3.21. Problema 21

Supongamos que tenemos que ejecutar el siguiente código en una arquitectura RISC muy similar al MIPS pero con memoria común de instrucciones y datos y ranura de salto de una instrucción.

```
Loop: LW R1,A(R4)
LW R2,B(R4)
ADD R1,R1,R2
SUB R1,R1,R3
SUB R4,R4,#1
SW C(R4),R1
BGTZ R4,Loop
ADD R3,R3,#1
ADD R1,R1,R5
...
```

Si el valor inicial del registro R4 es 1000, se pide:

1. CPI medio al ejecutar este programa.
2. Si la frecuencia de funcionamiento del reloj es de 20 MHz, ¿cuál es el tiempo de ejecución?

Apartado 1:

En este ejercicio tenemos que tener en cuenta que al haber memoria de datos e instrucciones común pueden aparecer riesgos estructurales cuando dos instrucciones quieren acceder a memoria. Para indicarlos, se ha subrayado las etapas que acceden a memoria. Si dos etapas quieren acceder simultáneamente a memoria se debe introducir una parada por riesgo estructural.

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOOP: LW R1,A(R4)	IF	ID	EX	ME	WB										
LW R2,B(R4)		IF	ID	EX	ME	WB									
ADD R1,R1,R2			IF	ID	ID	EX	ME	WB							
SUB R1,R1,R3				IF	IF	IF	ID	EX	ME	WB					
SUB R4,R4,#1							IF	ID	EX	ME	WB				
SW C(R4),R1								IF	ID	EX	ME	WB			
BGTZ R4, LOOP									IF	ID	EX	ME	WB		
PARADA										-	-	-	-	-	
DESTINO SALTO											IF	IF	ID	EX	ME
ADD R1,R1,R5													IF	ID	EX

Para calcular el CPI debemos ver el número de instrucciones que se ejecutan y cuantos ciclos tardan en ejecutarse. Se nos dice que el valor inicial de R4 es 1000, por lo tanto el bucle se ejecuta 1000 veces ya que en cada pasada se decrementa R4 en 1 y se salta si R4 es mayor que cero.

Hemos supuesto que existen adelantamientos y además la ranura de salto no es aprovechada, con lo cual cada salto perdemos un ciclo.

Para calcular el CPI procedemos a calcular el número de ciclos de la misma forma que en ejercicios anteriores:

1. Primera iteración: 4 ciclos de llenado + 7 instrucciones + 2 paradas por riesgos estructurales * 1 ciclo de parada + 1 ciclo de parada por salto = 14. No consideramos la parada por el riesgo RAW entre el LW y el ADD ya que debido al riesgo estructural cuando el ADD va a leer el dato, este ya está disponible.
2. 998 iteraciones: 7 instrucciones + 3 paradas por riesgos estructurales * 1 ciclo de parada + 1 parada por salto 11. El riesgo estructural adicional se debe a que cuando la instrucción destino del salto está en su etapa de IF , la operación SW de la iteración anterior está escribiendo en memoria.
3. Última iteración: 9 instrucciones + 3 paradas por riesgos estructurales * 1 ciclo de parada + 1 parada por salto = 13

$$CPI = \frac{ciclos}{instrucciones} = \frac{14 + 998 * 11 + 13}{999 * 7 + 9} = 1{,}57$$

Apartado 2:

Se nos pide el tiempo que tarda el programa en ejecutarse:

$$tiempo = \frac{ciclos}{frecuencia} = \frac{14 + 998 * 11 + 13}{20Mhz} = 0{,}55ns$$

3.22. Problema 22

Tenemos que ejecutar el siguiente código en una arquitectura MIPS que utiliza el adelantamiento de operandos y que calcula la dirección destino de los saltos y realiza la evaluación de las condiciones en la etapa EX.

```
L: LW R1,32(R2)
ADD R1,R1,R5
LW R10,64(R2)
SUB R4,R2,R10
BGTZ R4,L
ADD R1,R2,R6
ADD R1,R1,R7
ADD R1,R1,R1
SUB R1,R1,R3
```

Si el bucle se ejecuta 1000 veces ¿Qué CPI medio obtenemos al ejecutar el programa?

Solución:

Procedemos de igual forma que en ejercicios anteriores

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L: LW R1,32(R2)	IF	ID	EX	ME	WB										
ADD R1,R1,R5		IF	ID	ID	EX	ME	WB								
LW R10,64(R2)			IF	IF	ID	EX	ME	WB							
SUB R4,R2,R10					IF	ID	ID	EX	ME	WB					
BGTZ R4,L						IF	IF	ID	EX	ME	WB				
PARADA 3 CICLOS								-	-	-	-	-			
DESTINO SALTO											IF	ID	EX	ME	WB
ADD R1,R2,R7												IF	ID	EX	ME
ADD R1,R1,R1													IF	ID	EX
SUB R1,R1,R3														IF	ID

Para calcular el CPI procedemos a calcular el número de ciclos de la misma forma que en ejercicios anteriores:

1. Primera iteración: 4 ciclos de llenado + 5 instrucciones + 2 dependencias RAW * 1 ciclo de parada + 3 ciclos de parada por salto = 14
2. 998 iteraciones: 5 instrucciones + 2 dependencias RAW * 1 ciclo de parada + 3 ciclos de parada por salto = 10
3. Última iteración: 9 instrucciones + 2 dependencias RAW * 1 ciclo de parada + 3 ciclos de parada por salto = 14

$$CPI = \frac{ciclos}{instrucciones} = \frac{14 + 998 * 10 + 14}{999 * 5 + 9} = 2$$

3.23. Problema 23

Se ejecuta el siguiente código en el MIPS:

```
eti1 : LW R2,v(R1)
BEQ R2,R0,eti2
LW R3,W(R1)
SW v(R1),R3
SW w(R1),R2
eti2 : SUBI R1,R1,#4
BNE R1,R0,eti1
```

El bucle se aplica a vectores con un 80 % de componentes iguales a cero y nos piden:

1. Calcular el CPI medio de este código cuando se ejecuta un gran número de veces.
2. Reordenarlo para minimizar su tiempo de ejecución y calcular el CPI en este caso.

Solución:

Apartado 1:

Para calcular el CPI medio debemos distinguir dos casos. El primero cuando R2 es cero y por tanto el primer salto se toma y el segundo caso cuando R2 es distinto de cero. Con el resultado obtenido en cada una de las ramas, obtendremos el CPI total ponderando los resultados según el porcentaje de veces que R2 es cero.

Primeramente calcularemos el CPI cuando R2 es distinto de 0 y por lo tanto el salto no se toma. Asumiremos que el procesador dispone de anticipación de operandos y que los saltos se evalúan en la etapa de ID.

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
eti1: LW R2,v(R1)	IF	ID	EX	ME	WB										
BEQ R2,R0,eti2		IF	ID	ID	ID	EX	ME	WB							
LW R3,W(R1)			-	-	-	IF	ID	EX	ME	WB					
SW v(R1),R3							IF	ID	EX	ME	WB				
SW w(R1),R2								IF	ID	EX	ME	WB			
eti2: SUBI R1,R1,#4									IF	ID	EX	ME	WB		
BNE R1,R0, eti1										IF	ID	ID	EX	ME	WB
Destino salto											-	-	IF	ID	EX

En este caso el CPI será:

$$CPI_{R2 \neq 0} = \frac{ciclos}{instrucciones} = \frac{7ins + 3RAW + 2saltos}{7} = 1{,}71$$

También podemos calcular el número de ciclos que tarda en ejecutar el lazo mirando en que ciclo se vuelve a buscar la primera instrucción del lazo, en este caso en el ciclo 13, resultando por tanto que el bucle tarda 12 ciclos en ejecutarse.

En el otro caso, cuando R2 es igual a cero, la ejecución es la siguiente:

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14
eti1: LW R2,v(R1)	IF	ID	EX	ME	WB									
BEQ R2,R0,eti2		IF	ID	ID	ID	EX	ME	WB						
eti2: SUBI R1,R1,#4			-	-	-	IF	ID	EX	ME	WB				
BNE R1,R0, eti1							IF	ID	ID	EX	ME	WB		
Destino salto								-	-	IF	ID	EX	ME	WB

Calcularemos el CPI de forma análoga:

$$CPI_{R2=0} = \frac{ciclos}{instrucciones} = \frac{4ins + 3RAW + 2saltos}{4} = 2{,}25$$

Combinando ambos resultados obtenemos que el CPI medio es de:

$$CPI = 0{,}2 * 1{,}71 + 0{,}8 * 2{,}25 = 2{,}14$$

Apartado 2:

La reordenación que se propone es la siguiente:

```
LW R2,v(R1)
eti1:BEQ R2,R0,eti2
[SUBI R1,R1,#4]
LW R3,w+4(R1)
SW v+4(R1),R3
SW w+4(R1),R2
eti2:BNE R1,R0,eti1
[LW R2,v(R1)]
```

De esta forma aprovechamos los delay slot de los saltos para ejecutar código y además eliminamos la dependencia de datos entre la operación SUBI R1,R1,#4 y el salto BNE R1,R0,eti1 cuando el salto no se toma. Para eliminar esta dependiencia tenemos también que modificar los valores de los inmediatos con los que trabajan los loads y stores para que utilizen el valor correcto de R1.

Podemos calcular el nuevo CPI de la misma forma que antes:

$$CPI_{R2\neq 0} = \frac{ciclos}{instrucciones} = \frac{7ins + 2RAW + 0saltos}{7} = 1{,}28$$

$$CPI_{R2=0} = \frac{ciclos}{instrucciones} = \frac{4ins + 3RAW + 0saltos}{4} = 1{,}75$$

$$CPI = 0{,}2 * 1{,}28 + 0{,}8 * 1{,}75 = 1{,}65$$

3.24. Problema 24

Se ejecuta el siguiente fragmento de código en el MIPS:

```
loop: LD R1,0(R2)
DADDI R1,R1,#1
SD 0(R2),R1
DADDI R2,R2,#4
DSUB R4,R3,R2
BNE R0,R4,loop
```

Si el valor inicial de R3 es R2+396:

1. Mostrar el diagrama temporal de la ejecución si no se permite adelantamiento de operandos. ¿Cuántos ciclos tarda en ejecutarse el código?
2. Mostrar el diagrama temporal de la ejecución cuando se permite adelantamiento y se predice el salto como no tomado. ¿Cuántos ciclos se tarda ahora?
3. Reordenar las instrucciones para minimizar los ciclos de penalización por riesgos, suponiendo que usamos salto retardado. ¿Cuántos ciclos tarda en ejecutarse el código?

Solución:

Apartado 1:

Si no podemos realizar adelantamiento de operandos la ejecución del programa es la siguiente:

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
loop:LD R1,0(R2)	IF	ID	EX	ME	WB													
DADDI R1,R1,#1		IF	ID	ID	ID	EX	ME	WB										
SD 0(R2),R1			IF	IF	IF	ID	ID	ID	EX	ME	WB							
DADDI R2,R2,#4						IF	IF	IF	ID	EX	ME	WB						
DSUB R4,R3,R2									IF	ID	ID	ID	EX	ME	WB			
BNE R0,R4, loop										IF	IF	IF	ID	ID	ID	EX	ME	WB
Destino salto													-	-	-	IF	ID	EX

El valor inicial del registro R3=R2+396. Como cada pasada del bucle incrementa R2 en 4 y se le resta a R3 hasta que R4 vale cero, el bucle se ejecuta 396/4=99 veces.

En total el programa tarda:

Ciclos = 4 ciclos de llenado + 99*(6 instrucciones + 4 dependencias RAW * 2 ciclos + 1 salto * 1 ciclo) =1489 ciclos

Apartado 2:

Utilizando adelantamiento de operandos y prediciendo salto no tomado la ejecución del programa es la siguiente:

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13
loop:LD R1,0(R2)	IF	ID	EX	ME	WB								
DADDI R1,R1,#1		IF	ID	ID	EX	ME	WB						
SD 0(R2),R1			IF	IF	ID	EX	ME	WB					
DADDI R2,R2,#4					IF	ID	EX	ME	WB				
DSUB R4,R3,R2						IF	ID	EX	ME	WB			
BNE R0,R4, loop							IF	ID	ID	EX	ME	WB	
Destino salto								IF	IF	ID	EX	ME	WB

Para calcular el número de ciclos que se tarda en ejecutar el programa tenemos que tener en cuenta que 98 de las 99 veces que se ejecuta el bucle se falla la predicción del salto y en la ultima iteración se acierta. Por tanto:

Ciclos = 4 ciclos de llenado + 98*(6 instrucciones + 2 dependencias RAW * 1 ciclos + 1 fallo predicción * 1 ciclo) + (6 instrucciones + 2 dependencias RAW * 1 ciclo)=891 ciclos

Apartado 3:

Una posible reordenación del código sería:

```
loop: LD R1,0(R2)
DADDI R2,R2,#4
DADDI R1,R1,#1
DSUB R4,R3,R2
BNE R0,R4,loop
[SD -4(R2),R1]
```

Intercambiando las instrucciones DADDI eliminamos la dependencia del LD. Además para aprovechar el delay slot colocamos la instrucción SD dentro de él ya que es la única que podemos mover. Hay que tener en cuenta que hay que modificar el valor del offset del SD ya que al colocarlo detrás del DADDI R2,R2,#4, debemos restarle 4 para que siga accediendo a la misma posición.

3.25. Problema 25

Se ejecuta el siguiente fragmento de código en el MIPS:

```
loop: L.D F0,0(R2)
L.D F4,0(R3)
MUL.D F0,F0,F4
ADD.D F2,F0,F2
DADDUI R2,R2,#8
DADDUI R3,R3,#8
DSUBU R5,R4,R2
BNE R0,R5,loop
```

Si el valor inicial de R4 es R2+792, y suponemos que la multiplicación de flotantes tarda 7 ciclos en ejecutarse y la suma tarda 4.

1. Mostrar el diagrama temporal de la ejecución si no se permite adelantamiento de operandos. ¿Cuántos ciclos tarda en ejecutarse el código?
2. Mostrar el diagrama temporal de la ejecución cuando se permite adelantamiento y se predice el salto como no tomado. ¿Cuántos ciclos se tarda ahora?
3. ¿Y si se utiliza la técnica de salto retardado?

Solución:

Apartado 1 y 2:

El diagrama con la solución se puede consultar en la página siguiente.

Para calcular el número de ciclos que se tarda en ejecutar el programa en el caso 1 podemos hacer el siguiente cálculo, teniendo en cuenta que el bucle se ejecuta 792/8=99 veces.

Ciclos = 4 ciclo de llenado + 99*(8 instrucciones + 13 ciclos en los que no se termina ninguna instrucción + 1 parada por salto) = 2182 ciclos

En el apartado 2, haciendo un cálculo análogo, y teniendo en cuenta que fallamos la predicción de salto no tomado todas las veces menos la última,deducimos que el programa tarda:

Ciclos = 4 ciclos de llenado + 98*(8 instrucciones + 8 ciclos en los que no se termina ninguna instrucción + 1 parada por salto) + (8 instrucciones + 8 ciclos en los que no se termina instrucción) = 1686 ciclos.

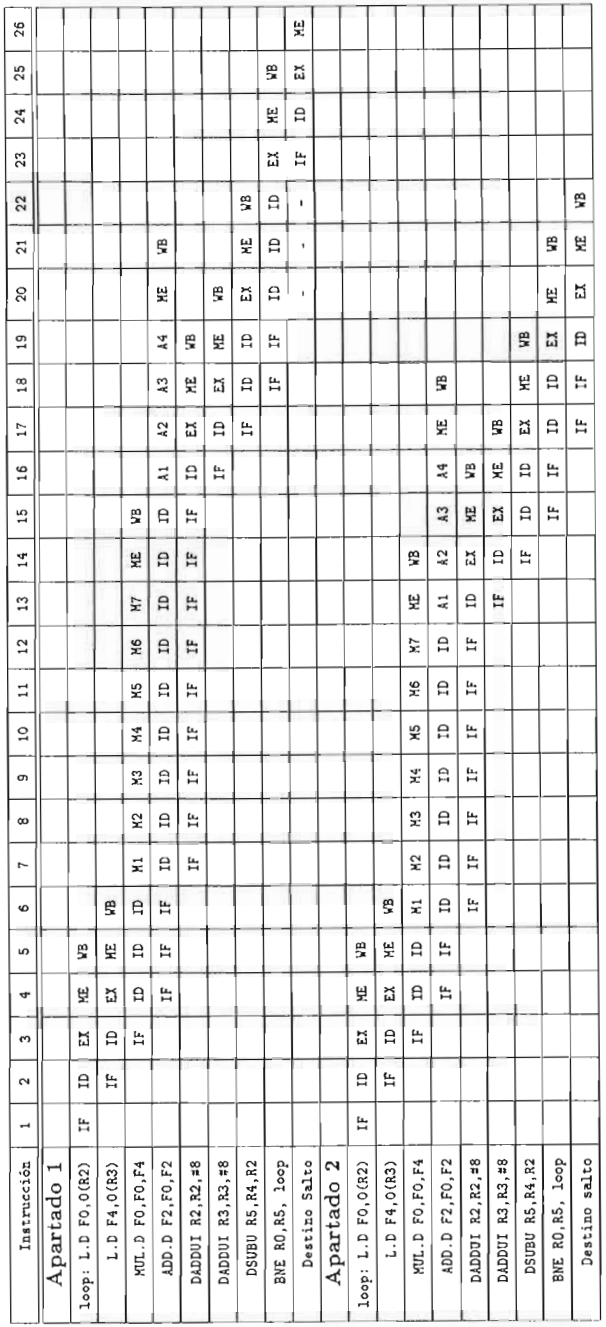
Apartado 3:

Una posible reordenación del código sería:

```
loop: L.D F0,0(R2)
L.D F4,0(R3)
MUL.D F0,F0,F4
ADD.D F2,F0,F2
DADDUI R2,R2,#8
DSUBU R5,R4,R2
BNE R0,R5,loop
[DADDUI R3,R3,#8]
```

De forma que aprovechamos el delay slot. En este caso no tendríamos penalizaciones por salto y el programa tardaría:

Ciclos= 4 + 99*(8+8) = 1588 ciclos



3.26. Problema 26

Se ejecuta en el MIPS el siguiente fragmento de código:

```
ADDI R10,R0,#0
ADDI R11,R0,#1
ADDI R12,R0,#800
IF: LW R1,A(R10)
BEQ R0,R1,ELSE
LW R2,B(R10)
ADD R3,R1,R2
SUB R4,R3,R11
SW R4,C(R10)
J FIN
ELSE: LW R2,B(R10)
SW R2,C(R10)
FIN: ADDI R10,R10,#8
BNE R10,R12,IF
```

Si el vector A tiene un 70 % de elementos iguales a 0:

1. Calcular el CPI medio de este código si se permite adelantamiento y la ranura de salto es de un ciclo.
2. Si se utiliza la técnica de salto retardado, mostrar la mejor reordenación posible de este código y calcular de nuevo el CPI medio.

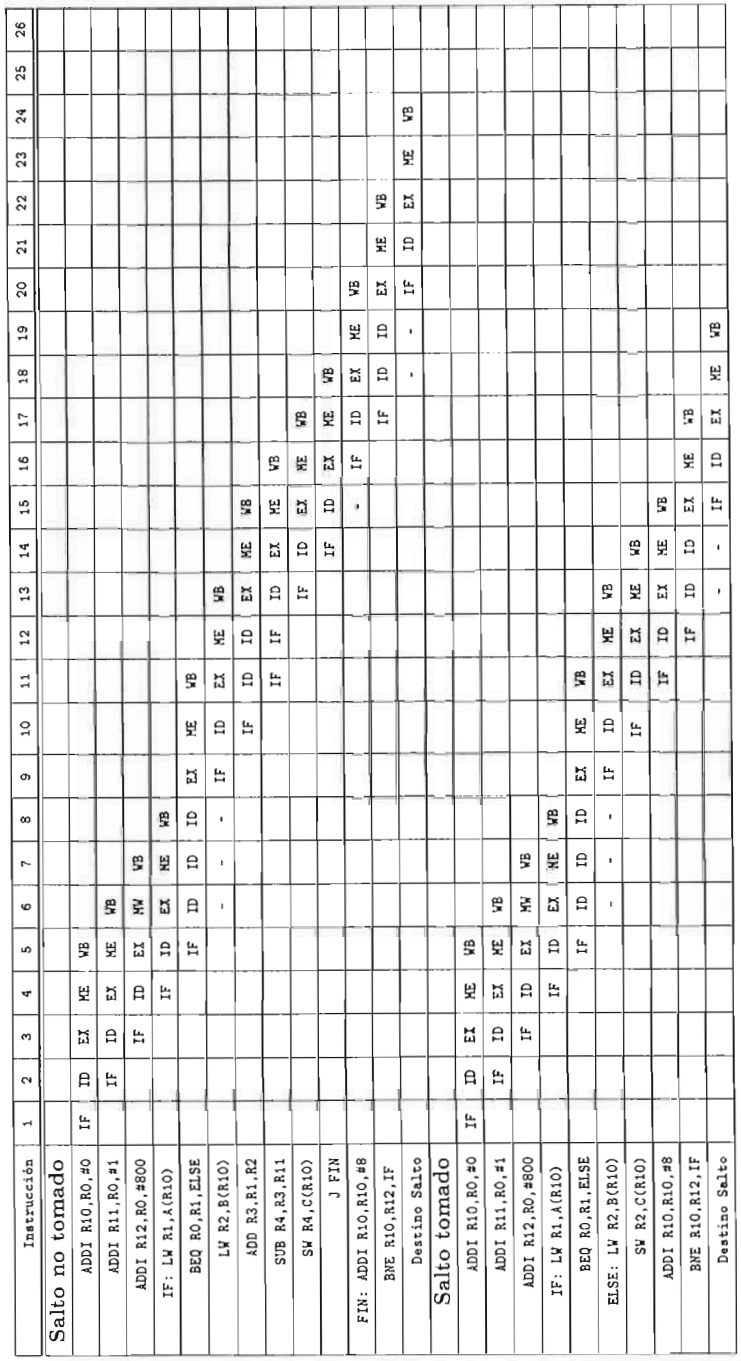
Solución:

Apartado 1:

Para resolver este ejercicio debemos calcular el CPI en dos casos, cuando el elemento que leemos de A vale cero y por tanto el salto se toma y cuando es distinto de cero y el salto no se toma.

Para calcular el CPI, calcularemos el CPI en ambos casos y tendremos en cuenta que el salto se toma el 70 % de las veces. En este caso para calcular el número de ciclos que se tarda en realizar una iteración del bucle restaremos 20, que es el ciclo en el que se empieza a buscar de nuevo la primera instrucción del bucle, menos 4 que es el ciclo en el que comienza a ejecutarse. A este valor le sumaremos 3 ciclos de las instrucciones que se ejecutan al principio.

$$CPI = \frac{ciclos}{instrucciones} = \frac{3ins + 100 * (0{,}3 * (20-4) + 0{,}7 * (15-4))}{3 + 100 * (0{,}3 * 9 + 0{,}7 * 6)} = 1{,}81$$



Apartado 2:

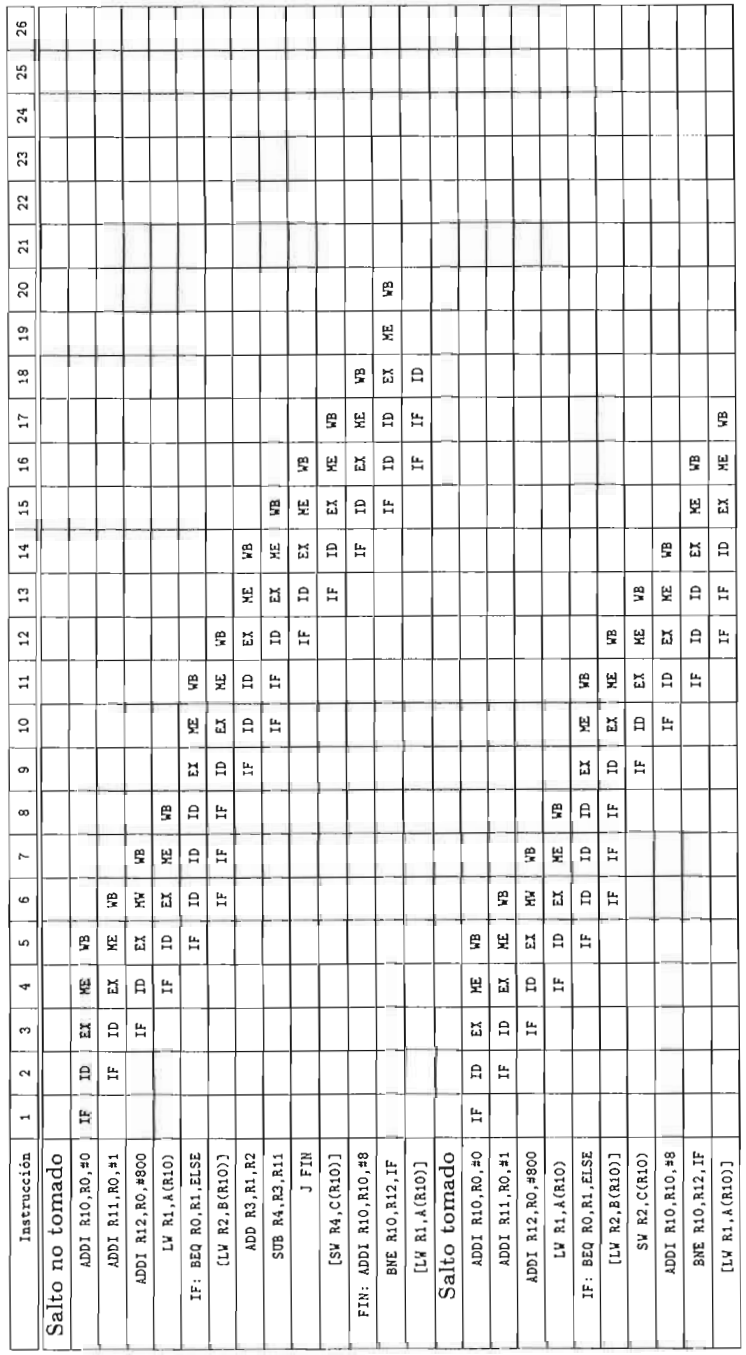
En este apartado se nos pide reordenar el código para conseguir el mejor resultado posible. Para ellos debemos aprovechar las ranuras de salto introduciendo en ellas instrucciones que se ejecuten siempre independientemente de si el salto se toma o no.

Para ello observamos que después del salto BEQ R0,R1,ELSE siempre se ejecuta la instrucción LW R2,B(R10) independientemente del resultado del salto. Por lo tanto colocaremos esta instrucción en la ranura de dicho salto. También vemos que siempre que se ejecuta el salto J FIN se ejecuta la instrucción anterior SW R4,C(R10) y que el salto no depende del resultado del SW. Por lo tanto podemos mover el SW a la ranura de salto del J FIN. Por último, podemos aprovechar la ranura del salto BNE R10, R12,IF introduciendo en ella el destino del salto y moviendo dicho destino una posición, en concreto directamente al otro salto BEQ R0,R1,ELSE.

Con estas modificaciones el código queda de la siguiente forma:

```
ADDI R10,R0,#0
ADDI R11,R0,#1
ADDI R12,R0,#800
LW R1,A(R10)
IF: BEQ R0,R1,ELSE
[LW R2,B(R10)]]
ADD R3,R1,R2
SUB R4,R3,R11
J FIN
[SW R4,C(R10)]
ELSE: LW R2,B(R10)
SW R2,C(R10)
FIN: ADDI R10,R10,#8
BNE R10,R12,IF
[LW R1,A(R0)]
```

De esta forma la ejecución del código es la siguiente:



En este caso calculamos el CPI de la misma forma:

$$CPI = \frac{ciclos}{instrucciones} = \frac{3ins + 100 * (0{,}3 * (18 - 5) + 0{,}7 * (14 - 5))}{3 + 100 * (0{,}3 * 9 + 0{,}7 * 6)} = 1{,}48$$

3.27. Problema 27

Ejecutamos el código siguiente en un procesador MIPS segmentado en cinco etapas.

```
LOOP: LD F2,0(R1)
DIVD F4,F4,F2
LD F2,8(R1)
DIVD F8,F2,F6
ADDD F12,F4,F6
ADDD F8,F2,F4
ADDI R1,R1,#32
SUB R5,R1,R3
SD 24(R1),F12
BEQZ R5,LOOP
```

Las unidades funcionales del procesador tienen las siguientes características:

Unidad	Cantidad	Latencia	Segmentado
Sumador	1	2	Si
Multiplicador	1	4	Si
Divisor	1	7	No

Además:

- Las dependencias se detectan y resuelven en la etapa ID
- Los saltos de resuelven en la etapa ID

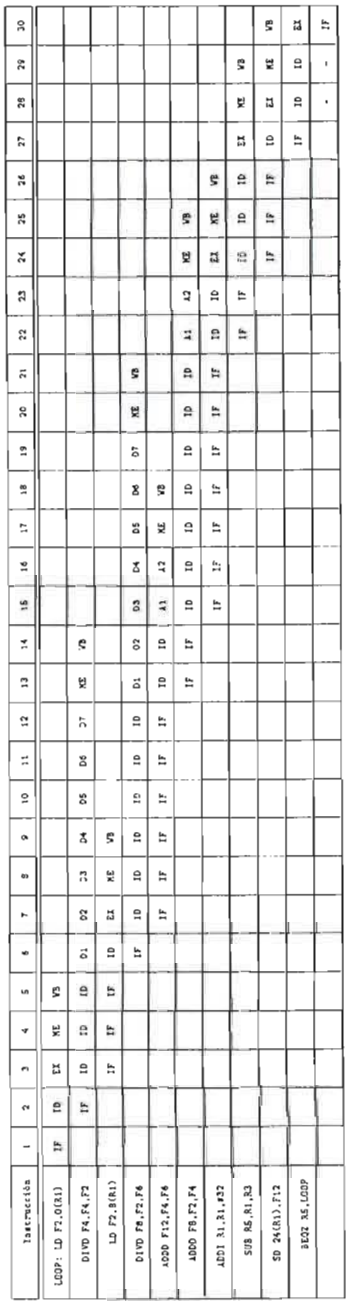
Suponiendo que el bucle se ejecuta 500 veces

1. Calcular el CPI medio de este código si NO se permite adelantamiento y no hay ranura de salto.
2. Calcular el CPI medio si se permite adelantamiento y suponiendo que los riesgos WAW se pueden solucionar en la etapa de WB inhibiendo la escritura.

Solución:

Apartado 1:

La ejecución del programa utilizando adelantamientos es la siguiente:



Hay que destacar varios aspectos de esta ejecución:

1. La instrucción DIV F8,F2,F6 tiene que esperar a que el divisor esté libre para poder ejecutarse.
2. La instrucción ADD F12,F4,F6 debe esperar a que DIV F4,F4,F2 termine ya que necesita el valor de F4.
3. La instrucción ADD F8,F2,F4 debe esperar a que DIV F8,F2,F6 termine ya que existe un riesgo WAW y si esta suma se ejecutase primero, el valor de F8 sería sobreescrito y por tanto incorrecto.
4. El resto de dependencias de datos son las habituales y no requieren más explicación.

El CPI en este caso es:

$$CPI = \frac{ciclos}{instrucciones} = \frac{30-1}{10} = 2{,}9$$

Apartado 2:

Hay que destacar varios aspectos en ejecución que se puede consultar en la página siguiente:

1. La instrucción DIV F8,F2,F6 tiene que esperar a que el divisor esté libre para poder ejecutarse. *Además cuando termina de ejecutar y va a escribir el resultado, como la instrucción ADDD F8,F2,F4 ya ha escrito ese registro, se inhibe la escritura para eliminar el riesgo WAW.
2. La instrucción ADD F8,F2,F4 puede ejecutarse ya que el sumador está segmentado.
3. La instrucción ADDI R1,R1,#32 tiene un ciclo de parada debido a un riesgo estructural ya que si no tuviera esa parada intentaría acceder a la etapa de memoria y escritura a la vez que la instrucción ADD F8,F2,F4.

El CPI en este caso es:

$$CPI = \frac{ciclos}{instrucciones} = \frac{19-1}{10} = 1{,}9$$

Instrucción	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
LOOP: LD F2,0(R1)	IF	ID	EX	ME	WB																
DIVD F4,F4,F2		IF	ID	ID	D1	D2	D3	D4	D5	D6	D7	ME	WB								
LD F2,8(R1)			IF	IF	ID	EX	ME	WB													
DIVD F8,F2,F6					IF	ID	ID	ID	ID	ID	ID	D1	D2	D3	D4	D5	D6	D7	ME	WB*	
ADDD F12,F4,F6						IF	IF	IF	IF	IF	IF	ID	A1	A2	ME	WB					
ADDD F8,F2,F4												IF	ID	A1	A2	ME	WB				
ADDI R1,R1,#32													IF	ID	ID	EX	ME	WB			
SUB R5,R1,R3														IF	IF	ID	EX	ME	WB		
SD 24(R1),F12																IF	ID	EX	ME	WB	
BEQZ R5,LOOP																	IF	ID	EX	ME	WB
																		-	IF		

Capítulo 4

Planificación Dinámica

4.1. Problema 1

Un procesador dispone de 1 unidad de enteros, 2 unidades de multiplicación/división y 3 unidades de suma/resta. Estas unidades emplean 2 ciclos para la suma y la resta, 9 para la multiplicación y 35 para la división. Además en este procesador se puede escribir en el banco de registros en la primera mitad del ciclo de reloj y se puede leer en la segunda mitad. Mostrar la evolución de las instrucciones , siguiendo los algoritmos de la Pizarra y de Tomasulo. Considerar el mismo número de estaciones de reserva que de unidades funcionales.

```
MUL F3,F2,F1
ADD F5,F3,F4
DIV F7,F6,F5
MUL F9,F1,F7
SUB F1,F2,F4
ADD F10,F1,F9
```

Solución:

Utilizando el algoritmo de la Pizarra la solución del ejercicio es la siguiente:

Instrucción	Emisión	Lectura	Ejecución	Escritura
MUL F3,F2,F1	1	2	3-11	12
ADD F5,F3,F4	2	13^{RAW}	13-14	15
DIV F7,F6,F5	3	15^{RAW}	16-50	51
MUL F9,F1,F7	13^{RE}	51^{RAW}	52-60	61
SUB F1,F2,F4	14	15	16-17	52^{WAR}
ADD F10,F1,F9	16^{RE}	61^{RAW}	62-63	64

Debemos destacar varios aspectos:

- Las instrucción MUL F9,F1,F7 debe esperar hasta el ciclo 13, ya que en el ciclo 4 cuando debería ser emitida los dos multiplicadores/divisores están ocupados y hasta

el ciclo 12 no se libera uno, ya que no están segmentados.

- La instrucción SUB F1,F2,F4 termina su ejecución en el ciclo 17, pero no puede escribir sus resultados en el ciclo 18 ya que sobreescribiría el valor de F1 antes de que la instrucción MUL F9,F1,F7 lo lea. Se trata por tanto de un riesgo WAR.

Utilizando el algoritmo de Tomasulo para planificar este código obtenemos:

Instrucción	Emisión	Ejecución	Escritura
MUL F3,F2,F1	1	2-10	11
ADD F5,F3,F4	2	$11 - 12^{RAW}$	13
DIV F7,F6,F5	3	$13 - 47^{RAW}$	48
MUL F9,F1,F7	12^{RE}	$48 - 56^{RAW}$	57
SUB F1,F2,F4	13	14-15	16
ADD F10,F1,F9	14	$57 - 58^{RAW}$	59

Aspectos a destacar en este ejercicio, son la ausencia de riesgos WAR o WAW debido a la propia estructura del algoritmo de Tomasulo que elimina estos riesgos mediante el renombrado de registros. Por lo demás los riesgos RAW y estructurales son similares a los encontrados al planificar el código mediante el algoritmo de la Pizarra,

4.2. Problema 2

Un procesador dispone de 2 unidades de multiplicación/división y de 3 unidades de suma/resta cuando utiliza el algoritmo de la Pizarra. Cuando utiliza Tomasulo tiene este número de estaciones de reserva pero una única unidad funcional de cada tipo. En estas unidades se emplean 2 ciclos para la suma y la resta, 10 para la multiplicación y 30 para la división. Además se puede escribir en los registros en la primera mitad del ciclo y leer en la segunda. Mostrar la evolución de las instrucciones, siguiendo los algoritmos de la Pizarra y de Tomasulo. Usar etiquetas numéricas para indicar los ciclos en los que comienzan y finalizan cada una de las etapas.

```
DIV F10,F11,F5
ADD F6,F10,F1
MUL F7,F6,F4
ADD F1,F13,F14
ADD F4,F1,F17
ADD F15,F16,F17
MUL F4,F3,F8
```


Solución:

Utilizando el algoritmo de la Pizarra la solución del ejercicio es la siguiente:

Instrucción	Emisión	Lectura	Ejecución	Escritura
DIV F10,F11,F5	1	2	3-32	33
ADD F6,F10,F1	2	33^{RAW}	34-35	36
MUL F7,F6,F4	3	36^{RAW}	37-46	47
ADD F1,F13,F14	4	5	6-7	34^{WAR}
ADD F4,F1,F17	5	34^{RAW}	35-36	37
ADD F15,F16,F17	35^{RE}	36	37-38	39
MUL F4,F3,F8	37^{WAW}	38	39-48	49

Los principales aspectos a destacar de este ejercicio son:

- La instrucción ADD F1,F13,F14 no puede escribir su resultado hasta el ciclo 34 ya que si no la instrucción ADD F6,F10,F1 leería un valor erróneo en el ciclo 33.
- La instrucción MUL F4,F3,F8 está bloqueada hasta el ciclo 37, ya que la instrucción ADD F4,F1,F17 tiene bloqueado el registro F4 para evitar la dependencia WAW y que otra instrucción escriba ese registro antes que ella. En el ciclo 37 termina esta suma y la instrucción de multiplicación puede comenzar.

Si planificamos el mismo código utilizando el algoritmo de Tomasulo el resultado es:

Instrucción	Emisión	Ejecución	Escritura
DIV F10,F11,F5	1	2-31	32
ADD F6,F10,F1	2	$32 - 33^{RAW}$	34
MUL F7,F6,F4	3	$34 - 43^{RAW}$	44
ADD F1,F13,F14	4	5-6	7
ADD F4,F1,F17	5	$7 - 8^{RAW}$	9
ADD F15,F16,F17	8	9-10	11
MUL F4,F3,F8	33^{RE}	44-53	54

En este caso tenemos que tener en cuenta que tenemos una unidad funcional de cada tipo pero varias estaciones de reserva para cada una, por lo tanto se pueden emitir tantas instrucciones como estaciones de reserva haya pero sólo una de esas instrucciones puede estar ejecutándose, ya que las unidades funcionales no están segmentadas. En el ejercicio debemos observar que la instrucción MUL F4,F3,F8 tiene que esperar a ser emitida ya que en el ciclo 9 y hasta el ciclo 33 las dos estaciones de reserva del multiplicador/divisor están ocupadas. Por lo demás el ejercicio no entraña ninguna dificultad.

4.3. Problema 3

Se dispone de un procesador similar al MIPS con una unidad de ejecución segmentada con las siguientes etapas:

- IF1: Primera fase de búsqueda de instrucción.

- IF2: Segunda fase de búsqueda de instrucción.
- ID: Decodificación de instrucción y lectura de registros.
- EX1: Evaluación de la condición y cálculo de la dirección destino de salto en las instrucciones de salto. Primera fase de ejecución.
- EX2: Segunda fase de ejecución.
- MEM: Acceso de memoria.
- WB: Escritura de registros.

Se desea comparar dos esquemas de predicción de saltos para su implementación en el procesador:

- PNT (Predict Not Taken): Siempre se predice que los saltos no se toman.
- BTB (Branch Target Buffer): Se utiliza un predictor de 1 bit, obteniendo la predicción y la dirección destino de salto al final de la fase IF2. Los saltos que no se toman no se almacenan en el BTB.

Para realizar la comparación de ambos esquemas de predicción se utiliza una carga típica en la que el 15 % de las instrucciones son saltos condicionales. De estas instrucciones el 80 % son saltos tomados. Además sabemos que el BTB acierta su predicción el 85 % de las veces. Las instrucciones que no son saltos tienen un CPI igual a 1.

Determinar qué esquema de predicción obtiene un CPI menor (analizar para cada predictor todos los casos posibles y los ciclos de penalización en cada caso).

Solución:

Utilizando el primer esquema PNT, debemos considerar dos casos, que el salto se tome, con lo cual estamos fallando la predicción o que el salto realmente no se tome, en cuyo caso acertamos.

En ambos casos debemos calcular cual es la penalización introducida.

PNT-Salto no tomado.

Salto	IF1	IF2	ID	EX1*	EX2	MEM	WB			
Salto+1		IF1	IF2	ID	EX1	EX2	MEM	WB		
Salto+2			IF1	IF2	ID	EX1	EX2	MEM	WB	
Salto+3				IF1	IF2	ID	EX1	EX2	MEM	WB

En la etapa EX1 de la instrucción de salto evaluamos la condición de salto con lo cual en ese punto sabemos que hemos acertado la predicción.

En este caso no existe penalización.

PNT-Salto tomado.

Salto	IF1	IF2	ID	EX1*	EX2	MEM	WB			
Salto+1		IF1	IF2	ID						
Salto+2			IF1	IF2						
Salto+3				IF1						
Destino					IF1	IF2	ID	EX1	EX2	MEM

En la etapa EX1 de la instrucción de salto evaluamos la condición de salto con lo cual en ese punto sabemos que hemos fallado la predicción, por lo tanto abortamos la ejecución de las instrucciones que ya han sido buscadas y saltamos al destino del salto.

En este caso la penalización es de 3 ciclos.

Por lo tanto el CPI con los datos que nos dan acerca del programa utilizando un predictor del tipo PNT es de:

$$CPI = 1 + (0{,}15 * 0{,}8) * 3 = 1{,}36$$

Donde 1 es el CPI de todas las instrucciones y 0.15*0.8 es el porcentaje de instrucciones de salto que son tomados.

En el caso de usar un BTB como el especificado en el enunciado hay que distinguir entre 4 casos diferentes. Se predice que se salta y se acierta, se predice que se salta y se falla, se predice que no se salta y se acierta y se predice que no se salta y se falla. Para cada uno de estos casos debemos calcular la penalización asociada.

BTB-Se predice que no se salta y se acierta.

Salto	IF1	IF2*	ID	EX1*	EX2	MEM	WB			
Salto+1		IF1	IF2	ID	EX1	EX2	MEM	WB		
Salto+2			IF1	IF2	ID	EX1	EX2	MEM	WB	
Salto+3				IF1	IF2	ID	EX1	EX2	MEM	WB

En la etapa IF2 se obtiene la predicción del salto, como se predice que no se salta se siguen ejecutando instrucciones. Cuando el salto llega a la etapa EX1 se resuelve la condición y al haber acertado se sigue la ejecución sin ninguna penalización.

BTB-Se predice que no se salta y se falla.

Salto	IF1	IF2*	ID	EX1*	EX2	MEM	WB			
Salto+1		IF1	IF2	ID						
Salto+2			IF1	IF2						
Salto+3				IF1						
Destino					IF1	IF2	ID	EX1	EX2	MEM

En este caso se predice que no se salta, por lo tanto se siguen ejecutando instrucciones y cuando la condición es resuelta en EX1 se salta al destino del salto. En este caso la penalización es de 3 ciclos.

BTB-Se predice que se salta y se falla.

Salto	IF1	IF2*	ID	EX1*	EX2	MEM	WB			
Salto+1		IF1								
Destino			IF1	IF2						
Destino+1				IF1						
Salto+1					IF1	IF2	ID	EX1	EX2	MEM

En este caso se predice que se salta en la etapa IF2 cuando ya se está buscando la instrucción Salto+1. Esa instrucción es descartada y en el ciclo siguiente se busca el destino del salto. Pero al llegar a la etapa EX1 se ve que se ha fallado la predicción con lo cual en el ciclo siguiente se vuelve a buscar la instrucción Salto+1 y se ejecuta.

La penalización en este caso es de 3 ciclos.

BTB-Se predice que se salta y se acierta.

Salto	IF1	IF2*	ID	EX1*	EX2	MEM	WB			
Salto+1		IF1								
Destino			IF1	IF2	ID	EX1	EX2	MEM	WB	
Destino+1				IF1	IF2	ID	EX1	EX2	MEM	WB

En este caso en IF2 se predice que se salta, mientras se está buscando la instrucción siguiente. En el ciclo siguiente se aborta esta instrucción y se busca la instrucción destino del salto. Cuando se evalúa la condición se ve que se ha acertado la predicción y la ejecución sigue normalmente. La penalización en este caso es de 1 ciclo.

Para calcular el CPI en el caso de utilizar este BTB procederemos de forma similar al caso del otro predictor.

$$CPI = 1 + \%saltos(\%fallos * 3 + \%tomadosacertados * 1) =$$

$$= 1 + 0{,}15 * (0{,}15 * 3 + 0{,}8 * 0{,}85 * 1) = 1{,}17$$

Por lo tanto el BTB funciona mejor que el predictor PNT.

4.4. Problema 4

Un procesador posee un predictor de 2 bits. Se ejecuta la siguiente secuencia de código en alto nivel:

```
filcol=0
diagonal=0
for (j=3,j-,j>=1)
  for (i=3,i-,i>=1)
    if i=j
      diagonal=diagonal+A(i,j)
    else filcol=filcol+A(i,j)
```

Que traducido a ensamblador queda:

```
ADDI R1,R0,#3
ADD R6,R0,R0
ADD R7,R0,R0
loop_j: ADDI R2,R0,#3
loop_i: SLL R3,R1,#2
ADD R3,R3,R2
LW R5,A(r3)
SUB R4,R2,R1
if: BNEZ R4,else
then: ADD R6,R6,R5
incond: BEQZ R0,endif
else: ADD R7,R7,r5
endif: SUBI R2,R2,#1
end_loop_i: BNEZ R2,loop_i
SUBI R1,R1,#1
end_loop_j: BNEZ R1,loop_j
```

Si denominamos a los estados del predictor A (11), B(10), C (01) y D(00), el estado inicial del buffer de predicción de saltos es:

if	incond	end_loop_i	end_loop_j
D	D	A	A

1. Explicar los estados por los que va pasando el predictor al ejecutar este fragmento de código. Para cada iteración, mostrar el estado inicial y el final del buffer de predicción de saltos.
2. Calcular el porcentaje de aciertos que se produce en cada uno de los saltos.
3. Repetir el ejercicio suponiendo que el predictor que se utiliza es multinivel (1,1) y que el último salto antes de ejecutar este código no se tomo.

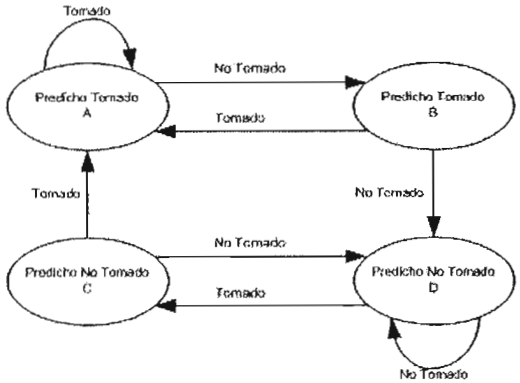
Solución:

Apartado 1

Mirando el código ensamblador, podemos ver donde están colocados en el código de alto nivel.

```
filcol=0
diagonal=0
for (j=3,j-,j>=1)
  for (i=3,i-,i>=1)
    if: if (i=j)
      diagonal=diagonal+A(i,j)
      salto incondicional
      else filcol=filcol+A(i,j)
  end_loop_i:
end_loop_j:
```

Con este código y sabiendo el funcionamiento del predictor mostrado en la figura podemos extraer los estados por los que va pasando y calcular el número de fallos que comete.



Para ello haremos una tabla donde para cada iteración de los bucles y sus valores veremos si los saltos se toman o no y anotaremos el estado del predictor así como si su predicción ha sido correcta:

		if			incond			end_loop_i			end_loop_j		
j	i	ini	fin	a/f	ini	fin	a/f	ini	fin	a/f	ini	fin	a/f
3	3	D	D	A	D	C	F	A	A	A	A	-	-
3	2	D	C	F	C	-	-	A	A	A	A	-	-
3	1	C	A	F	C	-	-	A	B	F	A	A	A
2	3	A	A	A	C	-	-	B	A	A	A	-	-
2	2	A	B	F	C	A	F	A	A	A	A	-	-
2	1	B	A	A	A	-	-	A	B	F	A	A	A
1	3	A	A	A	A	-	-	B	A	A	A	-	-
1	2	A	A	A	A	-	-	A	A	A	A	-	-
1	1	A	B	F	A	A	A	A	B	F	A	B	F

Apartado 2

Para calcular el porcentaje de acierto del predictor calcularemos cuantas veces acierta sus predicciones. Contando la columna de Acierto/Fallo tenemos que para cada salto:

	Tasa Fallo
if	4/9
incondicional	2/3
end_loop_i	3/9
end_loop_j	1/3

Apartado 3

En este apartado cambiamos de predictor por uno multinivel (1,1) donde el primer bit indica la predicción de si el último salto no se tomó y el segundo bit la predicción de si el último salto sí se tomó.

Si el bit de predicción es 0 predecimos que el salto no se toma y si es 1 predecimos que sí se toma. Si acertamos la predicción la dejamos igual, si fallamos la modificamos.

De esta forma y teniendo en cuenta que nos dicen que el último salto antes de los bucles no se tomó, la tabla queda de la siguiente forma:

		if			incond			end_loop_i			end_loop_j		
j	i	ini	fin	a/f	ini	fin	a/f	ini	fin	a/f	ini	fin	a/f
3	3	00	00	Ant	00	10	Ft	11	11	At	11	-	-
3	2	00	01	Ft	10	-	-	11	11	At	11	-	-
3	1	01	01	At	10	-	-	11	10	Fnt	11	11	At
2	3	01	01	At	10	-	-	10	11	Ft	11	-	-
2	2	01	00	Fnt	10	10	At	11	11	At	11	-	-
2	1	00	01	Ft	10	-	-	11	10	Fnt	11	11	At
1	3	01	01	At	10	-	-	10	11	Ft	11	-	-
1	2	01	01	At	10	-	-	11	11	At	11	-	-
1	1	01	00	Fnt	10	10	At	11	10	Fnt	11	01	Fnt

Subrayado se encuentra el bit de predicción que estamos utilizando en cada momento,

que como ya hemos dicho depende de si el salto anterior se tomo o no.

Con este esquema de predicción nuestra nueva tasa de fallos es:

	Tasa Fallo
if	4/9
incondicional	1/3
end_loop_i	5/9
end_loop_j	1/3

4.5. Problema 5

Un procesador realiza especulación con las siguientes fases:

- IF: Búsqueda de instrucción
- IS: Decodificación de instrucción y lectura de operandos
- EX: Ejecución (latencias de 1 ciclo para operaciones enteras, 2 para suma/resta, 4 para multiplicación y 10 para división)
- WB: Escritura de resultados en el CDB
- C: Confirmación de las instrucciones, escritura de resultados

Este procesador dispone de un BTB que obtiene sus predicciones al final de la fase IF y una unidad funcional de cada tipo, con todas las unidades de coma flotante segmentadas. Se supone un número ilimitado de entradas en el buffer de reordenamiento y en cuanto a las estaciones de reserva: 2 para operaciones enteras, 3 para suma y resta en punto flotante y 2 para multiplicación y división.

Además suponer que los datos que se escriben en el buffer de reordenamiento pueden ser leídos en el mismo ciclo.

Se ejecuta el siguiente fragmento de código:

```
MUL.D F2,F4,F8
ADD.D F4,F4,F10
SUB.D F6,F2,F4
BLTZ F6,salto
ADD.D F2,F4,F10
MUL.D F6,F8,F4
MUL.D F6,F6,F2
MUL.D F6,F6,F6
SUB.D F6,F6,F8
ADD.D F6,F6,F6
salto: SUB.D F2,F6,F4
DIV.D F4,F2,F2
S.D F4,8(R0)
```


Valores iniciales:

F0	F2	F4	F6	F8	F10
1	7	4	6	3	10

Calcular:

1. Suponiendo que el BTB predice que la instrucción de salto no salta, mostrar la evolución temporal de las instrucciones
2. Mostrar el estado de las estaciones de reserva y del buffer de reordenamiento cuando la instrucción SUB.D F6,F2,F4 ejecuta la fase C.

Solución:

Apartado 1

Al utilizar el algoritmo de Tomasulo con especulación, los datos son escritos en el buffer de reordenamiento para forzar la terminación en orden, por lo tanto aparece una nueva etapa denominada Commit en la cual los datos son escritos definitivamente en el banco de registros. En azul se muestran las instrucciones que están siendo especuladas.

Instrucción	Emisión	Lectura	Ejecución	Escritura	Commit
MUL.D F2,F4,F8	1	2	3-6	7	8
ADD.D F4,F4,F10	2	3	4-5	6	9
SUB.D F6,F2,F4	3	7^{RAW}	8-9	10	11
BLTZ F6,salto	4	10^{RAW}	11	12	13**
ADD.D F2,F4,F10	5	6	7-8	9	
MUL.D F6,F8,F4	6	8^{HK}	9-12	13	
MUL.D F6,F6,F2	7	13^{RAW}			
MUL.D F6,F6,F6	8	RAW			
SUB.D F6,F6,F8	9				
ADD.D F6,F6,F6	10				
salto: SUB.D F2,F6,F4	11				
DIV.D F4,F2,F2	12				
S.D F4,8(R0)	13				
salto: SUB.D F2,F6,F4	14	15	16-17	18	19
DIV.D F4,F2,F2	15	18^{RAW}	19-28	29	30
S.D F4,8(R0)	16	29^{RAW}	30	31	32

**La instrucción de salto BLTZ obtiene su predicción en la fase de búsqueda con lo cual desde el ciclo 4 al 13 que es cuando hace commit de su resultado podemos ejecutar instrucciones basándonos en la predicción del BTB. Mientras especulamos no volcamos los resultados en el banco de registros. En el caso de que la predicción fuera acertada haríamos el commit de todos esos datos. En el caso de haber fallado la predicción borraríamos todas las entradas del buffer de reordenamiento y continuaríamos buscando la instrucción destino del salto.

En nuestro caso calcularemos si el salto se toma o no, mirando el valor de F6.

F6=(F4*F8)-(F4+F10)=(4*3)-(4+10)=-2, por lo tanto el salto se toma y la predicción ha fallado, por lo tanto en el ciclo 13 todas las instrucciones que están siendo especuladas son eliminadas y se continúa ejecutando el salto.

Apartado 2

La instrucción SUB.D F6,F2,F4 hace su commit en el ciclo 11. En ese momento las instrucciones MUL.D F2,F4,F8, ADD.D F4,F4,F10 y ADD.D F2,F4,F10 ya han terminado.

Las estaciones de reserva tendrán las siguientes instrucciones en su interior:

UF	Instrucción	Vi	Vj	Qi	Qj	Destino
Add1	SUB.D	-	|F8|	Mult2	-	F6
Add2	ADD.D	-	-	Add1	Add1	F6
Add3	SUB.D	-	|F4|	Add2	-	F2
Mult1	MUL.D	-	[CDB]9	UFMult	-	F6
Mult2	MUL.D	-	-	Mult1	Mult1	F6

Para ello tenemos en cuenta que al estar las unidades funcionales segmentadas, las estaciones de reserva se vacían cuando la instrucción pasa a ejecución.

En la tabla |F8| significa que ese valor se ha leído del banco de registros. [CDB]9 significa que ese valor se ha leído del Common Data Bus en el ciclo 9 y UFMult significa que el valor está siendo generado por el multiplicador segmentado.

El buffer de reordenamiento contendrá las siguientes entradas, que corresponden con todas las instrucciones que han sido emitidas pero que todavía no han podido hacer commit:

	Instrucción	Destino	Resultado	Fase
1	BLTZ F6,salto			Ejecución
2	ADD.D F2,F4,F10	F2	[CDB]9	
3	MUL.D F6,F8,F4	F6		Ejecución
4	MUL.D F6,F6,F2	F6		Búsqueda
5	MUL.D F6,F6,F6	F6		Búsqueda
6	SUB.D F6,F6,F8	F6		Búsqueda
7	ADD.D F6,F6,F6	F6		Búsqueda
8	salto: SUB.D F2,F6,F4	F2		Búsqueda

4.6. Problema 6

Se utiliza Tomasulo para planificar en el MIPS el siguiente código:

```
LW R8,200(R0)
loop: DIV F10,F11,F5
ADD F6,F10,F1
MUL F7,F6,F4
ADD F15,F16,F17
SUBI R8,R8,#10
BNE R0,R8,loop
MUL F3,F2,F1
ADD F5,F3,F4
DIV F7,F6,F5
MUL F9,F1,F7
ADD F1,F13,F14
ADD F4,F1,F17
```

Este procesador dispone de un BTB que obtiene sus predicciones al final de la fase IF y todas las unidades funcionales de coma flotante están segmentadas. Las latencias de las operaciones son de 1 ciclo para las enteras, 2 para la suma/resta en FP, 4 para la multiplicación y 20 para la división. Además hay una unidad funcional de cada tipo y 2 estaciones de reserva para enteros, 3 para suma/resta en FP y 2 para multiplicación/división. Se suponen 10 entradas en el buffer de reordenamiento. El valor inicial de la posición 200 de la memoria de datos es un 20.

1. Suponiendo que el predictor predice que la instrucción de salto salta, mostrar la evolución temporal de las instrucciones hasta que surge el primer riesgo estructural por el ROB.

2. Mostrar el estado de las estaciones de reserva, del buffer de reordenamiento y del banco de registros cuando la instrucción BNE R0,R8,loop está en fase de ejecución.

Suponer que no se puede leer un dato en el mismo ciclo en el que ha sido escrito en el ROB.

Solución:

Instrucción	Emisión	Ejecución	Escritura	Commit
LW R8,200(R0)	1	2	3	4
loop: DIV F10,F11,F5	2	3-22	23	24
ADD F6,F10,F1	3	$24-25^{RAW}$	26	27
MUL F7,F6,F4	4	$27-30^{RAW}$	31	32
ADD F15,F16,F17	5	6-7	8	33
SUBI R8,R8,#10	6	7	9^{RE}	34
BNE R0,R8,loop	7	10^{RAW}	11	35
loop: DIV F10,F11,F5	8	9-28	29	36
ADD F6,F10,F1	9	$30-31^{RAW}$	32	37
MUL F7,F6,F4	10	$33-36^{RAW}$	37	38
ADD F15,F16,F17	11	12-13	14	39
SUBI R8,R8,#10	25^{RE}			
BNE R0,R8,loop				

En el ciclo 12 cuando se va a buscar la instrucción SUBI R8,R8,#10, el buffer de reordenamiento tiene ya 10 instrucciones, ya que las anteriores al estar especulando no han podido realizar su fase de commit y no han salido del ROB.

En este ejercicio también podemos ver como las unidades funcionales segmentadas ejecutan varias instrucciones simultáneamente.

En ese instante de tiempo el ROB tiene las siguientes entradas:

	Instrucción	Destino	Resultado	Fase
1	DIV.D F10,F11,F5	F10		Ejecución
2	ADD F6,F10,F1	F6		Ejecución
3	MUL F7,F6,F4	F7		Ejecución
4	ADD F15,F16,F17	F15	[CDB]8	Escritura
5	SUBI R8,R8,#10	R8	[CDB]9	Escritura
6	BNE R0,R8,loop			Escritura
7	DIV.D F10,F11,F5	F10		Ejecución
8	ADD.D F6,F10,F1	F6		Emisión
9	MUL.D F7,F6,F4	F7		Emisión
10	ADD.D F15,F16,F17	F17		Escritura

4.7. Problema 7

Se utiliza Tomasulo especulativo para planificar en el MIPS el siguiente código:

```
bucle: MUL.D F3,F1,F2
ADD.D F6,F3,F10
DIV.D F7,F7,F4
SUB.D F7,F16,F17
S.D F7,X(R0)
```



```
SUBI R8,R8,#1
BNE R0,R8,bucle
ADD.D F3,F10,F1
DIV.D F5,F1,F8
S.D F5,Z(R0)
MUL.D F9,F1,F2
ADD.D F1,F13,F14
ADD.D F4,F1,F17
```

Este procesador dispone de un BTB que obtiene sus predicciones al final de la fase IF y todas las unidades funcionales de coma flotante están segmentadas. Las latencias de las operaciones son de 1 ciclo para las enteras, 2 para la suma/resta en FP, 4 para la multiplicación y 15 para la división. Además hay una unidad funcional de cada tipo y 2 estaciones de reserva para enteros, 3 para suma/resta en FP y 2 para multiplicación/división. Se suponen 10 entradas en el buffer de reordenamiento y que los datos que se escriben en el CDB no pueden ser leídos hasta el ciclo siguiente.

1. Suponiendo que el predictor predice que la instrucción de salto salta en todas las ocasiones, mostrar la evolución temporal de las instrucciones hasta que surge el primer riesgo estructural por el ROB.
2. Mostrar el estado del ROB en ese momento (ciclo en el que se produce el primer riesgo estructural).

Solución:

Este problema es muy similar al anterior. La ejecución del programa utilizando el algoritmo de Tomasulo con especulación se muestra a continuación. La zona en azul corresponde al igual que en ejercicios anteriores a las instrucciones que están siendo especuladas.

Instrucción	Emisión	Ejecución	Escritura	Commit
bucle: MUL.D F3,F1,F2	1	2-5	6	7
ADD.D F6,F3,F10	2	$7-8^{RAW}$	9	10
DIV.D F7,F7,F4	3	4-13	19	20
SUB.D F7,F16,F17	4	5-6	7	21
S.D F7,X(R0)	5	8^{RAW}	10^{RE}	22
SUBI R8,R8,#1	6	7	8	23
BNE R0,R8,bucle	7	9^{RAW}	11^{RE}	24
bucle: MUL.D F3,F1,F2	8	9-12	13	
ADD.D F6,F3,F10	9	$14-15^{RAW}$	16	
DIV.D F7,F7,F4	10	11-		
SUB.D F7,F16,F17	11	12-13	14	
S.D F7,X(R0)	12	15^{RAW}	17^{RE}	
SUBI R8,R8,#1	21^{RE}			

En este caso también las estaciones de reserva quedan libres cuando la instrucción pasa a ejecución para poder aprovechar la segmentación de las unidades funcionales.

En el ciclo 21, hay 10 instrucciones en el ROB por lo tanto ocurre un riesgo estructural. En ese instante de tiempo el contenido del ROB es el siguiente:

	Instrucción	Destino	Resultado	Fase
1	DIV.D	F7		Ejecución
2	SUB.D	F7	[CDB]7	Escritura
3	S.D	Mem[X+R0]	[CDB]9	Escritura
4	SUBI	R8	[CDB]8	Escritura
5	BNE			Escritura
6	MUL.D	F3	[CDB]13	Escritura
7	ADD.D	F6		Emisión
8	DIV.D	F7		Ejecución
9	SUB.D	F7		Ejecución
10	S.D	Mem[X+R0]		Emisión

4.8. Problema 8

Se dispone del siguiente código:

```
DIVD F10, F11,F5
ADDD F6,F10,F1
MULD F10,F6,F1
ADDD F6,F1,F2
SUB F2,F6,F3
DIVD F6,F3,F2
MULD F6,F6,F6
```

1. Mostrar como sería la ejecución del código utilizando el algoritmo de la Pizarra en un procesador con dos unidades de multiplicación/división con latencia de 10/30 ciclos y tres unidades de suma/resta con latencia de 2 ciclos. Suponer que se puede escribir en los registros en la primera mitad del ciclo y leer en la segunda.

2. Mostrar como sería la ejecución del código utilizando el algoritmo de Tomasulo, en un procesador con una unidad de multiplicación/división segmentada con 2 estaciones de reserva con latencia 10/30 y una unidad de suma/resta segmentada con 3 estaciones de reserva y latencia de 2 ciclos. Suponer que se puede escribir en los registros en la primera mitad del ciclo y leer en la segunda.

Solución:

Utilizando el algoritmo de la Pizarra la solución del ejercicio es la siguiente:

Instrucción	Emisión	Lectura	Ejecución	Escritura
DIVD F10,F11,F5	1	2	3-32	33
ADDD F6,F10,F1	2	33^{RAW}	34-35	36
MULD F10,F6,F1	34^{WAW}	36^{RAW}	37-46	47
ADDD F6,F1,F2	37^{WAW}	38	39-40	41
SUB F2,F6,F3	38	41^{RAW}	42-43	44
DIVD F6,F3,F2	42^{WAW}	44^{RAW}	45-74	75
MULD F6,F6,F6	76^{WAW}	77	78-89	90

La principal dificultad de este ejercicio es la gran cantidad de dependencias WAW que aparecen. Para solucionarlas utilizando el algoritmo de la Pizarra es necesario parar la instrucción que tiene la dependencia hasta que la anterior haya escrito su resultado.

Utilizando el algoritmo de Tomasulo las dependencias WAR y WAW se solucionan mediante el renombrado de registros. No hay que olvidar que al estar las UF segmentadas las estaciones de reserva quedan libres en cuanto la instrucción pasa a ejecución.

Instrucción	Emisión	Ejecución	Escritura
DIVD F10,F11,F5	1	2-31	32
ADDD F6,F10,F1	2	$32 - 33^{RAW}$	34
MULD F10,F6,F1	3	$35 - 44^{RAW}$	45
ADDD F6,F1,F2	4	5-6	7
SUB F2,F6,F3	5	$7 - 8^{RAW}$	9
DIVD F6,F3,F2	6	$9 - 10^{RAW}$	11
MULD F6,F6,F6	9	$11 - 20^{RAW}$	21

Capítulo 5

Jerarquía de Memoria

5.1. Problema 1

Un sistema de memoria de dos niveles tiene un tiempo medio de acceso de 12 ns. El nivel superior de la jerarquía tiene una tasa de aciertos del 90 por 100 y un tiempo de acceso de 5 ns. ¿Cuál es el tiempo de acceso del nivel inferior del sistema de memoria?

Solución:

Si el tiempo medio es de 12 ns tenemos $0{,}9 * 5ns + 0{,}1 * Xns = 12$ lo que implica $X = \frac{12-0{,}9*5}{0{,}1}$ siendo el tiempo del nivel inferior 75 ns

5.2. Problema 2

Para una caché de capacidad igual a 32 KB ¿Cuántas líneas de caché existen para las longitudes de línea de 32, 64 y 128 bytes?

Solución:

El tamaño de la caché es de 32*1024 Bytes, para una longitud de linea de 32 Bytes tendremos $\frac{32}{32} * 1024$ lineas, resultando 1 KL.

En el caso de 64 Bytes $\frac{32}{64} * 1024$ que es un total de 512 L.

Finalmente para 128 bytes $\frac{32}{128} * 1024$ lo que hacen 256 L.

5.3. Problema 3

Si una caché tiene una capacidad de 16KB y la longitud de línea es de 128 bytes, ¿Cuántos conjuntos tiene la caché si es asociativa por conjuntos con 2 vías, o de 4 vías, o de 8 vías?

Solución:

La caché consta de $\frac{16*1024}{128} = 128$ líneas, una caché asociativa por conjuntos de n vías requiere una agrupación de las n líneas en lo que denominamos conjunto, entonces de 2 vías: tendremos $\frac{128}{2} = 64$ conjuntos, con 4 vías $\frac{128}{4} = 32$ conjuntos, finalmente con 8 vías $\frac{128}{8} = 16$ conjuntos.

5.4. Problema 4

Una caché tiene 64 KB de capacidad, 128 bytes por línea y es asociativa por conjuntos con 4 vías. El sistema que tiene la caché utiliza direcciones de 32 bits.

1. ¿Cuántas líneas y conjuntos tiene la caché?
2. ¿Cuántas etiquetas se necesitan?
3. ¿Cuántos bits de etiqueta se necesitan en cada entrada?
4. Si la caché es de escritura directa, ¿Cuántos bits de estado se necesitan, y cuanta memoria total se necesita para almacenar las etiquetas, si se utiliza una política de reemplazo LRU? ¿Qué pasaría si la caché es de post-escritura?

Solución

1. La caché tiene $\frac{64*1024}{128} = 512$ líneas, y $\frac{512}{4} = 128$ conjuntos
2. Se necesita una etiqueta por línea debido a la estructura hardware de la caché.
3. Si la dirección consta de 32 bits, los conjuntos necesitan $log_2 128 = 7$ bits en la dirección y con 128 bytes por línea hacen falta $log_2 128 = 7$ bits, luego para la etiqueta quedan $32 - 14 = 18$ bits.
4. Se necesitan 2 bits en cada entrada para determinar la antigüedad de la línea (asociativa de 4 conjuntos). La caché con escritura directa necesita 1 bit de validez, pero no el bit de actualización, de esta forma el tamaño de cada etiqueta será 128+2+1=21 bits debido a que hay 512 líneas el tamaño de las etiquetas es, 512*21=10752, y la caché con post escritura necesita un bit más por entrada = 22 bits

5.5. Problema 5

Un computador tiene una unidad de memoria de 64K × 16 bits y una memoria caché de 1K × 16 bits. La memoria caché utiliza correspondencia directa, con un tamaño de partición de 4 palabras.

1. ¿Cuántos bits hay en los diferentes campos del formato de dirección?
2. ¿Cuántas particiones contiene la memoria caché?
3. ¿Cuántos bits hay en cada partición de la memoria caché, y cómo se dividen según su función?

Solución:

1. Como la memoria tiene una capacidad total de 64 KB tendremos para direccionarla $log_2 64K = 16$bits, y la memoria caché $log_2 1K = 10$bits. Si además hay 4 palabras por partición, serán necesarios 2 bits para identificarla, entonces como la memoria caché se direcciona con 10bits totales y 2 indican la palabra, la partición viene dada por $10 - 2 = 8$bits. Quedando la estructura como sigue:

Etiqueta=6bits	Partición=8bits	Palabra=2bits

2. Al contener 4 palabras cada partición, el número total resulta $\frac{1K}{4} = 256$ particiones.
3. Si cada partición contiene 4 palabras, que ocupan 2 Bytes cada una, 6 bits de etiqueta y tendría una serie de bits de control pero como la caché es de correspondencia directa, su uso no es necesario, por tanto el número total de bits/partición es: $6 + 16 * 4 = 70$bits.

5.6. Problema 6

Para una caché con los mismos parámetros que la presentada en el problema 5, calcular el número de conjuntos que serían analizados para determinar si cada una de las siguientes direcciones están contenidas en la caché, y obtener el byte dentro de la línea de cada una de estas direcciones. Suponga que los bits utilizados para seleccionar un byte dentro de una línea son los bits de menor orden dentro de la dirección y que los bits utilizados para seleccionar el conjunto son los siguientes en la dirección.

1. OxABC89987
2. Ox32651987
3. Ox228945DB
4. Ox48569CAC

Solución:

1. OxABC89987 en binario es 10101011110010001 0110011 0000111 con lo cual tenemos el conjunto 51 y palabra 7.
2. Ox32651987 los bits de menor peso son los mismos con lo cual tenemos la palabra Ox7 y el conjunto Ox33=51
3. 0x228945DB donde nos quedamos con 45db que traducidos son la palabra Ox5B=91 y el conjunto OxB=11
4. Ox48569CAC finalmente esta dirección se refiere a la palabra Ox2C=44 y conjunto Ox39=57

5.7. Problema 7

En esta serie de direcciones de palabra: 1,4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17 y suponiendo una caché con 16 bloques que está inicialmente vacía, etiquetar cada referencia como acierto o fallo en la caché y mostrar los contenidos finales de la misma en lo siguientes casos:

1. Memoria de correspondencia directa con bloques de una palabra.
2. Memoria de correspondencia directa con bloques de cuatro palabras.
3. caché asociativa por conjuntos de dos vías con bloques de una palabra y política de reemplazo LRU.
4. Memoria completamente asociativa con bloques de una palabra, usando la política de reemplazo LRU.
5. Memoria completamente asociativa con bloques de 4 palabras y política de reemplazo LRU.

5.8. Problema 8

Suponga que se necesitan 2,5 ns para acceder a las etiquetas en una caché asociativa por conjuntos, 4 ns para acceder a los datos, 1 ns para realizar la comprobación de acierto/fallo y 1 ns para devolver los datos seleccionados por el procesador en caso de acierto.

1. ¿Es el tiempo para determinar si ha existido acierto o el acceso a los datos el camino crítico en un acierto de caché?
2. ¿Cuál es la latencia de acierto de la caché?
3. ¿Cuál sería la latencia de acierto en la caché si tanto el tiempo de acceso a las etiquetas como a la matriz de datos fuese 3 ns?

Solución:

1. Para determinar si ha habido acierto se necesita acceder a las etiquetas y compararlas, lo cual nos lleva 2,5 ns y 1 ns respectivamente y el acceder a los datos necesita 4 ns, que es el camino crítico.
2. la latencia de acierto viene dada por el tiempo del camino crítico (4ns) y el necesario para devolver el dato (1 ns) con lo cual la latencia es de 6 ns.
3. si tuviésemos un tiempo de acceso a etiquetas y datos de 3 ns la latencia sería:
   - tiempo de acceso a etiquetas y matriz = 3 ns.
   - tiempo de comparación acierto/fallo = 1 ns.
   - tiempo para devolver el dato = 1 ns.

   lo cual da un tiempo de 5 ns.

5.9. Problema 9

Suponga que una caché tiene un tiempo de acceso (tiempo de acierto en caché) de 10 ns y una tasa de fallos del 5 %. Si se realiza un cambio en la caché se disminuye la tasa de fallos al 3 %, pero se incrementa el tiempo de acierto en caché a 15 ns. ¿Bajo qué condiciones este cambio supondría una mejora de las prestaciones, es decir un tiempo medio de acceso a memoria menor?

Solución:

El tiempo medio de acceso viene dado por: $T_{acierto} * TA + TF * P$.

1. En el primer caso tenemos $T = 10ns * 0,95 + P * 0,05$
2. En el segundo tenemos $T = 15ns * 0,97 + P * 0,03$

Si ahora igualamos las ecuaciones podemos encontrar una solución para P.

$10 * 0,95 + 0,05 * P = 15 * 0,97 + 0,03 * P$ lo que nos da un valor $P = 252,5ns$, con lo cual si $P > 252,5ns$ el cambio produce una mejora. En caso contrario la reducción en la tasa de fallo no compensa el incremento en el tiempo de acierto.

5.10. Problema 10

Una caché tiene una tasa de acierto del 95 %, líneas de 128 bytes y un tiempo de acierto de 5 ns. La memoria principal necesita 100 ns para devolver la primera palabra (32 bits) de una línea y 10 ns para devolver las siguientes palabras.

1. ¿Cuál es la tasa de fallo de esta caché? Suponga que la caché espera hasta que la línea ha sido captada en la caché y entonces vuelve a ejecutar la operación de memoria, produciendo un acierto de caché. Desprecie el tiempo requerido para escribir la línea en la caché una vez que ha sido captada desde la memoria principal. Suponga que la caché requiere el mismo tiempo para detectar que ha ocurrido un fallo o para manipular un acierto de caché.
2. Si se duplica el tamaño de la longitud de línea se reduce la tasa de fallos al 3 %, ¿se reduce con ello el tiempo medio de acceso a memoria?

Solución:

1. 128 bytes son 32 palabras, con lo cual tenemos 32 palabras por línea, suponiendo que cada palabra son 4 bytes, y hay una tasa de fallo dada por :

   $T_{fallo} = T_{acierto} + P = 5 + 100 + 31 * 10 + 5 = 420ns$ donde 5 ns es lo que tarda el detector de fallo, 100+31*10 es el tiempo de penalización y leer con acierto son 5 ns.

2. En este caso duplicar la línea de caché implica $P = 5 + 100 + 63 * 10 + 5 = 740ns$

   en el primer caso el tiempo de acceso medio a memoria es $0,95 * 5 + 0,05 * 420 = 25,75$ns y ahora con el nuevo P y tasa de acierto es $0,97 * 5 + 0,03 * 740 = 27,1$ns de