Actividad 1.1

Ingeniería de Computadores II

Actividad 1.1

Suponiendo que se aplica una mejora a una máquina de tal forma que el rendimiento es 20 veces superior al que tenía y considerando que la mejora únicamente se puede aplicar durante el 20% del tiempo, ¿cuál es la ganancia obtenida?

Como el rendimiento es 20 veces superior, entonces el factor $p = 20$.

La mejora sólo se puede aplicar durante el 20% del tiempo, tenemos: $1 - f = 0,2 \rightarrow f = 0,8$

$$S_p = \frac{p}{1 + f(p-1)} = \frac{20}{1 + 0,8 \cdot (20-1)} = 1,234$$

f: fracción de tiempo donde no se puede aplicar la mejora.

Problema 1 septiembre 2012 reserva.
Igual que actividad 1.2 del libro de texto

Tras añadir un nuevo procesador a un computador se logra un aumento de la velocidad de ejecución en un factor 9. Se observa que tras aplicar esta mejora, el 55% del tiempo de ejecución se está utilizando el nuevo procesador. ¿Qué porcentaje del tiempo de ejecución original se ha reducido gracias a la mejora?

El recurso utilizado mejora en 9 el rendimiento.

Porcentaje del tiempo de ejecución que se aplica la mejora = 55%

Suponiendo que normalizamos el tiempo actual de ejecución a 1 (esto es, aplicando la mejora), el tiempo total de ejecución antes de aplicar la mejora es:

$T_{original} = 0.45 + (0.55*9) = 5.4$

Por lo tanto, la ganancia obtenida aplicando la mejora es del 5,4.

Sustituyendo en la expresión de la ganancia:

$$S_p = \frac{p}{1 + f\,(p-1)} \rightarrow 5,4 = \frac{9}{1 + f\,(9-1)} \rightarrow f = 8,3\%$$

(*f* es la fracción del tiempo donde no se puede aplicar la mejora).

Por lo tanto, el porcentaje de tiempo que se ha convertido al modo rápido es:

(100%-8.3%) = 91.7%

Actividad 1.3

Un procesador sin segmentación necesita 400 nseg. para procesar una instrucción. Con respecto a este procesador, calcular la aceleración que se obtiene en los dos casos siguientes:

a) Un procesador A dotado de una segmentación de 5 etapas, consumiendo cada etapa el mismo tiempo. Cada etapa ocasiona una sobrecarga de 10 nseg. no existiendo ningún tipo de detención en la segmentación.

De acuerdo con el enunciado el tiempo medio de ejecución de una instrucción en el procesador sin segmentar es de 400 nseg. La segmentación de 5 etapas de este apartado se caracteriza por acortar el tiempo medio de ejecución de una instrucción a 90 nseg.:

$$\frac{400\ nseg}{5\ etapas} + 10\ nseg = 90\ nseg$$

Por lo tanto, la aceleración obtenida por la máquina A con respecto a la máquina sin segmentar es 4,45:

$$\frac{400\ nseg}{90\ nseg} = 4,45$$

b) Un procesador b con una segmentación de 5 etapas, consumiendo cada una de ellas 60 nseg., 60 nseg., 80 nseg. y 100 nseg. respectivamente, y siendo la sobrecarga por cada etapa de 10 nseg. un 40% de todas las instrucciones de la segmentación son detenidas durante un ciclo de reloj y un 10% durante dos ciclos.

La etapa más lenta es la que dicta la velocidad de las restantes etapas, por lo que cada etapa consumirá 110 nseg. (100 nseg. más los 10 nseg. de retardo).

De acuerdo con esto, el tiempo medio consumido por una instrucción es:

$$0{,}1 \cdot 330\ nseg + 0{,}4 \cdot 220\ nseg + 0{,}50 \cdot 110\ nseg = 176\ nseg\ .$$

Por lo tanto, la aceleración obtenida por la máquina B con respecto a la máquina sin segmentar es de 2,27:

$$\frac{400\ nseg}{176\ nseg} = 2,27$$

Actividad 1.4

Al diseño A de un procesador se le propone añadir una instrucción ALU que tenga un operando fuente en memoria (el otro se encuentra cargado en un registro en el 25% de las operaciones de la ALU) obteniendo de esta manera el diseño del procesador B. si en este nuevo diseño B aumenta en 1 el número de ciclos de reloj de los saltos y suponiendo que la nueva instrucción consume 2 ciclos, ¿cuál de los dos diseños es el más rápido considerando el recuento de instrucciones de la siguiente tabla?

Operación	Frecuencia	Ciclos/instrucción
ALU	43%	1
Cargas	21%	2
Almacenamiento	12%	2
Saltos	24%	2

CPI original: número medio de ciclos por instrucción.

$$CPI\ original = 0,43 \cdot 1 + 0,21 \cdot 2 + 0,12 \cdot 2 + 0,24 \cdot 2 = 1,57$$

$$Tiempo\ original = 1,57 \cdot Recuento\ original\ de\ instrucciones \cdot Duración\ de\ ciclo$$

Al realizar el recuento en el nuevo sistema hay que tener en cuenta las instrucciones de la ALU y carga que desaparecen a causa de la nueva instrucción, así como las que surgen. Tenemos así:

ALU: 43% − 43% × 25% = 32.25%
Cargas: 21% − 43% × 25% = 10.25%
Almacenamientos: 12 %
Saltos: 24%
Nuevas: 43% × 25% = 10.75%
TOTAL: 89.25%

Para proceder a calcular el CPI necesitamos normalizar el recuento de instrucciones que hemos obtenido. De acuerdo con esto, el factor de normalización es 1 / 0,8925 = 1,1204.

Tendremos los siguientes valores:

ALU: 32.25% × 1.1204 = 36.1329%
Cargas: 10.25% × 1.1204 = 11.4841%
Almacenamientos: 12% × 1.1204 = 13.4448%
Saltos: 24% × 1.1204 = 26.8896%
Nuevas: 10.75% × 1.1204 = 12.0443%
TOTAL: 100%

El nuevo CPI y el nuevo tiempo de ejecución son:

$$CPI\ nuevo = \begin{pmatrix} 0,361329 \cdot 1\ ciclo + 0,114841 \cdot 2\ ciclos + 0,134448 \cdot 2\ ciclos + \\ 0,268896 \cdot 3\ ciclos + 0,120443 \cdot 2\ ciclos \end{pmatrix} = 1,908$$

$$Tiempo\ nuevo\ ejecución = \left(0,8925 \cdot Recuento\ original\ de\ instrucciones\right) \cdot 1,908 \cdot Duración\ de\ ciclo$$
$$= 1,703 \cdot Recuento\ original\ de\ instrucciones \cdot Duración\ de\ ciclo$$

Por lo tanto, el sistema original es un 8,47% más rápido que el nuevo:

$$\frac{1,703 - 1,57}{1,57} \cdot 100 = 8,47\%$$

Actividad 1.5

Suponga que se tiene un procesador segmentado en el que el 20% de las instrucciones son cargas y un 50% de las veces la instrucción siguiente se detiene un ciclo de reloj debido a un riesgo por dependencia de datos. Además, el 10% de las instrucciones de carga producen un fallo de caché que se tarda 4 ciclos en resolver. Ignorando cualquier otro tipo de riesgo, ¿cuántas veces es más rápida la segmentación ideal de CPI = 1 que esta nueva segmentación?

La máquina ideal será más rápida según el cociente entre el CPI de la máquina con segmentación ideal que es 1, y el CPI de la máquina cuyas características se explican en el enunciado. Teniendo en cuenta el riesgo por dependencia de datos, el CPI de la instrucción siguiente a la de carga es:

CPI siguiente instrucción = 1 *ciclo* + 1 *ciclo* × 50% *detenciones* = 1,5 *ciclos*

Debido a los fallos de la caché, el CPI de una instrucción de carga es:

CPI instrucción de carga = 1 *ciclo* + 4 *ciclo* × 10% = 1,4 *ciclos*

Por lo tanto, el CPI de la máquina segmentada es:

CPI = 60% × 1 *ciclo* + 20% × 1,5 *ciclos* + 20% × 1,4 *ciclos* = 1,18 *ciclos*

De acuerdo con esto, la máquina ideal es un 18% más rápida.

Actividad 1.6

Un procesador sin segmentación necesita 200 nseg. para procesar una instrucción. Con respecto a este procesador, calcular la aceleración que se obtiene en los dos siguientes casos:

a) Un procesador A dotado de una segmentación de 5 etapas, consumiendo cada etapa el mismo tiempo. Cada etapa ocasiona una sobrecarga de 4 nseg., no existiendo ningún tipo de detención en la segmentación

De acuerdo con el enunciado el tiempo medio de ejecución de una instrucción en el procesador sin segmentar es de 200 nseg. La segmentación de 5 etapas de este apartado se caracteriza por acortar el tiempo medio de ejecución de una instrucción a 44 nseg:

- (200 nseg / 5 etapas) + 4 nseg = 44 nseg

Por lo tanto la aceleración obtenida por la máquina A con respecto a la máquina sin segmentar es 4,54:

- 200 nseg / 44 nseg = 4,54

b) Un procesador B con una segmentación de 5 etapas, consumiendo cada una de ellas 30 nseg., 30 nseg., 40 nseg., 50 nseg. y 50 nseg., respectivamente, y siendo la sobrecarga por cada etapa de 4 nseg. un 20% de todas las instrucciones de la segmentación son detenidas durante un ciclo de reloj y un 5% durante dos ciclos.

La etapa más lenta es la que dicta la velocidad de las restantes etapas, por lo que cada etapa consumirá 54 nseg. (50 nseg. más los 4 nseg. de retardo).

El 20% de todas las instrucciones ocasionan una detención de 1 ciclo, por lo que consumen 108 nseg. (2 ciclos * 54). Por otra parte, un 5% ocasiona una detención de 2 ciclos, consumiendo 162 nseg. (3 ciclos * 54).

El resto de las instrucciones, un 75%, no provocan detenciones, empleando sólo un ciclo de reloj (54 nseg.).
De acuerdo con esto, el tiempo medio consumido por una instrucción es:

- 0,2 * 108 nseg. + 0,05 * 162 nseg. + 0,75 * 54 nseg. = 70,2 nseg.

Por tanto, la aceleración obtenida por la máquina B con respecto a la máquina sin segmentar es de:

- 200 nseg. / 70,2 nseg. = 2,85

Actividad 1.8

Dado el siguiente fragmento de código DLX en el que inicialmente la posición de memoria 2000 contiene el valor 0, ¿cuántas referencias de datos a memoria se efectuarán e instrucciones se ejecutarían?

```
i1:  LDR1,1500(R0)
i2:  LDR2,2000(R0)
i3:  LDR3,500(R2)
i4:  ADD R4,R3,R1
i5:  SD0(R2),R4
i6:  ADDIR2,R2,#4
i7:  SD2000(R0),R2
i8:  SUBIR4,R2,#400
i9:  BENZ 4,i2
```

Bucle de i2 a i9.

El número total de instrucciones ejecutadas es:

1 instrucción para la preparación del bucle
100 * 8 instrucciones del cuerpo del bucle = 800 instrucciones
Total = 801 instrucciones ejecutadas

El total de referencias a datos en memoria es:

1 vez para la carga del valor de R1 en un registro
4 veces por iteración: carga de R2 y R3 (2 LD) y almacenamiento de R4 y R2 (2 SD)
Total = 1 + 100 * 4 = 401 referencias a datos en memoria

Justificación de las 100 veces que se ejecuta el bucle:

Inicialmente, en la instrucción i2, R2 se carga con el valor 0 ya que en M[2000+R0] = M[2000] = 0.
El valor de R2 se incrementa en 4 bytes en cada i6 y se almacena de nuevo en la posición 2000 (en i7).
A continuación, en i8 se comprueba si el contenido en R2 es 400 y si no lo es se salta de nuevo a i2.
Por lo tanto, de 0 a 400 con incrementos de 4 en 4 son 100 iteraciones del bucle.

Actividad 1.10

En los fragmentos de código situados a continuación:

FRAGMENTO 1	**FRAGMENTO 2**
i1: DIV R1,R2,R3	i1: LD F2,0(R1)
i2: ADD R4,R1,R5	i2: MULT F4,F2,F0
i3: ADD R5,R6,R7	i3: LD F6,0(R2)
i4: ADD R1,R8,R9	i4: ADDD F6,F4,F6
	i5: SD 0(R2),F6
	i6: ADDI R1,R1,#8
	i7: ADDI R2,R2,#8
	i8: SGT R3,R1,#800
	i9: BEQZ R3,i1

- Señale las dependencias de datos y de memoria existentes en ambos fragmentos.

- Analice y explique lo que sucede con el registro R1 en sucesivas iteraciones de la segunda secuencia de código.

Dependencias

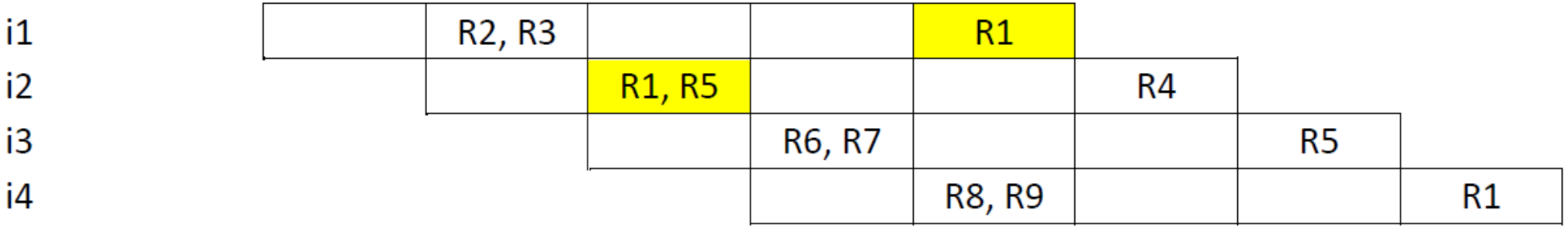
Se va a considerar únicamente el caso de riesgos tipo RAW.

Los riesgos de dependencias de datos se denominan:

Riesgos por dependencia de datos en registros en el caso de que se trate de instrucciones aritmético-lógicas.

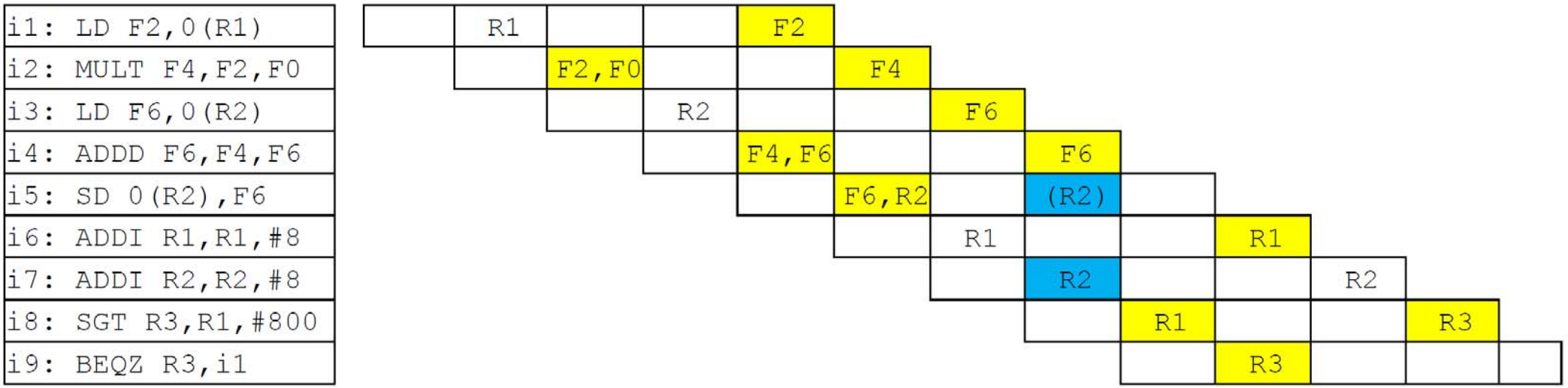
Riesgos por dependencia de datos en memoria cuando suceden con instrucciones de carga y almacenamiento.

Por lo tanto, para el fragmento 1 tenemos:



Riesgo por dependencia de datos en registros, entre i1 e i2 a través de R1.

Para el fragmento 2 tenemos:



Riesgo por dependencias de datos en registros, entre i1 e i2 a través de F2.

Riesgo por dependencias de datos en registros, entre i2 e i4 a través de F4.

Riesgo por dependencias de datos en registros, entre i3 e i4 a través de F6.

Riesgo por dependencias de datos en registros, entre i3 e i5 a través de F6.

Riesgo por dependencias de datos en registros, entre i4 e i5 a través de F6.

Riesgo por dependencias de datos en registros, entre i6 e i8 a través de R1.

Riesgo por dependencias de datos en registros, entre i8 e i9 a través de R3.

Riesgo por dependencias de datos en memoria, entre i5 e i7 a través de R2.

Análisis

En realidad no sé si realmente entiendo bien la pregunta, es decir, no sé si lo que se pide es la evolución del R1 o por el contrario se pide comentar que ocurre con ese registro en temas de dependencia de datos.

Para el primer caso, tenemos que:

En i1, R1 es accedido por una instrucción de carga.

En i6, el contenido de R1 es aumentado en 8 unidades, mediante una instrucción aritmética.

En i8 se compara R1 con el literal 800, mediante una instrucción aritmética.

Para el segundo caso:

En i1 e i6, no existe ningún tipo de dependencia.

En i8 se produce una dependencia RAW a través de R1, debido a que aún la i6 no ha escrito R1.

Actividad 1.12

Dado el código que aparece a continuación y utilizando la segmentación ASG sin ningún tipo de adelantamiento entre etapas:

```
i1: ADD R3,R1,R2
i2: LDR1,0(R4)
i3: SUB R5,R3,R4
i4: ADD R6,R1,R2
i5: SUB R1,R3,R6
i6: SD4(R4),R1
i7: LDR2,4(R4)
i8: SUB R3,R5,R6
```

- Indicar los riesgos por dependencias de datos que se producen al ejecutar el código. Suponer que las escrituras se producen en la primera mitad de la etapa WB y que las lecturas en ID se producen en la segunda mitad de la etapa.
- Compruebe si la siguiente afirmación es cierta: La instrucción `i7` no carga el valor previamente almacenado por la instrucción `i6` en la posición de memoria `M[R4+4]`. Indicar la razón de ello tanto en caso afirmativo como negativo.
- Reorganizar el código sin cambiar su efecto añadiendo el mínimo número posible de instrucciones NOP.
- Repetir el primer apartado sobre el código original considerando que se añade a la segmentación de ASG adelantamiento entre las etapas. Señalar el dato y la etapa de cada instrucción que hace uso del adelantamiento y a qué instrucción y etapa se adelanta.

Primer punto, veo que el problema trata principalmente sobre dependencias de datos y adelantamiento entre etapas. Además, el tercer punto parece indicar que la implementación ASG del problema carece de interbloqueo entre etapas. En caso contrario, no serían necesarias las instrucciones NOP.
Primer punto. Riesgos por dependencias de datos que se producen al ejecutar el código, sin ningún tipo de adelantamiento. WB primera mitad, ID segunda.

- Riesgo RAW por R3 entre i1 e i3, de 1 ciclo.
- Riesgo RAW por R1 entre i2 e i4, de 1 ciclo.
- Riesgo RAW por R6 entre i4 e i5, de 2 ciclos.
- Riesgo RAW por R1 entre i5 e i6, de 2 ciclos.

Segundo punto. La instrucción `i7` no carga el valor previamente almacenado por la instrucción i6

No. La instrucción `i7` **sí** carga el valor previamente almacenado por `i6`. Entre una y otra no cambia ni R4 ni el contenido de la posición de memoria `M[4+R4].` La etapa ID de `i7` ocurre al mismo tiempo que la etapa WB de `i4`. Si `i4` fuese "`ADD R4,R1,R2`", y como WB ocurre antes que ID, esta instrucción podría alterar el registro R4 entre las instrucciones `i6` e `i7`:

```
i4: ADD R4,R1,R2
i5: SUB R1,R3,R6
i6: SD  4(R4),R1
i7: LD  R2,4(R4)
```

IF	ID	EX	MEM	**WB**			
	IF	ID	EX	MEM	WB		
		IF	ID	EX	MEM	WB	
			IF	**ID**	EX	MEM	WB
				Nuevo valor de R4			

Tercer punto. Reorganizar con mínimo número de NOPs.

```
i1:  ADD  R3,R1,R2
i2:  LD R1,0(R4)
   NOP
   ;Evita los dos primeros riesgos RAW, i1-i3 e i2-i4
i3:  SUB  R5,R3,R4
i4:  ADD  R6,R1,R2
   NOP
      NOP
   ;Evitan el riego RAW i4-i5
i5:  SUB  R1,R3,R6
      NOP
i8:  SUB  R3,R5,R6
   ;Junto con el NOP previo evitan RAW i5-i6
i6:  SD 4(R4),R1
i7:  LD R2,4(R4)
```

Cuarto punto. Considerar ahora que hay adelantamiento entre etapas.

- Adelantamiento de i1 a i3. Registro R3, de la etapa EX1 a EX3.
- Adelantamiento de i2 a i4. Registro R1, de la etapa MEM2 a EX4.
- Adelantamiento de i4 a i5. Registro R6, de la etapa EX4 a EX5.
- Adelantamiento de i5 a i6. Registro R1, de la etapa EX5 a MEM6.
- Ningún riesgo por dependencias de datos.

Actividad 1.14

Mostrar la evolución de los Registros en coma flotante (FF) y las estaciones de Reserva (RS) para todos los ciclos que sean necesarios en la ejecución del siguiente fragmento de código utilizando el algoritmo de Tomasulo.

```
i1:  ADDD   F2,F0,F6
i2:  MULTD  F4,F0,F2
i3:  ADDD   F2,F2,F6
i4:  MULTD  F6,F2,F4
i5:  ADDD   F4,F4,F6
i6:  ADDD   F6,F2,F4
```

- Considere las siguientes hipótesis de partida:

- Para reducir el número de ciclos máquina se permite que la FLOS distribuya hasta dos instrucciones en cada ciclo según el orden del programa.
- Una instrucción puede comenzar su ejecución en el mismo ciclo en que se distribuye a una estación de reserva.
- La operación **suma** tiene una latencia de **dos ciclos** y la de **multiplicación** de **tres ciclos**.

- Se permite que una instrucción reenvíe su resultado a instrucciones dependientes durante su último ciclo de ejecución. De esta forma una instrucción a la espera de un resultado puede comenzar su ejecución en el siguiente ciclo si se detecta una coincidencia.
- Los valores de etiqueta 01, 02 y 03 se utilizan para identificar las tres estaciones de reserva de la unidad funcional de suma, mientras que 04 y 05 se utilizan para identificar las dos estaciones de reserva de la unidad funcional de multiplicación/división. Estos valores de etiqueta son los ID de las estaciones de reserva.
- Inicialmente, el valor de los registros es F0=4.0, F2=2.5, F4=10.0 y F6=3.5.

Ciclo 1: Se distribuye i1 e i2

i1: ADDD F2,F0,F6

i2: MULTD F4,F0,F2

Se ejecuta RS 01 1/2

No se ejecuta RS 04

FR

	bitOc.	etiqueta	dato
F0			4.0
F2	Si	01	2.5
F4	Si	04	10.0
F6			3.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i1: 01	00	4.0	00	3.5
02				
03				

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i2: 04	00	4.0	01	xx
05				

MULT/DIV

Ciclo 2: Se distribuye i3 e i4

I3: ADDD F2,F2,F6

i4: MULTD F6,F2,F4

Se ejecuta RS 01 2/2

NO se ejecuta RS 02

NO se ejecuta RS 05

Se envía RS 01: 7.5 al CDB

FR

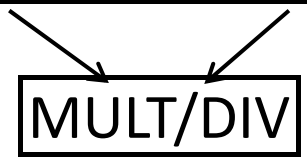
	bitOc.	etiqueta	dato
F0			4.0
F2	Si	02	2.5
F4	Si	04	10.0
F6	Si	05	3.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i1: 01	00	4.0	00	3.5
i3: 02	01	xx	00	3.5
03				

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i2: 04	00	4.0	01	xx
i4: 05	02	xx	04	xx



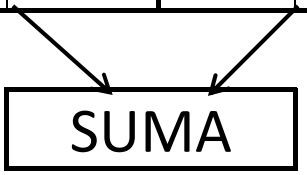
Ciclo 3: Se actualiza el valor de RS 01: 7.5
Se vacía RS 01
Se ejecuta RS 04 1/3
Se ejecuta RS 02 1/2
Se distribuye i5 e i6
i5: ADDD F4,F4,F6
i6: ADDD F6,F2,F4
NO se ejecuta RS 05, 03, 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2	Si	02	2.5
F4	Si	03	10.0
F6	Si	01	3.5

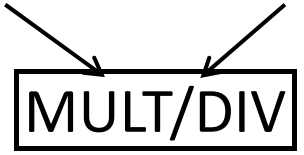
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	xx	03	xx
i3: 02	00	7.5	00	3.5
i5: 03	04	xx	05	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
i2: 04	00	4.0	00	7.5
i4: 05	02	xx	04	xx



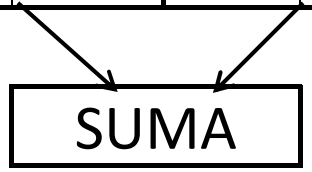
Ciclo 4: Se ejecuta RS 04 2/3
Se ejecuta RS 02 2/2
Se envía RS 02: 11.0 al CDB
NO se ejecuta RS 05, 03, 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2	Si	02	2.5
F4	Si	03	10.0
F6	Si	01	3.5

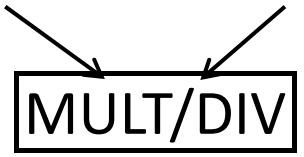
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	xx	03	xx
i3: 02	00	7.5	00	3.5
i5: 03	04	xx	05	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
i2: 04	00	4.0	00	7.5
i4: 05	02	xx	04	xx



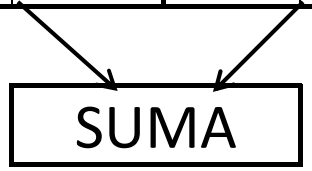
Ciclo 5: Se actualiza el valor de RS 02: 11.0
Se vacía RS 02
Se ejecuta RS 04 3/3
Se envía RS 04: 30.0 al CDB
NO se ejecuta RS 05, 03, 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4	Si	03	10.0
F6	Si	01	3.5

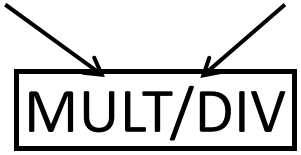
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	00	11.0	03	xx
02				
i5: 03	04	xx	05	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
i2: 04	00	4.0	00	7.5
i4: 05	00	11.0	04	xx



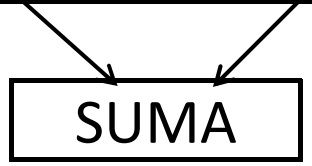
Ciclo 6: Se actualiza el valor de RS 04: 30.0
Se vacía RS 04
Se ejecuta RS 05 1/3
NO se ejecuta RS 03, 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4	Si	03	10.0
F6	Si	01	3.5

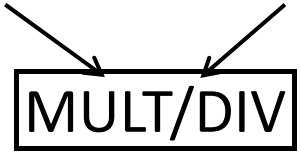
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	xx
02				
i5: 03	00	30.0	05	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i4: 05	02	11.0	00	30.0



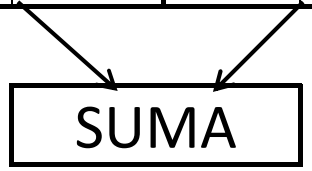
Ciclo 7: Se ejecuta RS 05 2/3
NO se ejecuta RS 03, 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4	Si	03	10.0
F6	Si	01	3.5

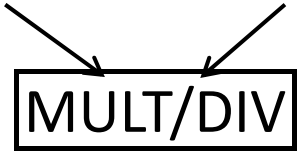
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	xx
02				
i5: 03	00	30.0	05	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i4: 05	02	11.0	00	30.0



Ciclo 8: Se ejecuta RS 05 3/3
Se envía RS 05: 330.0 al CDB
NO se ejecuta RS 03, 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4	Si	03	10.0
F6	Si	01	3.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	xx
02				
i5: 03	00	30.0	05	xx

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i4: 05	02	11.0	00	30.0

MULT/DIV

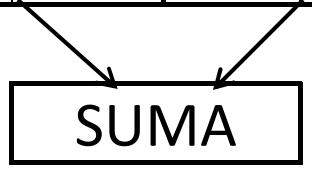
Ciclo 9: Se actualiza el valor de RS 05: 330.0
Se vacía RS 05
Se ejecuta RS 03 1/2
NO se ejecuta RS 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4	Si	03	10.0
F6	Si	01	3.5

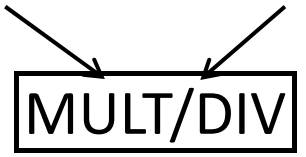
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	xx
02				
i5: 03	00	30.0	00	330.0



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				



Ciclo 10: Se ejecuta RS 03 2/2

Se envía RS 03: 360.0 al CDB

NO se ejecuta RS 01

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4	Si	03	10.0
F6	Si	01	3.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	xx
02				
i5: 03	00	30.0	00	330.0

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 11: Se actualiza el valor de RS 03: 360.0
Se vacía RS 03
Se ejecuta RS 01 1/2

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4			360.0
F6	Si	01	3.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	360.0
02				
03				

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 12: Se ejecuta RS 01 2/2

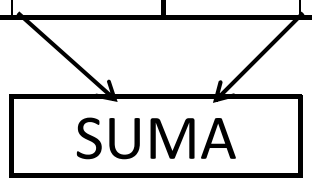
Se envía RS 01: 371.0 al CDB

FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4			360.0
F6	Si	01	3.5

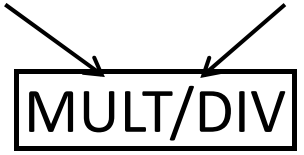
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i6: 01	02	11.0	03	360.0
02				
03				



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

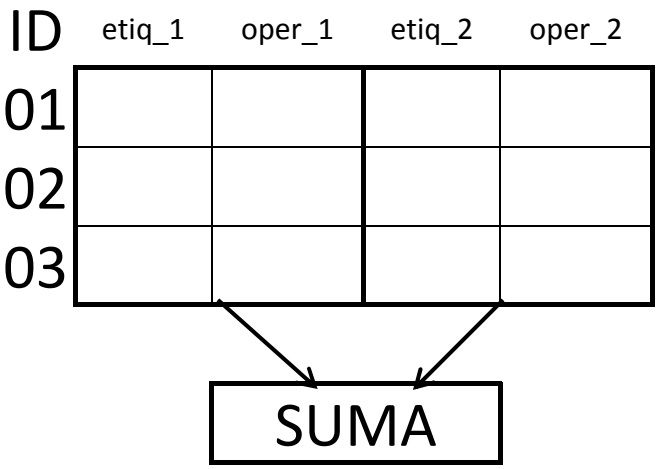


Ciclo 13: Se actualiza el valor de RS 0101: 371.0
Se vacía RS 01

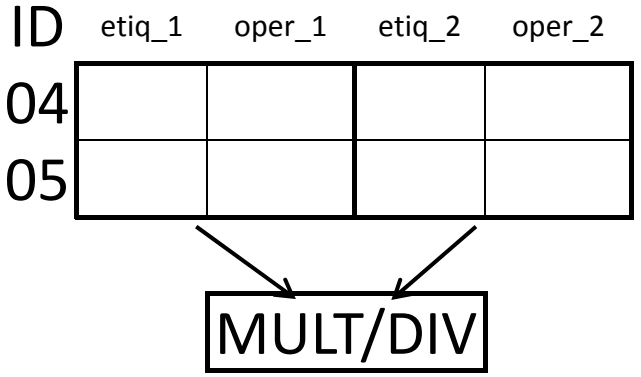
FR

	bitOc.	etiqueta	dato
F0			4.0
F2			11.0
F4			360.0
F6			371.0

RS



RS



Actividad 1.15

Mostrar la evolución de los Registros en coma flotante (FF) y las estaciones de Reserva (RS) para todos los ciclos que sean necesarios en la ejecución del siguiente fragmento de código utilizando el algoritmo de Tomasulo.

```
i1:  MULTD  F2,F2,F6
i2:  MULTD  F4,F2,F6
i3:  ADDD   F2,F2,F6
i4:  ADDD   F6,F2,F6
i5:  ADDD   F4,F4,F6
```

- Considere las siguientes hipótesis de partida:

- Para reducir el número de ciclos máquina se permite que la FLOS distribuya hasta dos instrucciones en cada ciclo según el orden del programa.

- Una instrucción puede comenzar su ejecución en el mismo ciclo en que se distribuye a una estación de reserva.

- La operación **suma** tiene una latencia de **dos ciclos** y la de **multiplicación** de **cuatro ciclos**.

- Se permite que una instrucción reenvíe su resultado a instrucciones dependientes durante su último ciclo de ejecución. De esta forma una instrucción a la espera de un resultado puede comenzar su ejecución en el siguiente ciclo si se detecta una coincidencia.

- Los valores de etiqueta 01, 02 y 03 se utilizan para identificar las tres estaciones de reserva de la unidad funcional de suma, mientras que 04 y 05 se utilizan para identificar las dos estaciones de reserva de la unidad funcional de multiplicación/división. Estos valores de etiqueta son los ID de las estaciones de reserva.

- Inicialmente, el valor de los registros es F0=2.0, F2=4.5, F4=8.0 y F6=3.0.

Ciclo 1: Se distribuye i1 e i2

i1: MULTD F2,F2,F6

i2: MULTD F4,F2,F6

Se ejecuta RS 04 1/4

No se ejecuta RS 05

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	04	4.5
F4	Si	05	8.0
F6			3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
01				
02				
03				

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i1: 04	00	4.5	00	3.0
i2: 05	04	xx	00	3.0

MULT/DIV

Ciclo 2: Se distribuye i3 e i4

i3: ADDD F2,F4,F6

i4: ADDD F6,F2,F6

Se ejecuta RS 04 2/4

NO se ejecuta RS 01,02,05

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	05	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
03				

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i1: 04	00	4.5	00	3.0
i2: 05	04	xx	00	3.0

Ciclo 3: Se distribuye i5

i5: ADDD F4,F4,F6

Se ejecuta RS 04 3/4

NO se ejecuta RS 01,02,03,05

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	05	xx	02	xx

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i1: 04	00	4.5	00	3.0
i2: 05	04	xx	00	3.0

Ciclo 4: Se ejecuta RS 04 4/4

Se envía RS 04: 13.5 al CDB

NO se ejecuta RS 01,02,03,05

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	05	xx	02	xx

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i1: 04	00	4.5	00	3.0
i2: 05	04	xx	00	3.0

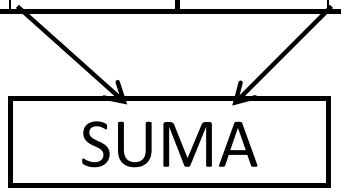
Ciclo 5: Se actualiza el valor de RS 04: 13.5
Se vacía RS 04
Se ejecuta RS 05 1/4
NO se ejecuta RS 01,02,03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

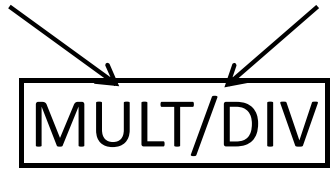
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	05	xx	02	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i2: 05	00	13.5	00	3.0



Ciclo 6: Se ejecuta RS RS 05 2/4

NO se ejecuta RS 01,02,03

FR

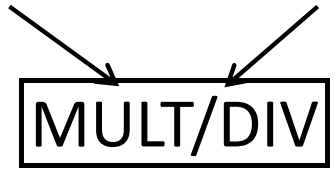
	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	05	xx	02	xx

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i2: 05	00	13.5	00	3.0



Ciclo 7: Se ejecuta RS RS 05 3/4

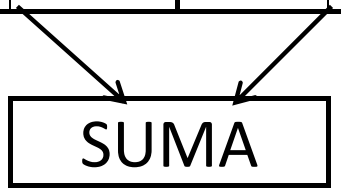
NO se ejecuta RS 01,02,03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

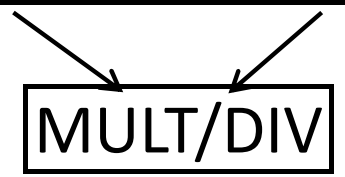
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	05	xx	02	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i2: 05	00	13.5	00	3.0



Ciclo 8: Se ejecuta RS RS 05 4/4

Se envía RS 05: 40.5 al CDB

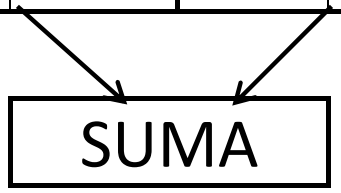
NO se ejecuta RS 01,02,03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

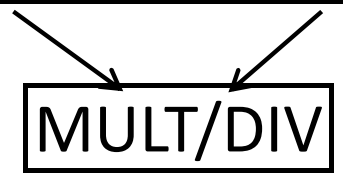
RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	05	xx	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	05	xx	02	xx



RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
i2: 05	00	13.5	00	3.0



Ciclo 9: Se actualiza el valor de RS 05: 40.5
Se vacía RS 05
Se ejecuta RS 01 1/2
NO se ejecuta RS 02,03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	00	40.5	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	00	40.5	02	xx

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 10: Se ejecuta RS 01 2/2

Se envía RS 01: 43.5 al CDB

NO se ejecuta RS 02,03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2	Si	01	4.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
i3: 01	00	40.5	00	3.0
i4: 02	01	xx	00	3.0
i5: 03	00	40.5	02	xx

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 11: Se actualiza el valor de RS 01: 43.5
Se vacía RS 01
Se ejecuta RS 02 1/2
NO se ejecuta RS 03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2			43.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
01				
i4: 02	00	43.5	00	3.0
i5: 03	00	40.5	02	xx

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 12: Se ejecuta RS 02 2/2

Se envía RS 02: 46.5 al CDB

NO se ejecuta RS 03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2			43.5
F4	Si	03	8.0
F6	Si	02	3.0

RS

ID	etiq_1	oper_1	etiq_2	oper_2
01				
i4: 02	01	43.5	00	3.0
i5: 03	00	40.5	02	xx

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 13: Se actualiza el valor de RS 02: 46.5
Se vacía RS 02
Se ejecuta RS 03 1/2

FR

	bitOc.	etiqueta	dato
F0			2.0
F2			43.5
F4	Si	03	8.0
F6			46.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
01				
02				
i5: 03	00	40.5	00	46.5

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 14: Se ejecuta RS 03 2/2

Se envía RS 03: 87 al CDB

FR

	bitOc.	etiqueta	dato
F0			2.0
F2			43.5
F4	Si	03	8.0
F6			46.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
01				
02				
i5: 03	00	40.5	00	46.5

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV

Ciclo 15: Se actualiza el valor de RS 03: 87

Se vacía RS 03

FR

	bitOc.	etiqueta	dato
F0			2.0
F2			43.5
F4			87.0
F6			46.5

RS

ID	etiq_1	oper_1	etiq_2	oper_2
01				
02				
03				

SUMA

RS

ID	etiq_1	oper_1	etiq_2	oper_2
04				
05				

MULT/DIV