ID	ALU	ME	EX1	EX2	WB						
PC+1		IF	X										
DEST			IF	ID	ALU	X							
DEST+1				IF	ID	X							
DEST+2					IF	X							
PC+1						IF	ID	ALU	ME	EX1	EX2	WB	

Se producen **cuatro** ciclos de parada.

## 3. CPI medio.

Faltaría obtener la penalización en caso de acierto en la predicción. En caso de que el salto no sea efectivo, no hay penalización. En caso contrario, el BTB busca la instrucción correcta tras acceder al predictor (fase ID), introduciendo sólo un ciclo.

### ■ Para el BTB:

Predicción	Condición	Probabilidad	Penalización
No	No	$0.9 \cdot 0.4$	0
No	Si	$0.1 \cdot 0.6$	4
Si	No	$0.1 \cdot 0.4$	4
Si	Si	$0.9 \cdot 0.6$	1

El número medio de ciclos de parada de penalización es: 0.94 ciclos, y el CPI medio de:  
 $1 + 0,15 \cdot 0,94 = 1,14$  ciclos.

□

## Ejercicio 2.14.

Considere el código de la función `ones`, que devuelve (en `$v0`) el número de bits a 1 contenidos en el argumento (`$a0`). El procedimiento es obtener el LSB del argumento (con `andi $t1, $a0, 1`), incrementar la cuenta si `LSB==1` (con `daddi $v0, $v0, 1`) y desplazar el argumento una posición hacia la derecha (con `dsrl $a0, $a0, 1`). Repitiendo 64 veces estas operaciones, se hace el trabajo especificado.

```

ones:  li $t0, 64          # Número de iteraciones
       li $v0, 0          # Cuenta inicial = 0
loop:  andi $t1, $a0, 1    # $t1 = LSB de $a0
       beqz $t1, next      # Si (LSB!=0 )
       daddi $v0, $v0, 1    # $v0++ /* Cuenta LSB=1 */
       dsrl $a0, $a0, 1
       
```

```

next:  dsrl $a0,$a0,1    # Desplaza $a0 a la derecha 1 bit
        daddi $t0,$t0,-1 # Contador de iteraciones restantes
        bgtz $t0,loop    # Siguiente iteración
        jr $ra           # Retorno de función

```

Note que el código contiene tres instrucciones de salto:

`beqz $t1,next` es efectiva cada vez que `LSB==0`

`bgtz $t0,loop` cierra el bucle

`jr $ra` es incondicional y vuelve al punto desde donde se llamó a la función.

El procesador donde se ejecuta está segmentado en las 5 etapas habituales, escribe el PC en la etapa *EX* (es decir, la latencia de salto es de dos ciclos) y aplica predicción de salto. Note que no se produce ningún ciclo de parada por conflictos estructurales ni de datos.

Calcule cuántas instrucciones ejecuta el procesador, cuántos ciclos de parada inserta y cuál es el tiempo de ejecución (en ciclos de reloj) de la función si:

1. El procesador aplica *predict-not taken* y `$a0=-1` (0xFFF...FFF).
2. El procesador cuenta con un BTB sólo para los saltos condicionales, con un predictor de 1 bit, que permite predecir la dirección de salto en la etapa *IF* y aplica *predict-not taken* al retorno de función `jr $ra`. El programa llama a `ones` por primera vez con `$a0=-1` (0xFFF...FFF)
3. El BTB del apartado 2 se ha perfeccionado con un predictor de 2 bits con histéresis. El programa llama a `ones` por primera vez con `$a0=0x8080808080808080`

### **Solución:**

Llamar a `ones` provoca la ejecución de  $3 + 5 \times 64 + n = 323 + n$  instrucciones donde  $n$  es el número de unos contenidos en el argumento.

1. *Predict-not taken*, `$a0=-1`

El salto `beqz` nunca es efectivo. Por lo tanto,

$$I = 323 + 64 = 387 \text{ instrucciones ejecutadas}$$

$$P = 64 \text{ saltos efectivos} \times 2 \text{ paradas de penalización} = 128 \text{ ciclos de parada}$$

$$T = 387 + 128 = 515 \text{ ciclos totales}$$

2. BTB con predictor de 1 bit, `$a0=-1`

Para `beqz` la predicción será siempre *not taken* (la primera iteración porque no hay historia, las demás porque el salto no será efectivo). No se produce ningún ciclo de parada.

Para `bgtz` la primera predicción será *not taken* (no hay historia), pero las predicciones posteriores siempre serán *taken*. La predicción fallará la primera iteración y la última.

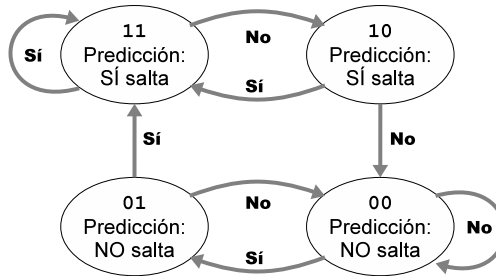
El salto `jr $ra`, al quedar fuera del BTB, siempre sufrirá penalización

$$I = 323 + 64 = 387 \text{ instrucciones ejecutadas}$$

$$P = 3 \text{ predicciones no acertadas} \times 2 \text{ paradas de penalización} = 6 \text{ ciclos de parada}$$

$$T = 387 + 6 = 393 \text{ ciclos totales}$$

3. BTB con predictor de 2 bits, `$a0=0x8080...` El autómata del predictor de dos bits es este:



Con esta ruta de datos con BTB, si una instrucción de salto no tiene historia la predicción solo puede ser *not taken*. Cuando se ejecuta por primera vez, si el salto resulta no efectivo no tendrá penalización y la instrucción entra en el BTB con el estado 00; mientras que si resulta efectivo se aplica la penalización y la instrucción entra en el BTB con el estado 11. Seguidamente, y mientras el BTB mantiene el historico de la instrucción, el estado almacenado seguirá el autómata.

Con `beqz`, la predicción fallará en  $b_0$  porque no tiene historia y entrará con el estado 11; en  $b_1$  acertará y se quedará en el estado 11. De este estado saldrá solo en las 8 iteraciones en que  $b_i = 1$  ( $i = 3, 7, 11, \dots$ ) para pasar al estado 10, pero siempre vuelve de nuevo al estado 11 en la iteración siguiente, donde  $b_i = 0$ . Por tanto, una vez insertada la instrucción de salto en el BTB, la predicción será siempre *taken* y fallará en los 8 casos donde  $b_i = 1$ . Total: 9 penalizaciones

Con `bgtz`, la predicción fallará en  $b_0$  por falta de historia, después acertará siempre gracias a la regularidad del comportamiento y volverá a fallar a la salida del bucle. Total: 2 penalizaciones

Con la penalización de `jr`, habrá un total de 12.

$$I = 323 + 8 = 331 \text{ instrucciones ejecutadas}$$

$$P = 12 \text{ predicciones no acertadas} \times 2 \text{ paradas de penalización} = 24 \text{ ciclos de parada}$$

$$T = 331 + 24 = 355 \text{ ciclos totales}$$

□

## Gestión dinámica de instrucciones y ejecución especulativa

### Especulación hardware

#### Ejercicio 2.15.

Un procesador MIPS aplica planificación dinámica de instrucciones con especulación para todas las instrucciones. Las instrucciones siguen las siguientes fases:

- **IF** Búsqueda de la instrucción.
- **IS** Decodificación de la instrucción y lanzamiento a ejecución siguiendo el método de Tomasulo con especulación.
- **Ei** Ejecución en el operador correspondiente.
- **WB** Fase de transferencia de resultados por el bus interno.
- **C** Fase de confirmación. Escritura de resultados en el registro destino.

Todas las fases duran un ciclo de reloj, excepto la fase **Ei** cuya duración depende del operador. El procesador dispone de un predictor de saltos del tipo branch target buffer que obtiene la predicción en la fase **IF**.

Las características de las unidades funcionales del procesador son:

Tipo	Unidades	Latencia (ciclos)	Otras
Aritmética entera	1	1	4 estaciones de reserva
Multiplicación FP	1	4	Segmentada, 4 estaciones de reserva
Suma/Resta FP	1	2	Segmentada, 4 estaciones de reserva
Carga/Almac.	1	2	Segmentada 2 tampones carga, 2 tampones almacenamiento

Sobre dicho procesador se pretende ejecutar el siguiente fragmento de código:

```

l.d    f8,d(r0)
l.d    f6,n(r0)
bucle: sub.d  f4,f2,f8
      mult.d f8,f8,f4
      mult.d f6,f6,f4
      lt.d   f8,f0
      bclt   bucle      ; Salta si f8 < f0
      s.d    f6,q(r0)
      ...

```

La instrucción `lt.d f8,f0` se evalúa en la unidad de suma/resta, y escribe su resultado en un registro interno (registro de estado de coma flotante). La instrucción `bclt salto` salta si el registro de estado de coma flotante está a *true*, calculando la dirección de salto y evaluando la condición en la unidad entera.

En el momento de empezar a ejecutar este código, el *reorder buffer* y los operadores se encuentran vacíos, y el banco de registros y la memoria contienen los siguientes valores:

F0	F2	F4	F6	F8	d(r0)	n(r0)
1	2	0	0	0	1	0.25

Obsérvese que, para los datos indicados, sólo debe ejecutarse una vez cada instrucción del bucle. Sin embargo, supóngase que el predictor predice **incorrectamente** que la instrucción `bclt salto` **salta**.

1. Dibuja un diagrama temporal en el que se indique qué fase está ejecutando cada instrucción en cada ciclo, desde el ciclo en que la instrucción `l.d f8,d(r0)` ejecuta la fase **IF** hasta el ciclo en que la instrucción `s.d f6, q(r0)` termina completamente.
2. Muestra el estado del *reorder buffer*, de las estaciones de reserva y tampones y del banco de registros al final del ciclo de reloj en que la instrucción `sub.d f4, f2, f8` de la primera iteración ejecuta la fase **C**.

**Nota:** Para representar las fases de ejecución en el diagrama temporal utilizar: M1, M2, M3 y M4 para las fases de la multiplicación/división, A1, A2 para las fases de la suma/resta, L1, L2 para la unidad de carga/almacenamiento y EX para la unidad entera.

**Solución:**

## Diagrama temporal

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
l.d	f8,d(r0)	IF	IS	L1	L2	WB	C																		
l.d	f6,n(r0)		IF	IS	L1	L2	WB	C																	
sub.d	f4,f2,f8			IF	IS	-	A1	A2	WB	C															
mul.d	f8,f8,f4				IF	IS	-	-	-	M1	M2	M3	M4	WB	C										
mul.d	f6,f6,f4					IF	IS	-	-	-	M1	M2	M3	M4	WB	C									
lt.d	f8,f0						IF	IS	-	-	-	-	-	-	A1	A2	WB	C							
bc1t	bucle							IF	IS	-	-	-	-	-	-	-	-	EX	WB	C					
sub.d	f4,f2,f8								IF	IS	-	-	-	-	-	A1	A2	WB	-	X					
mul.d	f8,f8,f4									IF	IS	-	-	-	-	-	-	-	M1	X					
mul.d	f6,f6,f4										IF	IS	-	-	-	-	-	-	-	X					
lt.d	f8,f0											IF	IS	-	-	-	-	-	-	X					
bc1t	bucle												IF	IS	-	-	-	-	-	X					
sub.d	f4,f2,f8													IF	IS	-	-	-	-	X					
mul.d	f8,f8,f4														IF	IS	-	-	-	X					
mul.d	f6,f6,f4															IF	IS	-	-	X					
lt.d	f8,f0																IF	IS	-	X					
bc1t	bucle																	IF	IS	X					
sub.d	f4,f2,f8																		IF	X					
mul.d	f8,f8,f4																			X					
s.d	f6,q(r0)																				IF	IS	C	L1	L2

### Estado en el ciclo 9:

Estaciones de reserva:							
Nombre	busy	op	Qj	Vj	Qk	Vk	reorder
<i>e1</i>	1	Salto	#6				#7
<i>e2</i>							
<i>e3</i>							
<i>e4</i>							
<i>a1</i>	0	-		2		1.0	#3
<i>a2</i>	1	<	#4			1.0	#6
<i>a3</i>	1	-		2	#4		#8
<i>a4</i>							
<i>m1</i>	1	*		1.0		1.0	#4
<i>m2</i>	1	*		0.25		1.0	#5
<i>m3</i>							
<i>m4</i>							

Reorder buffer:						
Entrada	busy	instr	estado	dest	value	pred
1	0	l.d f8,d(r0)	WB	F8	1.0	
2	0	l.d f6,n(r0)	WB	F6	0.25	
3	0	sub.d f4,f2,f8	WB	F4	1.0	
4	1	mult.d f8,f8,f4		F8		
5	1	mult.d f6,f6,f4		F6		
6	1	lt.d f8,f0		FPSR		
7	1	bc lt bucle		bucle		Salta
8	1	sub.d f4,f2,f8		F4		
9						
10						
11						
12						
13						
14						
15						

Registros:						
	F0	F2	F4	F6	F8	FPSR
busy			1	1	1	1
reorder			#8	#5	#4	#6
valor	1.0	2.0	1.0	0.25	1.0	

Tampones de lectura:					
Nombre	busy	Vj	Qj	disp	reorder
<i>l1</i>	0	0		d	#1
<i>l2</i>	0	0		n	#2

Tampones de escritura:							
Nombre	busy	Vj	Qj	disp	Vk	Qk	conf
<i>s1</i>							
<i>s2</i>							

□

**Ejercicio 2.16.** Un procesador compatible binario con el MIPS64 aplica gestión dinámica de instrucciones con especulación *hardware*. Las instrucciones siguen las siguientes fases:

- **IF** Búsqueda de la instrucción.
- **IS** Decodificación de la instrucción y lanzamiento a ejecución siguiendo el método de Tomasulo con *buffer* de reordenación.
- **Ei** Ejecución en el operador correspondiente.
- **WB** Fase de transferencia de resultados por el bus interno (el resultado transferido no estará disponible hasta el siguiente ciclo).
- **C** Fase de confirmación. Escritura de resultados en el registro destino.

Todas las fases duran un ciclo de reloj, excepto la fase **Ei** cuya duración depende del operador. El procesador dispone de un predictor de saltos perfecto que obtiene la predicción y dirección de destino en la fase **IF**.

Las características de las unidades funcionales del procesador son:

Tipo	Unidades	Latencia (ciclos)	Otras
Aritmética entera	1	1	3 estaciones de reserva (e1..e3)
Suma/Resta FP	1	2	Segmentada, 3 estaciones de reserva (a1..a3)
Multiplicación FP	1	4	Segmentada, 2 estaciones de reserva (m1..m2)
Carga/Almac.	1	2	No segmentada 3 tampones carga (l1..l3), 3 tampones almacenamiento (s1..s3)

Sobre dicho procesador se pretende ejecutar el siguiente fragmento de código:

```

    l.d      f0, A(r0)
salto:  l.d      f2, X(r1)
      mult.d   f4, f2, f0
      l.d      f6, Y(r1)
      sub.d    f4, f4, f6
      s.d      f4, Z(r1)
      dsub     r1, r1, #8
      bnez     r1, salto

```

En el momento de empezar a ejecutar este código, el *reorder buffer* y los operadores se encuentran vacíos. Los valores accedidos de los vectores X e Y se representarán como:  $x_1, x_2, \dots$  e  $y_1, y_2, \dots$  respectivamente. El valor que se encuentra en la dirección  $\text{Mem}[R0 + A]$  es a.

Mostrar el diagrama temporal de ejecución de las dos primeras iteraciones, y el estado de las estructuras de datos al final del ciclo en que el salto “bnez r1, salto” entra por **segunda** vez en la fase de búsqueda (IF). Calcular los MFLOPS que alcanzaría este código en un procesador con una frecuencia de reloj de 150 MHz.

**Nota:** Para representar las fases de ejecución **Ei** en el diagrama temporal utilizar: *EX* para operaciones enteras, *A<sub>i</sub>* para la suma y resta flotante, *M<sub>i</sub>* para la multiplicación flotante y *L<sub>i</sub>* para las cargas y los almacenamientos; donde el subíndice indica el número de ciclos en ejecución (1, 2, ...).

**Solución:**

Estado de la unidad de ejecución en el ciclo 15:

Estaciones de reserva:							
Nombre	busy	op	Qj	Vj	Qk	Vk	reorder
<i>e1</i>	SI	-		$[r1] - 8$		8	#14
<i>e2</i>							
<i>e3</i>							
<i>a1</i>							
<i>a2</i>	SI	-	#10		#11		#12
<i>a3</i>							
<i>m1</i>							
<i>m2</i>	SI	*		$x_2$		a	#10

Buffers de lectura:					
Nombre	busy	Vj	Qj	disp	reorder
<i>l1</i>					
<i>l2</i>	Si	$[r1]$		Y	#11
<i>l3</i>					

Buffers de escritura:							
Nombre	busy	Vj	Qj	disp	Vk	Qk	conf
$s1$	Si	$[r1]$		Z	$a \cdot x1 - y1$		
$s2$	Si	$[r1]-8$		Z		#12	
$s3$							

Reorder buffer:						
Entrada	busy	instr	estado	dest	valor	prediccion
1	NO	l.d f0, A(r0)	W	F0	$a$	
2	NO	l.d f2, X(r1)	W	F2	$x1$	
3	NO	mult.d f4, f2, f0	W	F4	$a \cdot x1$	
4	NO	l.d f6, Y(r1)	W	F6	$y1$	
5	SI	sub.d f4, f4, f6	W	F4	$a \cdot x1 - y1$	
6	SI	s.d f4,Z(r1)	W	s1		
7	SI	dsub r1, r1, #8	W	R1	$[r1] - 8$	
8	SI	bnez r1, salto	W	salto	$salta$	salta
9	SI	l.d f2, X(r1)	W	F2	$x2$	
10	SI	mult.d f4, f2, f0		F4		
11	SI	l.d f6, Y(r1)		F6		
12	SI	sub.d f4, f4, f6		F4		
13	SI	s.d f4,Z(r1)	W	s2		
14	SI	dsub r1, r1, #8		R1		
15						

Banco de registros:					
	F0	F2	F4	F6	R1
Busy		SI	SI	SI	SI
Reorder		#9	#12	#11	#14
Valor	$a$	$x1$	$a \cdot x1$	$y1$	$[r1]$

Diagrama temporal:



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
l.d	f0, A(r0)	IF	I	L1	L2	W	C																				
l.d	f2, X(r1)		IF	I	-	L1	L2	W	C																		
mult.d	f4, f2, f0			IF	I	-	-	-	M1	M2	M3	M4	W	C													
l.d	f6, Y(r1)				IF	I	-	L1	L2	W					C												
sub.d	f4, f4, f6					IF	I	-	-	-	-	-	A1	A2	W	C											
s.d	f4, Z(r1)						IF	I									C	L1	L2								
dsub	r1, r1, #8							IF	I	EX	W							C									
bnez	r1, salto								IF	I	-	EX	-	W					C								
l.d	f2, X(r1)									IF	I	L1	L2	-	W					C							
mult.d	f4, f2, f0										IF	I	-	-	-	M1	M2	M3	M4	W		C					
l.d	f6, Y(r1)											IF	I	L1	L2	-	W				C						
sub.d	f4, f4, f6												IF	I	-	-	-	-	-	-	A1	A2	W	C			
s.d	f4, Z(r1)													IF	I									C	L1	L2	
dsub	r1, r1, #8														IF	I	EX	W							C		
bnez	r1, salto															IF	I	-	EX	-	W					C	

Sin contar la primera instrucción, que no pertenece al bucle principal, el procesador propuesto lanza una instrucción por ciclo, luego tarda 7 ciclos en lanzar las instrucciones pertenecientes a una iteración. En cada iteración del bucle se realizan 2 operaciones de **coma flotante** (multd y subd). Así pues, los MFLOPs que puede alcanzar este procesador en la ejecución del presente código son:

$$R_{\infty} = \frac{2}{7} \text{ op./ciclo} \cdot 150 \cdot 10^6 \text{ ciclos/seg.} = \frac{300}{7} \cdot 10^6 \text{ op./seg.} = 42,86 \text{ MFLOPS}$$

□

### Ejercicio 2.17.

Se pretende utilizar el código correspondiente al bucle  $\vec{y} = a \cdot \vec{x}$  para evaluar las prestaciones de cierto computador. Dicho computador posee un procesador similar al MIPS que aplica especulación hardware para todas las instrucciones. Las instrucciones siguen las siguientes fases:

- **IF** Búsqueda de la instrucción.
- **I** Decodificación de la instrucción y lanzamiento a ejecución siguiendo el método de Tomasulo con especulación.
- **Ei** Ejecución en el operador correspondiente.
- **WB** Fase de transferencia de resultados por el bus común de datos.
- **C** Fase de confirmación. Escritura de resultados en el registro destino. Comprobación de la predicción y cancelación de las intrucciones, en su caso.

Todas las fases duran un ciclo de reloj, excepto la fase **Ei** cuya duración depende del operador. El procesador dispone de un predictor de saltos de **dos** bits, del tipo *branch target buffer*, que obtiene la predicción en la fase **IF**.

Las características de las unidades funcionales del procesador son:

Tipo	Unidades	Latencia (ciclos)	Otras
Aritmética entera	1	1	3 estaciones de reserva
Multiplicaión FP	1	3	Segmentada, 3 estaciones de reserva
Carga/Almac.	1	2	No segmentada 3 tampones carga, 3 tampones almacenamiento

El código que genera el compilador para dicho bucle es el siguiente:

```

loop:  l.d f2,x(r1)
      mult.d f2,f2,f0
      s.d f2, y(r1)
      dsub r1,r1,#8
      bnez r1,loop
      <sgte>

```

El estado de los registros y la memoria, antes de la última iteración, es el siguiente:

Reg.	R1	F0	F2	Mem	x	x+8	y	y+8
Valor	8	3	2	Valor	100	102	10	12

1. Dibuja un diagrama instrucciones-tiempo en el que se muestre la ejecución de las instrucciones que se lanzan en la última iteración del bucle, hasta el ciclo en el que se busca la instrucción <sgte>. Adviértase que el predictor predecirá **incorrectamente** la instrucción `bnez r1, loop` (predice que salta y finalmente no salta). Para representar las fases de ejecución en el diagrama utilizar: M1, M2 y M3 para las fases de la multiplicación, L1 y L2 para la unidad de carga/almacenamiento y EX para la unidad entera. Por simplicidad, supóngase que el ROB y las estaciones de reserva están vacías antes de la última iteración y que no hay ninguna instrucción pendiente de ejecución.
2. En las mismas condiciones que el apartado anterior (*reorder buffer* y estaciones de reserva vacíos y sin instrucciones pendientes de ejecución), muestra cuál sería el estado del *reorder buffer*, el banco de registros y de la memoria al final del ciclo de reloj en el que la instrucción `s.d f2, y(r1)` ejecuta la fase **Commit**.
3. Supóngase ahora que el salto `bnez r1, loop` no se encuentra en la tabla al comenzar la ejecución del bucle, que el valor inicial de R1 es 160. Indicar, justificando la respuesta, el tiempo de ejecución del bucle en ciclos. Considerar que el bucle acaba cuando la instrucción de salto de la última iteración llega a la etapa *Commit*.

### Solución:

1.		1	2	3	4	5	6	7	8	9	10	11	12	13	14
loop:	l.d f2,x(r1)	IF	I	L1	L2	WB	C								
	mult.d f2,f2,f0		IF	I	-	-	M1	M2	M3	WB	C				
	s.d f2,y(r1)			IF	I	-	-	-	-	-	-	C	L1	L2	
	dsub r1,r1,#8				IF	I	EX	WB					C		
	bnez r1,loop					IF	I	-	EX	-	WB			C	
loop:	l.d f2,x(r1)						IF	I	L1	L2	-	WB		X	
	mult.d f2,f2,f0							IF	I	-	-	-	M1	X	
	s.d f2,y(r1)								IF	I	-	-	-	X	
	dsub r1,r1,#8									IF	I	EX	WB	X	
	bnez r1,loop										IF	I	-	X	
loop:	l.d f2,x(r1)											IF	I	X	
	mult.d f2,f2,f0												IF	X	
	s.d f2,y(r1)													X	
	<sgte>														IF

2. Estado al final del ciclo 11.

- Reorder buffer:

Reorder buffer:						
Entrada	busy	instr	estado	dest	valor	prediccion
1	NO	load	WB	f2	102	
2	NO	alu	WB	f2	306	
3	NO	store	WB	s1	-	
4	SI	alu	WB	r1	0	
5	SI	branch	WB	loop	No salta	Salta
6	SI	load	WB	f2	100	
7	SI	alu		f2		
8	SI	store	WB	s2	-	
9	SI	alu		r1		
10	SI	branch				Salta

- Registros y memoria:

Reg.	R1	F0	F2	Mem	x	x+8	y	y+8
Valor	8	3	306	Valor	100	102	10	12
rob	#9		#7	-	-	-	-	-

3. Con el valor de R1 a 160, el bucle realizará 20 iteraciones. Cuando el predictor acierta, el tiempo para realizar una iteración es de 5 ciclos de reloj. Cada vez que se falla en la predicción (una vez, en la última iteración) o no se encuentra el salto en la tabla del predictor (en la primera iteración) se pierden 8 ciclos, por lo tanto:

$$T_{ejecucion} = 20 * 5 + 2 * 8 = 100 + 16 = 116 \text{ ciclos}$$

□

## Lanzamiento múltiple de instrucciones

### Ejercicio 2.18.

Se pretende utilizar el código correspondiente al bucle  $\vec{y} = a\vec{x} + b\vec{y} + c$  para evaluar las prestaciones de cierto computador. Dicho computador posee un procesador superescalar de **dos vías** que aplica planificación dinámica de instrucciones con especulación para todas las instrucciones. Las instrucciones siguen las siguientes fases:

- **IF** Búsqueda de la instrucción.
- **I** Decodificación de la instrucción y lanzamiento a ejecución.
- **Ei** Ejecución en el operador correspondiente.
- **WB** Fase de transferencia de resultados por el bus interno.
- **C** Fase de confirmación. Escritura de resultados en el registro destino.

Todas las fases duran un ciclo de reloj, excepto la fase **Ei** cuya duración depende del operador. El procesador dispone de un predictor de saltos del tipo *branch target buffer* que obtiene la predicción al final de la fase **IF**. El tiempo de transferencia por los buses comunes de datos es de 1 ciclo de reloj.

Las características de las unidades funcionales del procesador son:

Tipo	Unidades	Latencia (ciclos)	Otras
Aritmética entera	2	1	6 estaciones de reserva
Multipliación FP	1	4	Segmentada, 4 estaciones de reserva
Suma/Resta FP	1	2	Segmentada, 4 estaciones de reserva
Carga/Almac.	2	2	2 tampones carga, 2 tampones almacenamiento

El código que genera el compilador para dicho bucle es el siguiente:

```

; Código escalar
loop:  l.d f2,x(r1)
      l.d f4,y(r2)
      mult.d f2,f2,f0
      mult.d f4,f4,f8
      add.d f6,f10,f2
      add.d f6,f6,f4
      s.d f6,y(r2)
      dsub r1,r1,#8
      dsub r2,r2,#8
      bnez r1,loop
      ...

```

El estado de los registros y la memoria es el siguiente:

R1	R2	F0	F2	F4	F6	F8	F10	x+100	x+92	y+150	y+142
100	150	3	2	0	0	0.5	2	100	102	10	12

1. Dibuja un diagrama temporal en el que se muestre la ejecución de las instrucciones que se lanzan en las dos primeras iteraciones. Supóngase que el predictor predice **correctamente** que la instrucción `bnez r1, loop` **salta**. Se deberá indicar qué fase está ejecutando cada instrucción en cada ciclo. Para representar las fases de ejecución en el diagrama temporal utilizar: M1, M2, M3 y M4 para las fases de la multiplicación, A1, A2 para las fases de la suma/resta, L1, L2 para la unidad de carga/almacenamiento y EX para la unidad entera.
2. Muestra el estado del *reorder buffer*, de las estaciones de reserva y tampones, del banco de registros y de memoria al final del ciclo de reloj en que la instrucción `s.d f6,y(r2)` de la primera iteración ejecuta la fase C.

**Solución:**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
<code>l.d f2,x(r1)</code>	IF	I	L1	L2	WB	C																		
<code>l.d f4,y(r2)</code>	IF	I	L1	L2	WB	C																		
<code>mult.d f2,f2,f0</code>			IF	I	-	-	M1	M2	M3	M4	WB	C												
<code>mult.d f4,f4,f8</code>			IF	I	-	-	-	M1	M2	M3	M4	WB	C											
<code>add.d f6,f10,f2</code>				IF	I	-	-	-	-	-	-	A1	A2	WB	C									
<code>add.d f6,f6,f4</code>				IF	I	-	-	-	-	-	-	-	-	A1	A2	WB	C							
<code>s.d f6,y(r2)</code>				IF	I	-	-	-	-	-	-	-	-	-	-	-	C	L1	L2					
<code>dsub r1,r1,#8</code>					IF	I	EX	WB										C						
<code>dsub r2,r2,#8</code>						IF	I	EX	WB									C						
<code>bnez r1,loop</code>						IF	I	-	EX	WB										C				
<code>l.d f2,x(r1)</code>							IF	I	L1	L2	WB									C				
<code>l.d f4,y(r2)</code>								IF	I	-	L1	L2	WB								C			

El estado de la unidad en el ciclo 17 es el siguiente:

Entrada	busy	instr	estado	dest	value	pred
1	no	l.d f2,x(r1)	WB	f2	100	
2	no	l.d f4,y(r2)	WB	f4	10	
3	no	mult.d f2,f2,f0	WB	f2	300	
4	no	mult.d f4,f4,f8	WB	f4	5	
5	no	add.d f6,f10,f2	WB	f6	302	
6	no	add.d f6,f6,f4	WB	f6	307	
7	no	s.d f6, y(r2)	WB	s1	-	
8	sí	dsub r1,r1,#8	WB	r1	92	
9	sí	dsub r2,r2,#8	WB	r2	142	
10	sí	bnez r1,loop	WB	loop	salta	salta
11	sí	l.d f2,x(r1)	WB	f2	102	
12	sí	l.d f4,y(r2)	WB	f4	12	
13	sí	mult.d f2,f2,f0	WB	f2	306	
14	sí	mult.d f4,f4,f8	WB	f4	6	
15	sí	add.d f6,f10,f2		f6		
16	sí	add.d f6,f6,f4		f6		
17	sí	s.d f6, y(r2)	WB	s2		
18	sí	dsub r1,r1,#8	WB	r1	84	
19	sí	dsub r2,r2,#8	WB	r2	134	
20	sí	bnez r1,loop	WB	loop	salta	salta

Registros:								
	R1	R2	F0	F2	F4	F6	F8	F10
busy	sí	sí	no	sí	sí	sí	no	no
reorder	#18	#19		#13	#14	#16		
valor	100	150	3	300	5	307	0.5	2

Memoria de datos:			
	x+100	x+92	y+150
valor	100	102	10

Buffers de lectura:						Buffers de escritura:						
Nombre	busy	Vj	Qj	disp	reorder	Nombre	busy	Vj	Qj	disp	Vk	Qk
<i>l1</i>	No					<i>s1</i>	Si	150		y	307	Si
<i>l2</i>	No					<i>s2</i>	Si	142		y		#16

□

### Ejercicio 2.19.

Se tiene un procesador superescalar compatible binario con el MIPS, capaz de buscar y lanzar a ejecución hasta dos instrucciones por ciclo de reloj. La frecuencia de reloj es 900 MHz, y posee instrucciones enteras y de coma flotante, aplicando gestión dinámica de instrucciones con especulación para todas las instrucciones. Las fases que atraviesan las instrucciones durante su ejecución son:

**IF** Fase de búsqueda de la instrucción.

**I** Fase de lanzamiento de la instrucción.

**En** Fase de ejecución en los operadores correspondientes.

**WB** Fase de transferencia de resultados por el bus interno (el resultado transferido no estará disponible hasta el siguiente ciclo).

**C** Fase de confirmación de la instrucción. Comprobación de las predicciones. Escritura en registros. Lanzamiento de las escrituras a memoria.

Todas las fases duran un ciclo de reloj, excepto la fase En que puede durar varios ciclos, en función del tipo de instrucción. La unidad de ejecución dispone de los siguientes operadores independientes:

Tipo	Unidades	Latencia (ciclos)	Otras
Entera/saltos	2	1	6 tampones
Carga/almac	2	2	Segmentada lineal, 6 tampones
Coma flotante	2	3	Segmentada lineal, 6 tampones

La unidad de carga/almacenamiento calcula la dirección efectiva en el primer ciclo de su ejecución. El procesador resuelve los riesgos de control mediante un predictor perfecto de cero ciclos de reloj de penalización. Cada uno de los bancos de registros tiene cuatro puertos de lectura y dos de escritura. Se necesita un ciclo de reloj para transferir un dato por los buses comunes de datos en la unidad de ejecución.

Sobre este procesador se pretende ejecutar el código obtenido de la compilación de  $\vec{B}(i) := x + A(i)$ :

```
loop:  DSUB R1, R1, #8
        L.D F0, A(R1)
        ADD.D F4, F0, F2
        S.D F4, B(R1)
        BNEZ R1, loop
```

Suponiendo que la segunda iteración es representativa de lo que ocurre en el resto de iteraciones ¿Cuántos MFLOPS alcanza el computador ejecutando ese fragmento de código?

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20			
loop:	DSUB R1,R1,#8	IF	I	EX	WB	C																		
	L.D F0, A(R1)	IF	I	-	-	L1	L2	WB	C															
	ADD.D F4, F0, F2		IF	I	-	-	-	-	A1	A2	A3	WB	C											
	S.D F4,B(R1)		IF	I	-	-	-	-	-	-	-	-	C	L1	L2									
	BNEZ R1,loop		IF	I	EX	WB	-	-	-	-	-	-	C											
	<sigte>		IF	cancelada+																				
loop:	DSUB R1,R1,#8		IF	I	EX	WB	-	-	-	-	-	-	C											
	L.D F0, A(R1)		IF	I	-	-	L1	L2	WB	-	-	-	C											
	ADD.D F4, F0, F2			IF	I	-	-	-	-	A1	A2	A3	WB	C										
	S.D F4,B(R1)			IF	I	-	-	-	-	-	-	-	-	C	L1	L2								
	BNEZ R1,loop				IF	I	EX	WB	-	-	-	-	-	C										
	<sigte>				IF	cancelada+																		
loop:	DSUB R1,R1,#8				IF	I	EX	WB	-	-	-	-	-	C										
	L.D F0, A(R1)				IF	I	-	-	L1	L2	WB	-	-	-	C									
				Bucle=3																				

Cada iteración necesita 3 ciclos de reloj, y realiza una operación en coma flotante:

☐

$$\begin{aligned}\vec{Y} &= \vec{X} * \vec{Y} + b \\ \vec{X} &= a * \vec{X}\end{aligned}$$

El MIPS/S dispone de los operadores multiciclo segmentados mostrados en la tabla:

MIPS/S	
Uds. de carga/almacenamiento, segmentadas	2 operadores, 2 ciclos, 6 buffers de carga 6 buffers de almacenamiento
Sumadores, segmentados	2 operadores, 2 ciclos, 4 buffers
Multiplicadores, segmentados	2 operadores, 3 ciclos, 4 buffers
Ud. enteros/saltos	2 operadores, 1 ciclo, 6 buffers

39

Suponed que el compilador ha ubicado las constantes en los registros f0 y f1 y ha generado el siguiente código para un juego de instrucciones semejante al MIPS:

```

        .text 0x400000
loop:   l.d f2,X(r1)
        l.d f3,Y(r1)
        mul.d f3,f3,f2
        add.d f3,f3,f1
        s.d f3,Y(r1)
        mul.d f4,f0,f2
        s.d f4,X(r1)
        dadd r1,r1,#8
        bne r1,r4,loop
        trap #0 ; Fin de programa

```

Antes de la última iteración, el buffer de reordenación está vacío y el estado de los registros y la memoria es el siguiente:

Reg.	r1	r4	f0	f1	f2	f3	Mem	x+88	x+96	x+104	y+88	y+96	y+104
Valor	96	104	10	20	0.5	-1	Valor	88	96	104	188	196	204

Se pide, justificando la respuesta:

1. Dibuja el diagrama instrucciones-tiempo correspondiente a la **última iteración** del bucle en el MIPS/S. Muestra únicamente las instrucciones que alcanzan la etapa *Commit*. Por simplicidad, supóngase que el ROB y las estaciones de reserva están vacías antes de la última iteración y que no hay ninguna instrucción pendiente de ejecución.
2. Indica el número de ciclos consumido por una iteración cuando el predictor acierta y también cuando falla.
3. Indica en qué ciclo de reloj desde el comienzo de la iteración se consideraría que la variable x está actualizada en la memoria. Indica también en qué ciclo se buscará la instrucción `trap #0` que no se cancela.
4. Considera el ciclo de reloj en el que la instrucción `l.d f2,X(r1)` está en la fase *Commit* y muestra:
  - a) El estado del registro f3 al final de dicho ciclo.
  - b) El estado de los operadores de multiplicación al final de dicho ciclo.
  - c) El contenido de la entrada del ROB que corresponde a la instrucción `bne r1,r4,loop` al final de dicho ciclo.
5. Si el bucle se ejecuta 500 veces, calcula el tiempo de ejecución en ciclos, los CPI obtenidos y la velocidad alcanzada en MFLOPS. Supón que, inicialmente, la instrucción de salto que cierra el bucle no se ha ejecutado antes.

### Solución:

1. Dibuja el diagrama instrucciones-tiempo correspondiente a la **última iteración** del bucle en el MIPS/E. Muestra únicamente las instrucciones que alcanzan la etapa *Commit*.



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
loop:	l.d f2,X(r1)	IF	I	L1	L2	WB	C										
	l.d f3,Y(r1)	IF	I	L1	L2	WB	C										
	mul.d f3,f3,f2		IF	I			M1	M2	M3	WB	C						
	add.d f3,f3,f1		IF	I						A1	A2	WB	C				
	s.d f3,Y(r1)			IF	I								C	L1	L2		
	mul.d f4,f0,f2			IF	I		M1	M2	M3	WB				C			
	s.d f4,X(r1)				IF	I								C	L1	L2	
	dadd r1,r1,#8				IF	I	EX	WB							C		
	bne r1,r4,loop					IF	I	-	EX	-	WB				C		
	trap #0						IF	X									
loop:	l.d f2,X(r1)						IF	I	L1	L2	WB						X
	l.d f3,Y(r1)						IF	I	L1	L2	-	WB					X
	mul.d f3,f3,f2							IF	I			M1	M2	M3	X		
	add.d f3,f3,f1							IF	I						X		
	s.d f3,Y(r1)								IF	I					X		
	mul.d f4,f0,f2							IF	I		M1	M2	M3	WB	X		
	s.d f4,X(r1)								IF	I					X		
	dadd r1,r1,#8								IF	I	EX	WB			X		
	bne r1,r4,loop									IF	I	-	EX	WB	X		
	trap #0									IF	X						
loop:	l.d f2,X(r1)									IF	I	L1	L2	X			
	l.d f3,Y(r1)									IF	I	L1	L2	X			
	mul.d f3,f3,f2										IF			X			
	add.d f3,f3,f1										IF	I		X			
	s.d f3,Y(r1)											IF	I	X			
	mul.d f4,f0,f2											IF	I	X			
	s.d f4,X(r1)												IF	X			
	dadd r1,r1,#8												IF	X			
	bne r1,r4,loop													X			
	trap #0													X			
	bne r1,r4,loop														X		
	trap #0															X	
																	X
																	IF I ...

2. Indica el número de ciclos consumido por una iteración cuando el predictor acierta y también cuando falla.

Si acierta: 5 ciclos

Si falla: 15 ciclos

3. Indica en qué ciclo de reloj desde el comienzo de la iteración se consideraría que la variable x está actualizada en la memoria. Indica también en qué ciclo se buscará la instrucción `trap #0`.

La variable x está actualizada en la memoria al final del ciclo 16

La instrucción `trap #0` se buscará en el ciclo 16

4. Considera el ciclo de reloj en el que la instrucción `l.d f2,X(r1)` está en la fase *Commit* y muestra:

a) El estado del registro f3 al final de dicho ciclo.

Reg.	F3
Valor	196
rob	#4

b) El estado de los operadores de multiplicación al final de dicho ciclo.

Buffer	busy	op	Vj	Qj	Vk	Qk	rob
m1	Si	*	196	–	96	–	#3
m2	Si	*	10	–	96	–	#6

c) El contenido de la entrada del ROB que corresponde a la instrucción `bnez r1, r4, loop`.

Entrada	busy	instr	dest	valor	estado	pred
#9	Si	Salto	loop	–	–	Salta

5. Si el bucle se ejecuta 500 veces, calcula el tiempo de ejecución en ciclos, los CPI obtenidos y la velocidad en MFLOPS.

El predictor de saltos falla una vez al principio del bucle y otra más al final. Al principio del bucle, el salto no está en la tabla, por lo que se predice que “No Salta”. Como es la primera iteración, el salto es efectivo y el predictor falla. Al final del bucle, el salto está en la tabla en estado “Salta”. Como es la última iteración, el salto no es efectivo y el predictor falla. Por lo tanto:

$$T(500) = 498 * 5 + 2 * 15 = 2520 \text{ ciclos}$$

Cada iteración ejecuta 9 instrucciones. Por tanto:

$$CPI = \frac{2520}{9 * 500} = 0,56 \text{ ciclos/instruc.}$$

Cada iteración ejecuta 3 operaciones en coma flotante. Por tanto:

$$MFLOPS = \frac{3 * 500}{2520} = 0,595 \text{ op./ciclo @ 3 GHz} = 1786 \text{ MFLOPS.}$$

□

### Ejercicio 2.21.

El siguiente código ensamblador para un procesador MIPS implementa la operación  $\vec{Z} = a\vec{X} + b\vec{Y}$  condicionada al contenido del vector de máscara  $\vec{M}$ .

```

loop:  LD R3, M(R1)
      BEQZ R3, endif      ; Salta si M[i] == 0
      L.D F2, X(R1)       ; Valor inicial de R1 = 0
      L.D F3, Y(R1)
      MULT.D F3, F1, F3   ; F1 contiene b = 5.0
      MULT.D F2, F0, F2   ; F0 contiene a = 2.5
      ADD.D F4, F2, F3
      S.D F4, Z(R1)
endif: DADD R1, R1, #8
      BNE R1, R2, loop    ; R2 contiene 64 * 8

```

El código se ejecuta sobre un procesador **superescalar de 2 vías** que aplica gestión dinámica de instrucciones con especulación hardware. Las instrucciones atraviesan durante su ejecución las siguientes etapas:

- IF - Búsqueda de instrucciones;
- I - Decodificación y lanzamiento de instrucciones;

- En - Ejecución en el operador correspondiente;
- WB - Transferencia de resultados;
- C - Etapa de confirmación.

Todas las etapas tienen una duración de 1 ciclo, excepto  $E_i$ , cuya duración varía de un operador a otro. Los riesgos de control se solucionan utilizando un predictor dinámico de saltos de tipo *Branch Target Buffer* de 1 bit. El predictor obtiene su predicción al final de la etapa IF y **actualiza su predicción en la etapa de confirmación**. Los bancos de registros poseen 4 puertos de lectura y 2 de escritura. El tiempo de transferencia a través de los buses de datos internos es de 1 ciclo.

Las características de los operadores existentes en el procesador son:

Tipo	Operadores	Latencia	Características
Entero	2	1	8 estaciones de reserva
Carga/Almacenamiento	1	2	Segmentado, 4 buffers de escritura y 4 de lectura
Suma/Resta CF	1	2	Segmentado, 4 estaciones de reserva
Multiplicación CF	1	4	Segmentado, 4 estaciones de reserva

Supongase que el contenido inicial del vector  $\vec{M}$  es 1, 0, 1, 0, .... Así pues, en la primera iteración el salto `BEQZ R3, endif` **efectivamente no saltará**. El estado inicial del predictor es:

Salto	Predicción
<code>BEQZ R3, endif</code>	Salta
<code>BNE R1, R2, loop</code>	Salta

Se solicita:

1. Representar el diagrama instrucciones-tiempo de la primera iteración del bucle (hasta que el salto `BNE R1, R2, loop` ejecuta la etapa Commit). Muestra sólo aquellas instrucciones que quepan en la hoja de respuesta. Para representar las fases de ejecución (En) en el diagrama se utilizará M1, M2, M3 y M4 para las fases de multiplicación, A1 y A2 para las fases de suma/resta, L1 y L2 para las fases de carga/almacenamiento, y E para la fase de ejecución en la unidad entera.
2. Mostrar el estado de los registros F2, F3 y F4 al final de la etapa de confirmación de la instrucción `LD F2, X(R1)` de la primera iteración. El valor inicial de dichos registros es 2, 3 y 4, respectivamente. Mostrar el contenido de los campos `valor` y `reorder`. El contenido del vector  $\vec{X}$  es 100, 108, ... y el de  $\vec{Y}$  es 10, 11, ....

### Solución:

1. Diagrama instrucciones-tiempo de las dos primera iteraciones del bucle.

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
#1	loop: LD R3, M(R1)	IF	I	L1	L2	WB	C																			
#2	BEQZ R3, endif	IF	I	-	-	-	E	WB	C																	
#3	endif: DADD R1, R1, #8		IF	I	E	WB			X																	
#4	BNE R1, R2, loop		IF	I	-	-	E	WB	X																	
#5	loop: LD R3, M(R1)			IF	I	-	L1	L2	X																	
#6	BEQZ R3, endif			IF	I	-	-	-	X																	
#7	endif: DADD R1, R1, #8				IF	I	-	E	X																	
#8	BNE R1, R2, loop				IF	I	-	-	X																	
#9	loop: LD R3, M(R1)					IF	I	-	X																	
#10	BEQZ R3, endif					IF	I	-	X																	
#11	endif: DADD R1, R1, #8						IF	I	X																	
#12	BNE R1, R2, loop							IF	I	X																
	loop: LD R3, M(R1)								IF	X																
	BEQZ R3, endif									IF	X															
	endif: DADD R1, R1, #8										X															

[illegible]

2. Estado de los registros al final de la etapa de confirmación de la instrucción L.D F2, X(R1) (ciclo 14).

Registro	F2	F3	F4
Valor	100	3	4
Reorder	#4	#3	#5

☐

### Ejercicio 2.22.

La figura siguiente muestra el diagramas instrucciones–tiempo simplificado correspondientes a la ejecución de dos aplicaciones A y B sobre un procesador superescalares de 3 vías. Se considera que ocurre un evento de alta latencia si se interrumpe la emisión de instrucciones durante dos o más ciclos:

0	1	2	3	4	5	6	7	8
		A						
A		A					A	A
A	A	A	A			A	A	A

0	1	2	3	4	5	6	7	8
				B				
	B			B	B	B		
B	B			B	B	B	B	B

Con el objeto de aumentar la utilización de los recursos del procesador, se plantea ejecutar múltiples hilos o *threads*.

1. Desde el punto de vista de cómo se comparten los recursos del procesador, explica cuáles son las diferencias principales entre un procesador multihilo de grano fino, de grano grueso y simultáneo *SMT*.
2. Muestra un posible diagrama instrucciones–tiempo, similar a los mostrados, correspondiente a la ejecución de las dos aplicaciones A y B sobre un procesador multihilo de grano fino, de grano grueso y simultáneo *SMT*.

**Solución:**

- Multihilo de grano fino.
  - Cambia de hilo cada ciclo de reloj, habitualmente round-robin.
  - Necesita poca sobrecarga en el cambio de hilo.
  - Oculta eventos de baja y alta latencia, pero puede retardar la ejecución de hilos que no tengan detenciones.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
				A		B							
A			B	A		B	B		B	A		A	
A	B	A	B	A	A	B	B	A	B	A	B	A	B

- Multihilo de grano grueso.
  - Cambia de hilo ante eventos de alta latencia.

- Tolera mayor sobrecarga en el cambio de hilo.
- No oculta eventos de baja latencia, pero no retarda la ejecución de hilos sin detenciones.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
		A							B				
A		A			B		A	A	B	B	B		
A	A	A	A	B	B	A	A	A	B	B	B	B	B

■ *SMT*

- Ejecuta simultáneamente varios hilos, tratando de ocupar todos los recursos ya disponibles en el procesador superescalar.
- El número de hilos en ejecución varía a lo largo del tiempo según el ILP de cada hilo y los recursos disponibles.

0	1	2	3	4	5	6	7	8
B	B	A		B		B	B	B
A	B	A		B	B	B	A	A
A	A	A	A	B	B	A	A	A

□