TECNICAS AVANZADAS EN PARALELISMO

4.1.1. Sea un procesador RISC con estructura Harvard, segmentado en cuatro etapas: 1) Captura, 2) Decodificación/Detección de riesgos/Emisión, 3) Ejecución y 4) Escritura. Para todas las instrucciones cada una de las etapas tardan un ciclo, excepto la etapa 3 de la instrucción LOAD que tarda cinco y no hay adelantamientos. Considerar la siguiente secuencia de instrucciones, donde el primero de los operandos indica el registro destino.

I0: ADD R3, R1, R2
I1: LOAD R6, (R3)
I2: AND R7, R5, 3
I3: ADD R1, R6, 20
I4: SRL R7, R0, 8
I5; OR R2, R4, R7
I6: SUB R5, R3, R4
I7: ADD R0, R1, 10
I8: LOAD R6, (R5)
I9: SUB R2, R1, R6
I10: AND R3, R7, 15

Se pide:

a) Indicar los riesgos RAW, WAR y WAW que puedan ocurrir en la ejecución del programa.

b) Escribir el cronograma de uso del procesador y calcular el número de ciclos necesarios para ejecutar el programa, sabiendo que las instrucciones se ejecutan en el orden secuencial en el que han sido escritas. Cuando se detecta un riesgo de los anteriores, el sistema se detiene hasta que se soluciona el problema.

c) Para optimizar el rendimiento de este sistema, la etapa de ejecución (S3) se subdivide en dos unidades independientes, una para las operaciones con memoria y la otra para operaciones con registros. Se supone, por tanto, un cauce escalar sencillo que sólo emite una instrucción al tiempo, pero que permite ejecución desordenada y tiene ventana de instrucciones ilimitada. En este sistema se puede completar la siguiente tabla, que ya incluye las primeras 7 instrucciones, en donde el número indica el ciclo de reloj en el que cada instrucción inicia cada fase.

INSTRUCCIÓN	I0	I1	I2	I3	I4	I5	I6
CAPTURA	0	1	2	3	4	5	6
DECODIFICACION	1	2	3	4	5	6	7
EJECUCIÓN	2	4	5	10	6	8	11
ESCRITURA	3	9	6	11	7	10	12

Se pide: completar la tabla anterior para las 10 instrucciones del programa y

d) repetir el cálculo anterior con su cronograma correspondiente, suponiendo una implementación superescalar que puede manejar dos instrucciones a la vez en todas las etapas.

SOLUCION:

a) Los riesgos posibles por dependencia de datos son:

RIESGOS RAW		RIESGOS WAR	RIESGOS WAW
I1 con I0 por R3	I9 con I3 por R1	I3 con I0 por R1	I4 con I2 por R7
I3 con I1 por R6	I9 con I8 por R6	I5 con I0 por R2	I8 con I1 por R6
I5 con I4 por R7	I10 con I4 por R7	I6 con I2 por R5	I9 con I5 por R2
I6 con I0 por R3		I7 con I4 por R0	I10 con I0 por R3
I7 con I3 por R1		I8 con I3 por R6	
I8 con I6 por R5		I9 con I0 por R2	
		I10 con I1 por R3	
		I10 con I6 por R3	

b) * = Usa memoria. w = Ejecutada sin poder avanzar. D = Retenida por riesgo de datos o estructural, en espera de ser ejecutada

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21	T22	T23	T24
S1	I0*	I1*	I2*	I2w	I3*	I3w	I3w	I3w	I3w	I4*	I5*	I6*	I6w	I7*	I8*	I9*	I10*	I10w	I10w	I10w	I10w	I10w			
S2		I0	I1D	I1	I2	I2w	I2w	I2w	I2w	I3	I4	I5	I5D	I6	I7	I8	I9D	I9D	I9D	I9D	I9D	I9	I10		
S3			I0	-	I1*	I1*	I1*	I1*	I1*	I2	I3	I4	-	I5	I6	I7	I8*	I8*	I8*	I8*	I8*	-	I9	I10	
S4				I0	-	-	-	-	-	I1	I2	I3	I4	-	I5	I6	I7	-	-	-	-	I8	-	I9	I10

c) Por ser un cauce único, cada segmento sólo puede comenzar/ejecutar una única instrucción al tiempo. En negrita se indica el ciclo de finalización en cada segmento

	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20
S1	0*	1*	2*	3*	4*	5*	6*	7*	8*	9*	10*										
S2		0	1^D	1 2^D	2 3^D	3^D 4	3^D 5^D	3^D 5 6^D	3^D 6^D 7^D	3 6^D 7^D 8^D	6 7^D 8^D 9^D	7 8^D 9^D 10^D	8 9^D 10^D	9^D 10	9^D	9^D	9^D	9^D	9		
S3			0			2	4		5^D	5	3	6	7		10					9	
					1*	1*	1*	1*	1*					8*	8*	8*	8*	8*			
S4				0	-	-	2	4	-	1	5	3	6	7	-	10	-	-	8	-	9

INSTRUCCIÓN	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
CAPTURA	0	1	2	3	4	5	6	7	8	9	10
DECODIFICACION	1	2	3	4	5	6	7	8	9	10	11
EJECUCIÓN	2	4	5	10	6	8	11	12	13	19	14
ESCRITURA	3	9	6	11	7	10	12	13	18	20	15

d) Por ser un cauce superescalar de grado 2, cada segmento puede comenzar/ejecutar dos instrucciones.

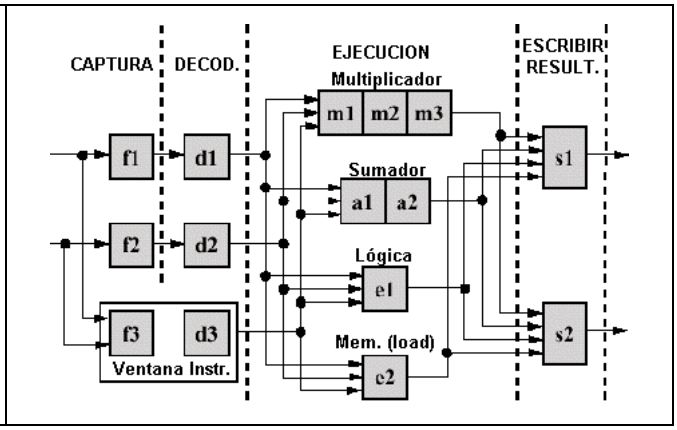
	T0	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
S1	0* 1*	2* 3*	4* 5*	6* 7*	8* 9*	10 *									
S2		0 1^D	1^D 2 3^D	1 3^D 4 5^D	3^D 5^D 6 7^D	3^D 5 7^D 8^D 9^D	3^D 7^D 8 9^D 10	3^D 7^D 9^D	3^D 7^D 9^D	3 7^D 9^D	7^D 9^D	7 9^D	9		
S3			0	2	4	6	5	10			3		7	9	
					1*	1*	1*	1*	1*						
								8*	8*	8*	8*	8*			
S4				0	2	4	6	5	10	1	-	3	8	7	9

INSTRUCCIÓN	I0	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
CAPTURA	0	0	1	1	2	2	3	3	4	4	5
DECODIFICACION	1	1	2	2	3	3	4	4	5	5	6
EJECUCIÓN	2	4	3	10	4	6	5	12	7	13	7
ESCRITURA	3	9	4	11	5	7	6	13	12	14	8

TECNICAS AVANZADAS EN PARALELISMO

4.1.2. Sea un procesador superescalar de cuatro etapas como el mostrado en la figura. La etapa de decodificación es la encargada de detector los posibles riesgos. El procesador puede emitir dos instrucciones por ciclo siempre que no haya conflictos con los recursos ni con dependencias de datos. La ventana de anticipación asociada a las dos primeras etapas, solo está disponible cuando se permite la emisión desordenada de instrucciones. No hay adelantamiento de datos pero se permite leer y escribir a la vez del mismo registro. En caso de que sea necesario, se elegirá una instrucción anterior para avanzar de etapa con respecto a una instrucción posterior. Considerar el siguiente programa a ejecutar en donde el primer operando es el registro destino:

I1:	LD	R1,A	; R1 ⇐ [A]
I2:	ADD	R2,R5,R1	; R2 ⇐ R5 + R1
I3:	MUL	R3,R5,R4	; R3 ⇐ R5 x R4
I4:	ADD	R4,R8,R5	; R4 ⇐ R8 + R5
I5:	CMP	R6	; R6 ⇐ R6 - 0
I6:	MUL	R6,R5,R7	; R6 ⇐ R5 x R7



a) Señalar las dependencias que aparecen en el programa.
b) Escribir una tabla indicando para cada instrucción, el ciclo de comienzo en cada uno de los segmentos, para cada uno de los tres casos siguientes: **1:** Una política de emisión y finalización en orden (las instrucciones siempre se capturan a pares y se decodifican a pares. También se emiten a pares a no ser que algún riesgo obligue a romper la pareja. Instrucciones que se emiten a pares tienen que finalizar juntas.), **2:** una política de emisión en orden y finalización desordenada (las instrucciones siempre se capturan a pares y se decodifican a pares. También se emiten a pares a no ser que algún riesgo obligue a romper la pareja.), y **3:** una política de emisión y finalización desordenada.

SOLUCION:

a) Riesgos RAW: I2 con I1 por R1.
Riesgos WAR: I4 con I3 por R4, I6 con I5 por R6.
Riesgos WAW: I6 con I5 por R6

b1) I2 no se emite con I1 por un riesgo RAW, por lo tanto se deshace la pareja.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
CAP	I1* I2*	I3* I4*	$I3^W$ $I4^W$	$I3^W$ $I4^W$	I5* I6*					
DEC		I1 $I2^D$	$I2^D$	I2	I3 I4	I5 I6				
M1						I3	I6			
M2							I3	I6		
M3								I3	I6	
A1					I2	I4	I5	$I5^W$		
A2						I2	I4	$I4^W$	I5	
E1										
E2			I1							
WR				I1			I2		I3 I4	I5 I6

INSTRUCCIÓN	I1	I2	I3	I4	I5	I6
CAPTURA	1	1	2	2	5	5
DECODIFICACION	2	2	5	5	6	6
EJECUCIÓN	3	5	6	6	7	7
ESCRITURA	4	7	9	9	10	10

b2) I2 no se emite con I1 por un riesgo RAW, por lo tanto se deshace la pareja.

TECNICAS AVANZADAS EN PARALELISMO

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
CAP	I1* I2*	I3* I4*	$I3^{w}$ $I4^{w}$	$I3^{w}$ $I4^{w}$	I5* I6*					
DEC		I1 $I2^{D}$	$I2^{D}$	I2	I3 I4	I5 I6				
M1						I3	I6			
M2							I3	I6		
M3								I3	I6	
A1					I2	I4	I5			
A2						I2	I4	I5		
E1										
E2			I1							
WR				I1			I2	I4	I3 I5	I6

INSTRUCCIÓN	I1	I2	I3	I4	I5	I6
CAPTURA	1	1	2	2	5	5
DECODIFICACION	2	2	5	5	6	6
EJECUCIÓN	3	5	6	6	7	7
ESCRITURA	4	7	9	8	9	10

b3) I5 no se emite con I6 por un riesgo estructural en el sumador. Al permitir emisión fuera de orden se deshace esta pareja pero se emite simultáneamente una nueva I2 con I6.

	T1	T2	T3	T4	T5	T6	T7	T8
CAP	I1* I2*	I3* I4*	I5* I6*					
DEC		I1 $I2^{D}$	$I2^{D}$ I3 I4	I2 I5 I6	$I5^{w}$			
M1				I3	I6			
M2					I3	I6		
M3						I3	I6	
A1				I4	I2	I5		
A2					I4	I2	I5	
E1								
E2			I1					
WR				I1		I4	I2 I3	I5 I6

INSTRUCCIÓN	I1	I2	I3	I4	I5	I6
CAPTURA	1	1	2	2	3	3
DECODIFICACION	2	2	3	3	4	4
EJECUCIÓN	3	5	4	4	6	5
ESCRITURA	4	7	7	6	8	8

4.1.3. Usar la técnica de renombramiento de registros para eliminar todas las dependencias de datos del siguiente código, en donde F y R son registros internos del sistema:

I1:	Loop: LD F0, 1000(R1)	**I5:**	SUB R1, R1, 8
I2:	ADD F4, F0, F2	**I6:**	ADD F8, F4, F0
I3:	MUL F2, F6, F8	**I7:**	ADD F5, F2, F0
I4:	ADD F0, F10, F12	**I8:**	DIV F2, F2, F1

SOLUCION:

a) La técnica de renombrar registros se utiliza para eliminar los riesgos WAR y WAW.

I1: Loop: LD F0, 1000(R1)
I2: ADD F4, F0, F2
I3: MUL F2', F6, F8
I4: ADD F0', F10, F12
I5: SUB R1', R1, 8
I6: ADD F8', F4, F0'
I7: ADD F5, F2', F0'
I8: DIV F2'', F2', F1

TECNICAS AVANZADAS EN PARALELISMO

4.1.4. Suponer un sistema superescalar que emite una instrucción por ciclo. El sistema dispone de una unidad para suma en CF con dos segmentos (A1 y A2). Una unidad para multiplicar en CF con tres estados (M1, M2 y M3) y una unidad caché también segmentada que necesita tres ciclos para completar su operación (C1, C2 y C3). Se ejecuta el siguiente código:

I1:	LD	F0, 0(R2)	**I5:**	ADDF	F12, F6, F0
I2:	MULF	F6, F4, F2	**I6:**	MULF	F14, F6, F2
I3:	LD	F8, 0(R3)	**I7:**	LD	F8, 0(R4)
I4:	ADDF	F10, F8, F0	**I8:**	ADDF	F16, F8, F0

Añadir una etapa previa S1, donde se decodifica, se capturan operandos y cuando sea posible, se emite para su ejecución la instrucción. Añadir también una etapa final S3 en donde se escriben los resultados. Se pide:

a) Escribir el cronograma del sistema para este código emitido secuencialmente.

b) Reordenar la emisión de instrucciones para mejorar el rendimiento, utilizando si fuera necesario la técnica de renombrar registros.

NOTA: Suponer en ambos casos, que se permite la escritura/lectura en/desde los registros internos en un mismo ciclo.

SOLUCION:

a) Ejecutando secuencialmente y teniendo en cuenta los riesgos de datos se necesitan 17 ciclos en total. En la tabla se indica para cada instrucción el ciclo de los estados inicial S1 y final S3.

INSTRUC.	I1	I2	I3	I4	I5	I6	I7	I8
S1	1	2	3	7	8	9	10	14
S3	5	6	7	10	11	13	14	17

b) Si se renombran los registros se eliminan los riesgos de datos del tipo WAR y WAW. Además si se permite la ejecución fuera de orden se mejora el sistema necesitándose 13 ciclos, 4 menos que en el caso anterior.

INSTRUC.	I1	I2	I3	I7	I5	I4	I8	I6
S1	1	2	3	4	6	7	8	9
S3	5	6	7	8	9	10	11	13

TECNICAS AVANZADAS EN PARALELISMO

4.1.5. La secuencia de código realiza la operación MAC, que consiste en la multiplicación por constante y acumulación sobre los elementos de un vector (Y(i) = a·X(i) + Y(i)). Se ejecuta en un sistema con arquitectura Von Neumann, con una unidad de tratamiento que segmenta la ejecución de cada una de las instrucciones en las siguientes etapas: **IF** (captura instrucción), **ID** (decodifica, detecta riesgos, captura operandos y calcula el destino en las instrucciones de salto). **E** (ejecuta las instrucciones aritméticas tanto para enteros como para CF, comprueba la condición en saltos y calcula direcciones efectivas). Esta etapa emplea un ciclo de reloj salvo en las operaciones en **CF**, que necesitan tres ciclos. **M** (accede a memoria) y **W** (escribe en el banco de registros). La unidad de control predice estáticamente todo los saltos que decrementan el PC como efectivos y el resto como no efectivos. Todas las instrucciones pasan por todas las etapas.

```
I1      ADD    R3, R0, 32      ; R3 = 32                ; i = 4
I2 L1:  LOAD   F4, 0(R1)       ; F4 = M(R1+0)           ; X(i)
I3      MULF   F6, F4, F2      ; F6 = F4*F2             ; aX(i)
I4      LOAD   F8, 0(R2)       ; F8 = M(R2+0)           ; Y(i)
I5      ADDF   F8, F6, F8      ; F8 = F6+F8             ; a·X(i)+Y(i)
I6      STORE  0(R2), F8       ; M(R2+0) = F8
I7      ADD    R1, R1, 8       ; R1 = R1+8
I8      ADD    R2, R2, 8       ; R2 = R2+8
I9      BNE    R3, R1, L1      ; Si R3 es distinto de R1, ir a L1
```

Inicialmente el contador de programa contiene la dirección de la instrucción I1. Suponga que inicialmente el valor de F2 es la constante "a", el registro R1 = 0, el registro R2= 8*i (con i=4) y el registro R0 está definido con valor 0.

a) Analice todos los riesgos de datos presentes en el código, suponiendo que pueda ser ejecutado por cualquier tipo de procesador.

b) Utilizando el cronograma adjunto muestre la evolución temporal de la primera iteración de la secuencia de instrucciones. En este apartado no considere ningún tipo de adelantamiento salvo que las etapas **ID** y **W** pueden acceder en el mismo ciclo de reloj al banco de registros (la etapa **W** accede en la primera mitad y la etapa **ID** en la segunda mitad).

c) Repita el apartado (b) pero considerando el adelantamiento de datos entre etapas mas eficiente posible. Marque con una flecha en el cronograma entre que etapas se produce adelantamiento de datos.

d) Se mejora el sistema del apartado anterior, añadiendo en la etapa de ejecución otra unidad funcional (UF) para la ejecución de las operaciones de CF, independiente de la UF de enteros. Pese a esta mejora se mantiene un cauce escalar simple en donde la emisión y finalización de las instrucciones es en orden. Indique en el cronograma adjunto la ejecución de las instrucciones

SOLUCION:

a) Nótese que en las instrucciones de salto condicional los registros actúan como operando fuente.

RAW	WAR	WAW
I3 con I2 por F4	I6 con I4 por M(R2+0)	I5 con I4 por F8
I5 con I3 por F6	I7 con I2 por R1	
I5 con I4 por F8	I8 con I6 por R2	
I6 con I4 por F8 (*)	I8 con I4 por R2	
I6 con I5 por F8		
I9 con I1 por R3		
I9 con I7 por R1		
En iteraciones sucesivas	En iteraciones sucesivas	En iteraciones sucesivas
I2' con I7 por R1	I2' con I3 por F4	I2' con I2 por F4
I2'' con I7' por R1	I3' con I5 por F6	…
...	I4' con I6 por F8	
I4' e I6 con I8 por R2	…	
I4'' e I6'' con I8' por R2		

En procesadores VLIW WAR de I6 con I2 por posible acceso a la misma posición de memoria y en iteraciones sucesivas RAW de I2' con I6,…

b) ¿En qué ciclo comienza la segunda iteración?: Ciclo 22

Instrucciones	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

TECNICAS AVANZADAS EN PARALELISMO

I1: ADD R3, R0, 32	IF*	ID	E	M	W																		
I2: L1:LOAD F4, 0(R1)		**IF***	**ID**	**E**	**M***	**W**																**IF***	**ID**	**E**
I3: MULF F6, F4, F2			**IF***	**id**	**id**	**ID**	**E1**	**E2**	**E3**	**M**	**W**												**IF***	**id**
I4: LOAD F8, 0(R2)				**IF***	**if**	**if**	**ID**	**id**	**id**	**E**	**M***	**W**												**IF***
I5: ADDF F8,F6,F8							**IF***	**if**	**if**	**id**	**id**	**ID**	**E1**	**E2**	**E3**	**M**	**W**							
I6: STORE 0(R2),F8										**IF***	**if**	**if**	**id**	**id**	**id**	**id**	**ID**	**E**	**M***	**W**				
I7: ADD R1,R1,8													**IF***	**if**	**if**	**if**	**if**	**ID**	**E**	**M**	**W**			
I8: ADD R2,R2,8																		**IF***	**ID**	**E**	**M**	**W**		
I9: BNE R3,R1, L1																			*	**IF***	**ID**	**E**	**M**	**W**

c) ¿En qué ciclo comienza la segunda iteración?: Ciclo 18

Instrucciones	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
I1: ADD R3, R0, 32	IF*	ID	E	M	W																	
I2: L1:LOAD F4, 0(R1)		**IF***	**ID**	**E**	**M***	**W**												**IF***	**ID**	**E**	**M**	**W**
I3: MULF F6, F4, F2			**IF***	**ID**	**id**	**E1**	**E2**	**E3**	**M**	**W**												
I4: LOAD F8, 0(R2)				**IF***	**if**	**ID**	**id**	**id**	**E**	**M***	**W**											
I5: ADDF F8,F6,F8						**IF***	**if**	**if**	**id**	**ID**	**E1**	**E2**	**E3**	**M**	**W**							
I6: STORE 0(R2),F8									**IF***	**if**	**ID**	**id**	**id**	**E**	**M***	**W**						
I7: ADD R1,R1,8											**IF***	**if**	**if**	**ID**	**E**	**M**	**W**					
I8: ADD R2,R2,8														**IF***	**ID**	**E**	**M**	**W**				
I9: BNE R3,R1, L1															*	**IF***	**ID**	**E**	**M**	**W**		

d) ¿En qué ciclo comienza la segunda iteración?: Ciclo 18

Instrucciones	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**
I1: ADD R3, R0, 32	IF*	ID	E1	M	W																	
I2: L1:LOAD F4, 0(R1)		**IF***	**ID**	**E1**	**M***	**W**												**IF***	**ID**	**E1**	**M**	**W**
I3: MULF F6, F4, F2			**IF***	**ID**	**ID**	**E2**	**E2**	**E2**	**M**	**W**												
I4: LOAD F8, 0(R2)				**IF***	**if**	**ID**	**E1**	**e1**	**e1**	**M***	**W**											
I5: ADDF F8,F6,F8						**IF***	**id**	**id**	**id**	**ID**	**E2**	**E2**	**E2**	**M**	**W**							
I6: STORE 0(R2),F8							**IF***	**if**	**if**	**if**	**ID**	**E1**	**e1**	**e1**	**M***	**W**						
I7: ADD R1,R1,8											**IF***	**ID**	**id**	**id**	**E1**	**M**	**W**					
I8: ADD R2,R2,8												**IF***	**if**	**if**	**ID**	**E1**	**M**	**W**				
I9: BNE R3,R1, L1															*	**IF***	**ID**	**E1**	**M**	**W**		

Aunque no se pide en el problema se anexa una solución que corresponde a un procesador que permite finalización fuera de orden y tiene en cuenta todas las dependencias para no alterar el resultado final de la ejecución de la secuencia de código.

TECNICAS AVANZADAS EN PARALELISMO

Instrucciones	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
I1: ADD R3, R0, 32	IF*	ID	E1	M	W															
I2: L1:LOAD F4, 0(R1)		**IF***	**ID**	**E1**	**M***	**W**										**IF***	**ID**	**E 1**	**M**	**W**
I3: MULF F6, F4, F2			**IF***	**ID**	**id**	**E2**	**E2**	**E2**	**M**	**W**										
I4: LOAD F8, 0(R2)				**IF***	**if**	**ID**	**E1**	**M***	**W**											
I5: ADDF F8,F6,F8						**IF***	**ID**	**id**	**E2**	**E2**	**E2**	**M**	**W**							
I6: STORE 0(R2),F8							**IF***	**if**	**ID**	**id**	**id**	**E1**	**M***	**W**						
I7: ADD R1,R1,8									**IF***	**if**	**if**	**ID**	**E1**	**M**	**W**					
I8: ADD R2,R2,8												**IF***	**ID**	**E1**	**M**	**W**				
I9: BNE R3,R1, L1													*	**IF***	**ID**	**E1**	**M**	**W**		

¿En qué ciclo comienza la segunda iteración?: Ciclo 16

TECNICAS AVANZADAS EN PARALELISMO

4.1.6. Un sistema con paralelismo a nivel de instrucción, dispone de dos unidades de ejecución idénticas con un ciclo de latencia (esto es, la ejecución y la escritura de resultados tarda un ciclo). El sistema permite emitir dos instrucciones al tiempo siempre que no exista entre ambas ningún tipo de riesgo. Señalar, justificando la respuesta indicando en cada caso la secuencia de ejecución, el número de ciclos empleado para la ejecución del siguiente fragmento de código:

```
I1: LD   R1, (R2)        ; Mem[R2] => R1
I2: SUB  R4, R5, R6      ; R5 – R6 => R4
I3: ADD  R3, R1, R7      ; R1 + R7 => R3
I4: MUL  R8, R3, R3      ; R3 x R3 => R8
I5: ST   (R11), R4       ; R4 => Mem[R11]
I6: ST   (R12), R8       ; R8 => Mem[R12]
I7: ADD  R15, R14, R13   ; R14 + R13 => R15
I8: SUB  R10, R15, R10   ; R15 – R10 => R10
I9: ST   (R9), R10       ; R10 => Mem[R9]
```

a) Si la ejecución es en orden

b) Si la ejecución es fuera de orden y el sistema dispone de un ventana de instrucciones suficientemente grande para alojar todo el fragmento de código

SOLUCION:

a) Ejecución en orden tarda 6 ciclos:

Ciclo 1:	LD	R1, (R2)	SUB	R4, R5, R6
Ciclo 2:	ADD	R3, R1, R7		
Ciclo 3:	MUL	R8, R3, R3	ST	(R11), R4
Ciclo 4:	ST	(R12), R8	ADD	R15, R14, R13
Ciclo 5:	SUB	R10, R15, R10		
Ciclo 6:	ST	(R9), R10		

b) Ejecución fuera de orden tarda 5 ciclos:

Ciclo 1:	LD	R1, (R2)	SUB	R4, R5, R6
Ciclo 2:	ADD	R3, R1, R7	ST	(R11), R4
Ciclo 3:	MUL	R8, R3, R3	ADD	R15, R14, R13
Ciclo 4:	ST	(R12), R8	SUB	R10, R15, R10
Ciclo 5:	ST	(R9), R10		

TECNICAS AVANZADAS EN PARALELISMO

4.1.7. Se dispone de un procesador superescalar organizado en cuatro etapas como el que se muestra la figura. Las cuatro etapas son:

- **F:** Captura de instrucciones
- **D:** Decodificación, detección de riesgos (estructurales y de datos), captura de operandos y emisión.
- **E:** Ejecución
- **W:** Escritura en registros

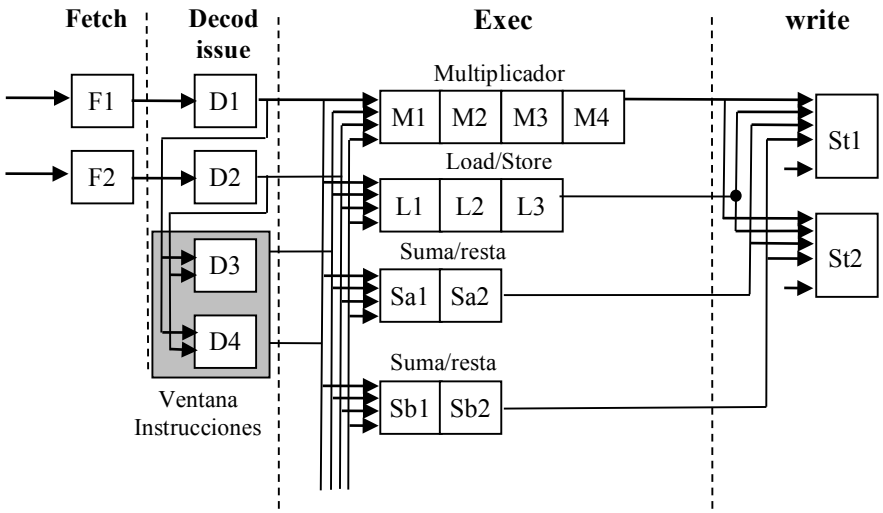
Las rutas de datos de las unidades funcionales son segmentadas (en cada ciclo puede comenzar una nueva instrucción). La unidad de ejecución cuenta con diferentes tipos de unidades funcionales, pero para este problema sólo se consideran las cuatro presentes en el gráfico (multiplicación 4 ciclos, load/store 3 ciclos y dos unidades de suma-resta que necesitan de dos ciclos).

El procesador permite emitir hasta dos instrucciones por ciclo siempre que no existan conflictos con los recursos ni con dependencias de datos. Gracias a la ventana de instrucciones asociada a las dos primeras etapas, se permite la emisión fuera de orden de instrucciones y finalización fuera de orden.

Notas: No se emite una instrucción hasta que se hayan capturado los operandos. La captura de operandos se puede llevar a cabo en el mismo ciclo en que se escribe el registro. Es decir en el ciclo que se escribe se captura el operando y al siguiente se puede comenzar a ejecutar. A efectos de avance de las instrucciones, se elegirá una instrucción capturada (o que esté en el código antes) con anterioridad con respecto a una capturada con posterioridad (o que esté en el código después) para el avance de la primera en el pipeline.

Evalúe la ejecución del siguiente trozo de código C, que ha sido compilado en un ensamblador y ha dado como resultado el código que se muestra:

Código C		**Ensamblador**		
	I1:	**LD**	**R1, A**	*; R1 = M[A]*
	I2:	**LD**	**R2, B**	*; R2 = M[B]*
E = (A*B) + (C*D);	**I3:**	**MUL**	**R5, R1, R2**	*; R5 = R1 * R2*
F = (A+B);	**I4:**	**LD**	**R3, C**	*; R3 = M[C]*
	I5:	**LD**	**R4, D**	*; R4 = M[D]*
	I6:	**MUL**	**R6, R3, R4**	*; R6 = R3 * R4*
	I7:	**ADD**	**R7, R6, R5**	*; R7 = R6 + R5*
	I8:	**ADD**	**R8, R1, R2**	*; R8 = R1 + R2*
	I9:	**ST**	**R7, E**	*; M[E] = R7*
	I10:	**ST**	**R8, F**	*; M[F] = R8*



a) Rellenar una tabla indicando para cada instrucción, el ciclo de comienzo y finalización en cada uno de las etapas.

b) Completar el contenido de las etapas **F** (captura de instrucciones) y **D** (decodificación, detección, captura y emisión) en las figuras adjuntas al final de esta hoja para los 10 primeros ciclos de ejecución.

SOLUCION:

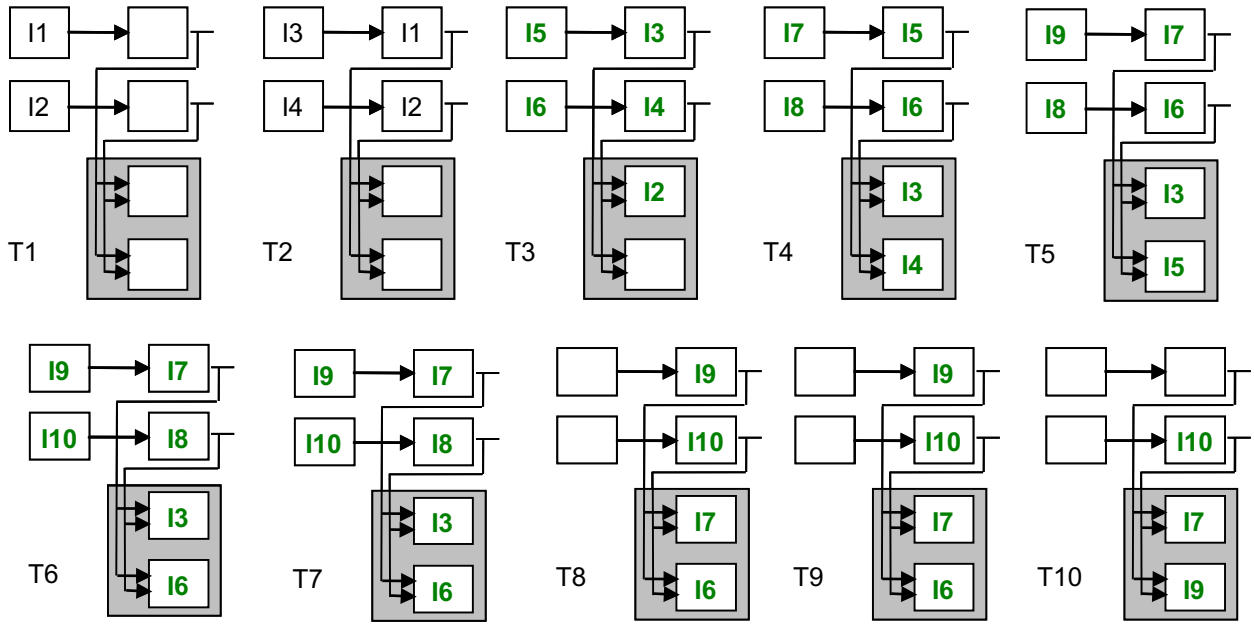
Instruction	**I1**	**I2**	**I3**	**I4**	**I5**	**I6**	**I7**	**I8**	**I9**	**I10**
Fetch	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4-5**	**5-7**	**6-7**
Decod/Emis	**2**	**2-3**(1)	**3-7**(4)	**3-4**(2)	**4-5**(3)	**4-9**(5)	**5-14**(6)	**6-7**(7)	**8-17**(8)	**8-10**(9)
Exec	**3-5**	**4-6**	**8-11**	**5-7**	**6-8**	**10-13**	**15-16**	**8-9**	**18-20**	**11-13**
Write	**6**	**7**	**12**	**8**	**9**	**14**	**17**	**10**	**21**	**14**

1, 2, 3, 5- Existe una única unidad de load-store y en cada ciclo solo puede empezar una operación

TECNICAS AVANZADAS EN PARALELISMO

3, 4, 6-9 - No emite hasta no tener sus operandos

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
I1	F	D	E	E	E	W																
I2	F	(D)	D	E	E	E	W															
I3		F	(D)	(D)	(D)	(D)	D	E	E	E	E	W										
I4		F	(D)	D	E	E	E	W														
I5			F	(D)	D	E	E	E	W													
I6			F	(D)	(D)	(D)	(D)	(D)	D	E	E	E	E	W								
I7				F	(D)	(D)	(D)	(D)	(D)	(D)	(D)	(D)	(D)	D	E	E	W					
I8				(F)	F	(D)	D	E	E	W												
I9					(F)	(F)	F	(D)	(D)	(D)	(D)	(D)	(D)	(D)	(D)	(D)	D	E	E	E	W	
I10						(F)	F	(D)	(D)	D	E	E	E	W								



4.1.8. Se Considere el siguiente fragmento de código de un programa.

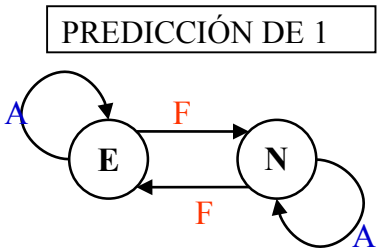
El valor de R0 es 0 y R1 antes de ejecutarse la instrucción de la dirección 5000 llega alternativamente con valor 0 y valor 2.

```
5000:  L0:   BNEZ  R1, L1        ; salto 1 ( salta si R1 es distinto 0)
5004:        ADDI  R1, R0, #1    ;  R1=1;
5008:  L1:   SUBI  R3, R1, #1    ;  R3 = R1-1;
500C:        BNEZ R3, L2         ;  Salto 2 (salta si R3 es distinto de 0);
…
5020:  L2:    …..
…
5040:         JUMP L0            ; salto incondicional al principio
```

Este fragmento de programa se ejecuta 4 veces con los siguientes valores
Primera ejecución con R1= 0;
Segunda ejecución con R1= 2;
Tercera ejecución con R1= 0;
Cuarta ejecución con R1= 2;

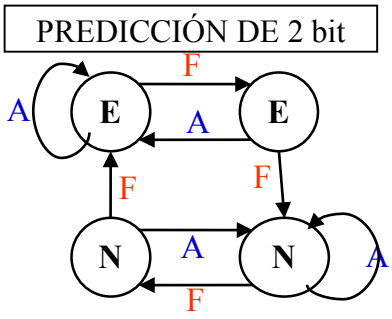
Suponga que la entrada al BHT se realiza utilizando como índice los bits de menor peso de la dirección, quitando previamente los bits necesarios para que la palabra (instrucciones de 32 bits) esté alineada.

Analice el rendimiento con los siguientes predictores



a) BHT de 16 entradas y predictor de 1 bit (predictor (0,1))
Suponga que inicialmente predice que el salto es no-efectivo.

Aciertos: 2/8 = 25%; Fallos: 6/8 = 75%.



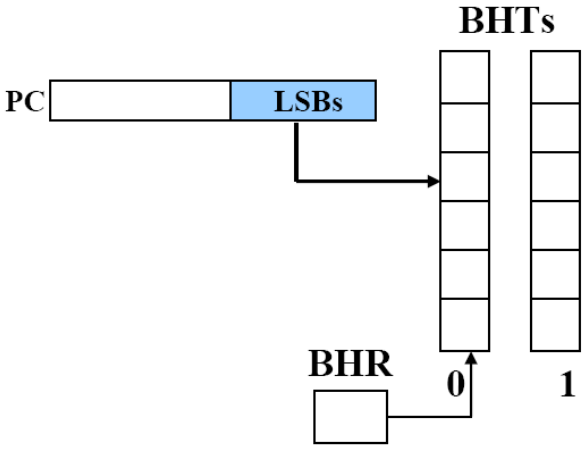
b) BHT de 16 entradas y predicción de dos bits (predictor (0,2))
Suponga que inicialmente predice que el salto es no-efectivo fuerte (Nf).

Aciertos: 4/8 = 50%; Fallos: 4/8 = 50%.

c) Predictor correlacionado con información de un sólo salto anterior (BHR de 1 bit) y BHT de 16 entradas. (predictor (1,1))
Suponga que inicialmente predice que el salto es no-efectivo y todos los valores de BHR y BHT son 0 (NE)

Aciertos: 6/8 = 75%; Fallos: 2/8 = 25%.

d) Si se repite indefinidamente la alternancia de valores en R1, indique en todos los casos el porcentaje de fallos en la predicción de los saltos



Tras entrar en régimen de funcionamiento:

a) Falla siempre.
b) 50 % de fallos
c) 100% acierto.

TECNICAS AVANZADAS EN PARALELISMO

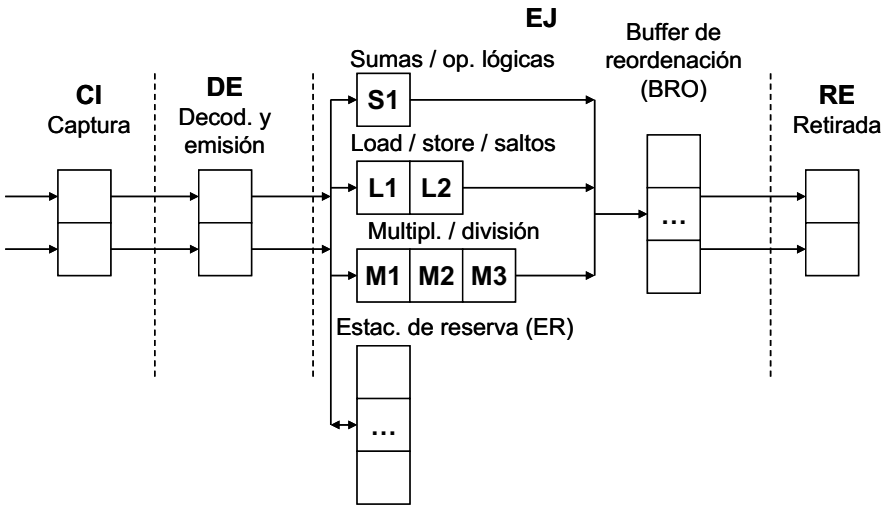
4.1.9. Se tiene un microprocesador superescalar de grado 2, que puede capturar, emitir y retirar hasta 2 instrucciones por ciclo. Dicho micro ejecuta sus instrucciones en 4 etapas:

CI: Captura de instrucciones
DE: Decodificación, detección de riesgos (estructurales y de datos) y emisión.
EJ: Ejecución
RE: Retirada (incluyendo la escritura final en registro o memoria)

Todas las etapas, salvo la de ejecución, se completan en 1 ciclo. En la etapa de ejecución hay 3 unidades funcionales (ver figura). La primera es para instrucciones de suma e instrucciones lógicas, que se completa en 1 ciclo. La segunda es para instrucciones con memoria de datos (load/store) y saltos, que se completa en 2 ciclos. La tercera es para instrucciones de multiplicación o división, que se completa en 3 ciclos. Todas ellas son segmentadas, es decir, cada ciclo de reloj puede empezar su ejecución una instrucción nueva.

La estructura de adelantamiento es tal que una instrucción puede empezar su ejecución el ciclo posterior al último de ejecución de la instrucción que generaba el dato.

Además hay estaciones de reserva (**ER**) y buffer de reordenación (**BRO**), de tal forma que se pueden ejecutar instrucciones fuera de orden. Hacia las estaciones de reserva se emiten aquellas instrucciones ya decodificadas que no pueden empezar su ejecución por riesgos estructurales o de datos, en cuyo caso se quedan a la espera de que llegue el(los) dato(s) que le falta(n). El buffer de reordenación garantiza que las instrucciones sean retiradas en orden (hasta de 2 en 2 por ser superescalar de grado 2). Para efectos de la resolución del ejercicio ambas estructuras se considerarán de capacidad ilimitada.



Dado el código que se muestra a continuación, se pide contestar a las siguientes preguntas.

```
I1:   ADD   R1, R1, 1      ; R1 ⇐ R1 + 1
I2:   LD    R2, 100(R4)    ; R2 ⇐ M[100+R4]
I3:   LD    R3, 104(R4)    ; R3 ⇐ M[104+R4]
I4:   ADD   R5, R1, R2     ; R5 ⇐ R1 + R2
I5:   MUL   R6, R2, R3     ; R6 ⇐ R2 * R3
I6:   ST    R5, 100(R4)    ; M[100+R4] ⇐ R5
I7:   ST    R6, 104(R4)    ; M[104+R4] ⇐ R6
I8:   ADD   R4, R4, 8      ; R4 ⇐ R4 + 8
```

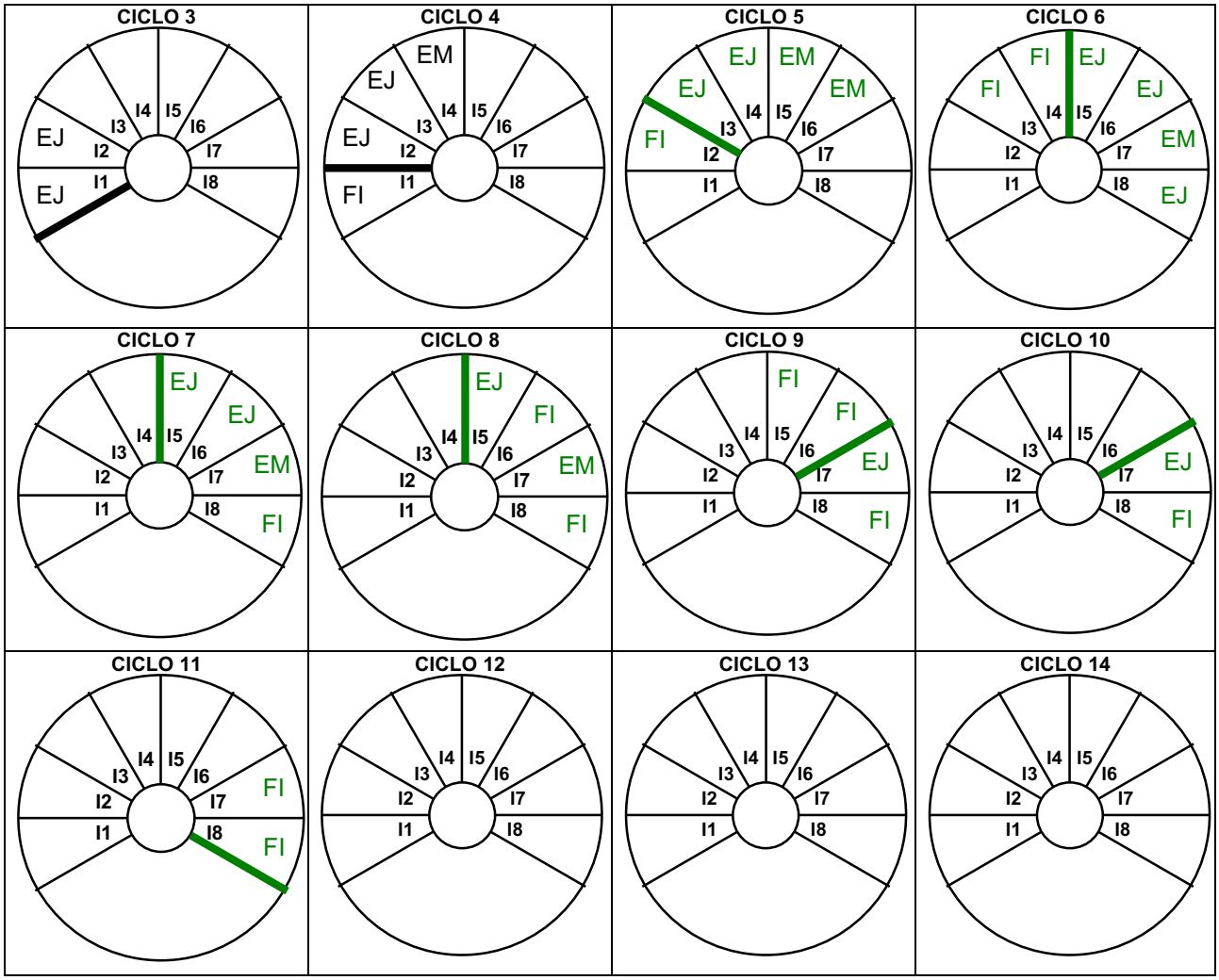
a) Escriba todos los riesgos potenciales de datos, indicando de qué tipo es cada uno.

RAW	**WAR**	**WAW**
I4 con I1 por R1 I4 con I2 por R2 I5 con I2 por R2 I5 con I3 por R3 I6 con I4 por R5 I7 con I5 por R6	I8 con I7, I6, I3 e I2 por R4 I7 con I3 por memoria I6 con I2 por memoria (No se incluyen las combinaciones cruzadas de memoria I7 con I2 ni I6 con I3 porque R4 no varía desde I2 hasta I7)	No se incluye I7 con I6 por memoria porque 100+R4 no puede ser igual a 104+R4 sin una instrucción intermedia que modifique R4

b) Cronograma de ejecución del fragmento de código anterior, indicando con CI la captura de instrucciones, DE la decodificación y emisión, ER las instrucciones emitidas pero en estaciones de reserva, S1, L1, L2, M1, M2, M3 la ejecución (dependiendo de la unidad funcional y segmento de la misma), ROB las instrucciones ya finalizadas pero no retiradas todavía y RE las que se retiran en cada ciclo.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
I1 ADD R1, R1, 1	CI	DE	S1	RE										
I2 LD R2, 100(R4)	CI	DE	L1	L2	RE									
I3 LD R3, 104(R4)		CI	DE	L1	L2	RE								
I4 ADD R5, R1, R2		CI	DE	ER	S1	RE								
I5 MUL R6, R2, R3			CI	DE	ER	M1	M2	M3	RE					
I6 ST R5, 100(R4)			CI	DE	ER	L1	L2	ROB	RE					
I7 ST R6, 104(R4)				CI	DE	ER	ER	ER	L1	L2	RE			
I8 ADD R4, R4, 8				CI	DE	S1	ROB	ROB	ROB	ROB	RE			

c) Estado en cada ciclo de la FIFO circular con que se implementa el buffer de reordenación. Se indicará con EM si la instrucción está emitida, con EJ si está en ejecución y con FI si está finalizada (tanto si está retirada como si no). Se marcará con línea gruesa hasta dónde llega la retirada en cada ciclo.



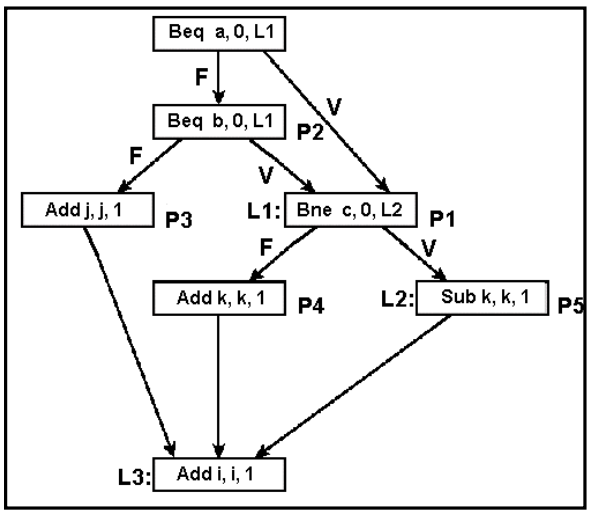
Se eliminan de cada diagrama las instrucciones retiradas en el ciclo anterior ya que, en realidad, suponen huecos libres para que la FIFO circular siga avanzando.

TECNICAS AVANZADAS EN PARALELISMO

4.2.1. Aplicando la técnica de ejecución con predicados diseñada en la arquitectura del IA-64, y para el programa definido por el diagrama de flujo de la figura se pide:

a) Señalar las instrucciones que pueden ejecutarse en paralelo.

b) Indicar las instrucciones que puedan agruparse en el mismo paquete de instrucciones de la arquitectura IA-64 (paquetes de 3 unidades).



SOLUCIÓN:

El diagrama de flujo de la figura implementado en código y su equivalente en código de ejecución con predicados serían:

I1: Beq a, 0, L1	I1:	<P0>	P1, P2 = Cmp (a == 0)
I2: Beq b, 0, L1	I2:	<P2>	P1, P3 = Cmp (b == 0)
I3: Add j, j, 1	I3:	<P3>	Add j, j, 1
I4: Jmp L3	I4:	<P1>	P5, P4 = Cmp (c != 0)
I5: L1: Bne c, 0, L2	I5:	<P5>	Sub k, k, 1
I6: Add k, k, 1	I6:	<P4>	Add k, k, 1
I7: Jmp L3	I7:	<P0>	Add i, i, 1
I8: L2: Sub k, k, 1			
I9: L3: Add i, i, 1			

a) Las instrucciones que se pueden ejecutar en paralelo sin conflicto son:

- **I1** con **I7** **=>** se ejecutan siempre, ya que se asocian al registro predicado P0 = 1.
- **I2** con **I7 =>** I2 no puede ejecutarse e I4 dependen de predicados complementarios entre si e I7 se debe ejecutar siempre.
- **I3** con **I5**, con **I6** y con **I7 =>** I3, I5 e I6 dependen de predicados que no pueden ser simultáneamente verdaderos e I7 se debe ejecutar siempre.

b) Una posible agrupación en paquetes de 3 instrucciones (*bundle*) para la arquitectura del IA64 será:

I1	I7	NOP
I2	I4	NOP
I3	I5	I6

4.2.2. Los DSPs de la familia 6200 de TI hacen la captura de instrucciones en grupos de 256 bits, que contienen 8 instrucciones de 32 bits. Esas 8 instrucciones se agrupan en uno o varios "paquetes de ejecución", o sea, instrucciones que se pueden ejecutar en paralelo. De esta manera, puede ser que las 8 instrucciones se ejecuten en paralelo en las 8 unidades funcionales de las que dispone el procesador (caso más favorable), que se tengan que ejecutar todas en serie (caso más desfavorable), o cualquiera de los casos intermedios. Las ocho unidades funcionales de las que se dispone son:

- Dos unidades de aritmética (suma/resta) y lógica
- Dos unidades de aritmética (suma/resta), desplazamiento y saltos
- Dos unidades de multiplicación
- Dos unidades de load/store

El procesador está segmentado, teniendo 4 etapas de captura, 2 de decodificación y 5 de ejecución (si bien muchas instrucciones utilizan menos de 5 ciclos de ejecución). Para aumentar las prestaciones, el procesador no tiene ninguna lógica de detección de riesgos (ni de datos ni de control), que deben ser resueltos por el compilador.

a) ¿Qué tipo de microprocesador es: VLIW o superescalar? ¿Por qué?

b) Compare el CPI de un programa que tiene sólo sumas y restas con el de otro que tenga sólo multiplicaciones.

c) En un procesador que corre a 250 MHz se necesita ejecutar un algoritmo que debe hacer 10 multiplicaciones y 12 sumas para cada dato. ¿Cuál es la máxima velocidad [datos/segundo] a la que se pueden introducir los datos?

SOLUCIÓN:

a) Es un procesador VLIW porque la planificación es realizada por el compilador y aparece explícitamente en el código que se ejecuta. Otro dato adicional es que no realiza detección de riesgos.

b) Un programa que tenga sólo sumas y restas puede alcanzar un CPI de 0,25 (hay 4 unidades funcionales que pueden hacer estas operaciones, por lo que se pueden emitir 4 instrucciones por ciclo). Sin embargo, como sólo hay 2 unidades para la multiplicación, el mínimo CPI que se puede obtener realizando sólo multiplicaciones es de 0,5, o sea, la mitad de velocidad que en el caso anterior.

c) En cada ciclo de reloj se pueden realizar 4 sumas y 2 multiplicaciones. Por lo tanto, para cada dato se necesitarían 12/4=3 ciclos para las sumas y 10/2=5 ciclos para las multiplicaciones. La limitación viene impuesta por el número de multiplicadores disponibles, por lo que cada dato tardará 5 ciclos en procesarse como muy poco. Si la frecuencia de reloj es de 250 MHz, se pueden procesar 250/5=50 Mdatos/segundo

TECNICAS AVANZADAS EN PARALELISMO

4.2.3. Se dispone de un procesador VLIW con 4 unidades de ejecución. Una unidad para load/store y saltos que necesita 2 ciclos de ejecución, una unidad de enteros (ALU) de 2 ciclos y dos unidades de punto flotante que requieren de 4 ciclos. Todas las unidades de ejecución son segmentadas (en cada ciclo de reloj puede empezar una nueva ejecución). Los ciclos se refieren a la latencia total del de la segmentación (pipeline).
Se utiliza este procesador para ejecutar el siguiente programa que multiplica cada elemento de un vector de 6 elementos por una constante y además calcula la suma de los elementos del vector. Los elementos del vector (A[i]) están en coma flotante de tamaño Word (4 bytes):

```
for (i = 6; i > 0; i--)
{ Suma = Suma + A[i] ;
  A[i] = A[i] * cte; }
```

El código ensamblador (al margen de las inicializaciones) correspondiente es:

```
Loop:   Load R2, 100(R1)     ; R1 es el apuntador al arreglo
        Addf R3, R3, R2      ; R3 lleva la suma
        Mulf R4, R2, R5      ; En R5 la constante
        Store 100(R1), R4    ; guarda el Valor A[i]
        Sub  R1, R1, 4       ; El P. flotante es de 32 bits
        Bnez R1, loop
```

a) ¿Cuántos ciclos necesita para ejecutarse este código si no se aplica ninguna optimización?
b) Utilizando desenrollamiento de bucles, actualización de referencias y renombramiento de registros, en cuántos ciclos se puede ejecutar el código anterior. Suponga que no tiene límite en la cantidad de registros y nómbrelos de la forma Rxx (xx números). Utilice la tabla adjunta
c) Si cada unidad de operación requiere instrucciones de 32 bits, ¿qué tamaño tiene el código de la respuesta a y b sin considerar ningún tipo empaquetado?

SOLUCIÓN:

Ciclo	LD/ST saltos	ALU enteros	Coma Flotante 1	Coma Flotante 2
1	Load R2, 100(R1)	Nop	Nop	Nop
2	Nop	Nop	Nop	Nop
3	Nop	Nop	Addf R3, R3, R2	Mulf R4, R2, R5
4	Nop	Nop	Nop	Nop
5	Nop	Nop	Nop	Nop
6	Nop	Nop	Nop	Nop
7	Store 100(R1), R4	Sub R1, R1, 4	Nop	Nop
8	Nop	Nop	Nop	Nop
9	Bnez R1, loop	Nop	Nop	Nop
10	Nop	Nop	Nop	Nop

a) Tiempo: 10*6 = 60 ciclos

Ciclo	LD/ST saltos	ALU enteros	Coma Flotante 1	Coma Flotante 2
1	Load R2, 100(R1)	Nop	Nop	Nop
2	Load R12, FC(R1)	Nop	Nop	Nop
3	Load R22, F8(R1)	Nop	Nop	Mulf R4, R2, R5
4	Load R32, F4(R1)	Nop	Addf R3, R2, R12	Mulf R14, R12, R5
5	Load R42, F0(R1)	Nop	Nop	Mulf R24, R22, R5
6	Load R52, EC(R1)	Nop	Addf R13, R22, R32	Mulf R34, R32, R5
7	Store 100(R1), R4	Nop	Nop	Mulf R44, R42, R5
8	Store FC(R1), R14	Nop	Addf R23, R42, R52	Mulf R54, R52, R5
9	Store F8(R1), R24	Nop	Nop	Nop
10	Store F4(R1), R34	Nop	Addf R3, R3, R13	Nop
11	Store F0(R1), R44	Nop	Nop	Nop
12	Store EC(R1), R54	Nop	Nop	Nop
13	Nop	Nop	Nop	Nop
14	Nop	Nop	Addf R3, R3, R23	Nop
15	Nop	Nop	Nop	Nop
16	Nop	Nop	Nop	Nop
17	Nop	Nop	Nop	Nop

b) Tiempo: 17 ciclos

c) Código a: 10 instrucciones x 4 codop x 32 bits / (8 bits/Byte) = 10*16 = 160 bytes.
Código b: 17 instrucciones x 4 codop x 32 bits / (8 bits/Byte) = 17*16 = 272 bytes.

TECNICAS AVANZADAS EN PARALELISMO

4.2.4. Se dispone de un procesador VLIW con 5 unidades de ejecución. Una de load/store que necesita 4 ciclos de ejecución, una unidad de enteros (ALU) y saltos de 2 ciclos y tres unidades de punto flotante que requiere de 3 ciclos. Todas las unidades de ejecución son segmentadas (en cada ciclo de reloj puede empezar una nueva ejecución). Los ciclos se refieren a la latencia total del pipeline (segmentación). Dentro del pipeline no existe adelantamiento de datos.
Se utiliza este procesador VLIW para computar el programa que multiplica y suma elemento a elemento los 500 valores de dos vectores almacenados en memoria (arreglos A y B). Además calcula la suma del vector A y B por separado. Los elementos del arreglo (A[i], B[i], C[i] y D[i]) son en coma flotante de tamaño Word (4 bytes).
El código C y ensamblador (al margen de las inicializaciones) correspondiente es:

Código C		**Ensamblador**	
		; suponer los reg	*ya inicializados*
	Loop:	Load R2, 1000(R1)	*; R2 <- mem(1000+R1)*
		Load R4, 2000(R1)	*; R4 <- mem(2000+R1)*
for (i=500; i>0; i--)		Mulf R5, R2, R4	*; R5 <- A[i]*B[i]*
{ C[i] = A[i]*B[i];		Addf R6, R2, R4	*; R6 <- A[i]+B[i]*
D[i] = A[i]+B[i];		Addf R7, R7, R2	*; R7 suma los A[i]*
Sum1 += A[i];		Addf R8, R8, R4	*; R8 suma los B[i]*
Sum2 += B[i];		Store 3000(R1), R5	*; mem(3000+R1) <- R5*
};		Store 4000(R1), R6	*; mem(4000+R1) <- R6*
		Sub R1, R1, 4	*; R1 <- R1 - 4*
		Bnez R1, loop	*; salto si R1 > 0*
		Stop	

a) ¿Cuantos ciclos necesita para ejecutarse este código si no se aplica ninguna optimización?
b) Utilizando desenrollamiento de bucles, actualización de referencias y renombramiento de registros, se quiere mejorar el tiempo de ejecución. El desenrollamiento completo del programa ocuparía demasiada memoria. Se propone desenrollar, replicando el bucle central cinco veces. Suponga que no tiene límite en la cantidad de registros y nómbrelos de la forma Rxx (xx números).

SOLUCIÓN:

Ciclo	LD/ST	ALU enteros y saltos	Coma Flot 1	Coma Flot 2	Coma Flot 3
1	Load R2, 1000(R1)	Nop	Nop	Nop	Nop
2	Load R4, 2000(R1)	Nop	Nop	Nop	Nop
3	Nop	Nop	Nop	Nop	Nop
4	Nop	Nop	Nop	Nop	Nop
5	Nop	Nop	Nop	Nop	Addf R7, R7, R2
6	Nop	Nop	Mulf R5, R2, R4	Addf R6, R2, R4	Addf R8, R8, R4
7	Nop	Nop	Nop	Nop	Nop
8	Nop	Nop	Nop	Nop	Nop
9	Store 3000(R1), R5	Sub R1, R1, 4	Nop	Nop	Nop
10	Store 4000(R1), R6	Nop	Nop	Nop	Nop
11	Nop	Bnez R1, loop	Nop	Nop	Nop
12	Nop	Nop	Nop	Nop	Nop
13	Stop	Stop	Stop	Stop	Stop
14					

Tiempo: 500*12 + 1 = 6001 ciclos

TECNICAS AVANZADAS EN PARALELISMO

Ciclo	LD/ST	ALU enteros y saltos	Coma Flot 1	Coma Flot 2	Coma Flot 3
1	Load R2, 1000(R1)	Nop	Nop	Nop	Nop
2	Load R4, 2000(R1)	Nop	Nop	Nop	Nop
3	Load R12, 0FFC(R1)	Nop	Nop	Nop	Nop
4	Load R14, 1FFC(R1)	Nop	Nop	Nop	Nop
5	Load R22, 0FF8(R1)	Nop	Nop	Nop	Addf R7, R7, R2
6	Load R24, 1FF8(R1)	Nop	Mulf R5, R2, R4	Addf R6, R2, R4	Addf R8, R8, R4
7	Load R32, 0FF4(R1)	Nop	Nop	Nop	Nop
8	Load R34, 1FF4(R1)	Nop	Mulf R15,R14,R12	Addf R16,R14,R12	Addf R7, R7, R12
9	Load R42, 0FF0(R1)	Nop	Nop	Nop	Addf R8, R8, R14
10	Load R44, 1FF0(R1)	Nop	Mulf R25,R24,R22	Addf R26,R24,R22	Nop
11	Store 3000(R1), R5	Nop	Nop	Nop	Addf R17,R22,R32
12	Store 4000(R1), R6	Nop	Mulf R35,R34,R32	Addf R36,R34,R32	Addf R18,R24,R34
13	Store 2FFC(R1), R15	Nop	Nop	Nop	Addf R7,R7,R42
14	Store 3FFC(R1), R16	Nop	Mulf R45,R44,R42	Addf R36,R34,R32	Addf R8,R8,R44
15	Store 2FF8(R1), R25	Nop	Nop	Nop	Nop
16	Store 3FF8(R1), R26	Nop	Nop	Nop	Addf R7, R7, R17
17	Store 2FF4(R1), R35	Nop	Nop	Nop	Addf R8, R8, R18
18	Store 3FF4(R1), R36	Nop	Nop	Nop	Nop
19	Store 2FF0(R1), R45	Sub R1, R1, 14	Nop	Nop	Nop
20	Store 3FF0(R1), R46	Nop	Nop	Nop	Nop
21	Nop	Bnez R1, loop	Nop	Nop	Nop
22	Nop	Nop	Nop	Nop	Nop
23	Stop	Stop	Stop	Stop	Stop
24					

Tiempo: 22*100 + 1 = 2201 ciclos

TECNICAS AVANZADAS EN PARALELISMO

4.2.5. La familia de microprocesadores DSP C64x de Texas Instruments (arquitectura VLIW) está segmentada en 3 etapas (fetch, decode, execute) por la que pasan todas las instrucciones. La etapa de captura (fetch) a su vez está segmentada en 4 etapas:

PG: Program address generate (generación de dirección)
PS: Program address send (envío de la dirección a la caché de instrucciones)
PW: Program access ready wait (estado de espera)
PR: Program fetch packet receive (recepción de la instrucción)

La etapa de decodificación a su vez tiene dos etapas de segmentación

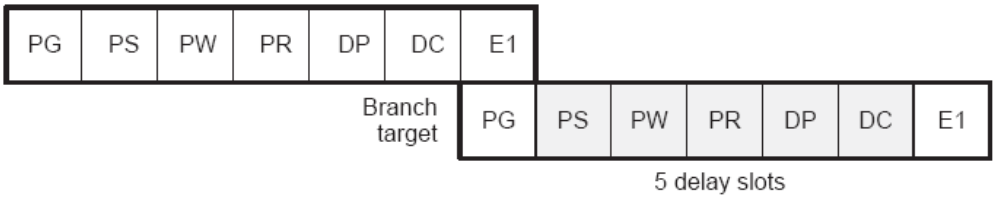
DP: Instruction dispatch (donde se envía a la unidad funcional que corresponda)
DC: Instruction decode (decodificación de la instrucción)

Finalmente la ejecución está segmentada en 5 etapas (**E1, E2, E3, E4 y E5**) y se necesitan diferente cantidad de ciclos de ejecución dependiendo de la instrucción. Todas las instrucciones capturan sus operandos en la etapa E1. Las instrucciones no pueden capturar un registro en el mismo ciclo que otra lo escribe.
Posee dos rutas de datos con 4 unidades funcionales cada una. Las unidades funcionales se llaman:
(L => Aritmética y lógica; S => Aritmética y saltos; M => Multiplicación y D =>Load/store). En definitiva existen ocho unidades funcionales. Dos unidades de aritmética (suma/resta) y lógica (L1 y L2). Dos unidades de aritmética (suma/resta), desplazamiento y saltos (S1 y S2). Dos unidades de multiplicación (M1 y M2). Dos unidades de load/store (D1 y D2)
Además posee dos bancos de registros internos de 32 bits (total 64 registros). El procesador captura 8 instrucciones de 32 bits por ciclo de instrucción. Para aumentar las prestaciones, el procesador no tiene ninguna lógica de detección de riesgos (ni de datos ni de control), los que deben ser resueltos por el compilador. En la figura se muestra un esquema de las etapas de segmentación.

← Fetch →				← Decode →		← Execute →				
PG	PS	PW	PR	DP	DC	E1	E2	E3	E4	E5

Respecto a la ejecución de algunas instrucciones:

- Los saltos se resuelven en E1 y modifican la dirección de programa en la (etapa PG) generando 5 ciclos perdidos.



- Las instrucciones de Load requieren las 5 etapas de ejecución escribiendo el resultado en la última etapa (E5)
- Las instrucciones de store escriben en la caché de datos en E3.
- Las instrucciones de suma calculan y escriben registros en E1
- Las de multiplicación escriben los registros en E2.

a. Determinar el CPI de un programa que tiene sólo sumas y restas, con otro que tenga sólo multiplicaciones y con un tercero que solo posea instrucciones de load y store. Sin considerar riesgos.

Si solo sumas, puede ejecutar 4 sumas por ciclo => CPI = 1/4 = 0,25

Si solo multiplicaciones, puede ejecutar 2 multiplicaciones por ciclo => CPI = 1/2 = 0,5

Si solo load/store, puede ejecutar 2 load/store por ciclo => CPI = 1/2 = 0,5

b. Cuánto tiempo tardaría el código C que se adjunta si **no hay límites para el tamaño del código y se pretende la máxima velocidad posible.** El procesador funciona a una velocidad de 200 MHz. Puede obviar los ciclos de llenado del pipeline para el cálculo. Justifique su respuesta usando el sitio adecuado y/o la tabla que representa las 8 unidades funcionales. Utilice los nemónicos LD, ST, ML, AD para las instrucciones de load, store, multiplicación y suma. Para las instrucciones de load y store (LD, ST) suponga que están definidas constantes A,B,C,D que apuntan a la primera posición de cada vector. Así LD r1, [A] es cargar A(1) y LD r2, [C+4] es cargar C(2). También existen las constantes X, Z que apuntan a las direcciones de X y Z respectivamente.

TECNICAS AVANZADAS EN PARALELISMO

```
Z= 0; X=0
for (int i=1; i  < 4; i ++) {
    Z += A[i] * B[i] * C[i] * D[i];
    X += A[i] + B[i] + C[i] + D[i]; }
```

Justifique su respuesta, puede utilizar la tabla que representa las 8 unidades para ello:

D1	D2	M1	M2	L1	L2	S1	S2
Ld r1, [A]	Ld r2, [B]	-	-	-	-	-	-
Ld r3, [C]	Ld r4, [D]	-	-	-	-	-	-
Ld r5, [A+4]	Ld r6, [B+4]	-	-	-	-	-	-
Ld r7, [C+4]	Ld r8, [D+4]	-	-	-	-	-	-
Ld r9, [A+8]	Ld r10,[B+8]	-	-	-	-	-	-
Ld r11, [C+8]	Ld r12, [D+8]	ML r15, r1, r2	-	Ad r21, r1, r2	-	-	-
-	-	ML r16, r3, r4	-	Ad r22, r3, r4	-	-	-
-	-	ML r17, r5, r6	-	Ad r23, r5, r6	Ad r2,r21,r22	-	
-	-	ML r18, r7, r8	ML r1,r15,r16	Ad r24, r7, r8	-	-	
-	-	ML r19, r9, r10	-	Ad r25,r9,r10	Ad r4,r23,r24	-	-
-	-	ML r20, r11,r12	ML r3,r17,r18	Ad r26,r11,r12			Ad r30,r2,r4
-	-	-	-	-	Ad r6,r25,r26	-	-
-	-	-	ML r5,r19,r20	-	-	Ad r31,r1,r3	Ad r30,r30,r6
-	St r30, [X]	-	-	-	-	-	-
-	-	-	-	-	-	Ad r31,r31,r5	-
St r31, [Z]	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

Número de ciclos y tiempo de ejecución para el código:

Se supone que X y Z no tienen valor previo, y que r30 y r31 empiezan en 0.

Desenrollando completamente las 3 iteraciones se tiene:

6 etapas de pipeline (4 de fetch + 2 de decode) para comenzar a ejecutar (es válido tenerlo en cuenta o no).

Además hay 5 ciclos del primer load y 3 productos de 2 ciclos de retardo por iteración y 7 ciclos finales hasta terminar.

Total= 5 ciclos del load + 3*2 ciclos mult + 1 pipe mult + 2 último mult + 1 sum + 3 store = 18 ciclos

Tiempo = 1/200 MHz * 18 ciclos = 5 * 18 = 90 ns

Si se tiene en cuenta el llenado del pipeline: 24 ciclos y 120 ns

c. Número de ciclos y tiempo necesarios para la ejecución del código anterior pero para cien iteraciones en vez de tres en el bucle for. (*for (int i=1; i < 101; i ++)*)

Desenrollando completamente las 100 iteraciones y utilizando el razonamiento anterior, se tiene:

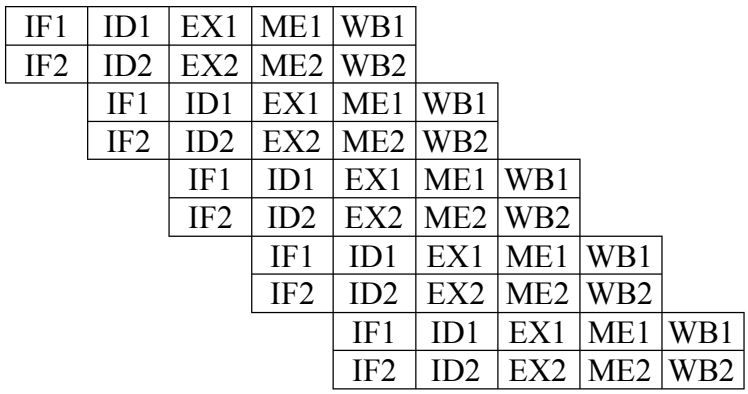
Total= 5 ciclos del load + 100*2 ciclos mult + 1 pipe mult + 2 último mult + 1 sum + 3 store = 212 ciclos

Tiempo = 1/200 MHz * 212 ciclos = 5 * 212 = 1060 ns = 1,06 microsegundos

TECNICAS AVANZADAS EN PARALELISMO

4.2.6. Se ha hecho una versión superescalar del procesador de teoría y prácticas. En esta versión simplificada se utiliza inicialización, ejecución y finalización en orden. Cada cauce superescalar es un sistema segmentado en las mismas 5 etapas que el procesador de teoría (IF, ID, EX, MEM y WB).

El procesador captura (fetch) dos instrucciones en cada ciclo de reloj y en las etapas ID si detecta riesgos que no puede resolver, detiene el cauce que no puede ejecutar hasta que se solucione el riesgo. Para evitar riegos estructurales y simplificar el diseño, hasta que las dos instrucciones capturadas en el mismo ciclo no salen de la decodificación (ID1 e ID2) se detienen nuevas capturas.



La arquitectura posee además adelantamiento de datos "internal forwarding" en cada ruta de datos y permite adelantar entre los dos cauces (rutas de datos). Es decir a EX1 y a EX2 se pueden adelantar datos desde ME1, WB1, ME2 y WB2. También a las etapas ME1 y ME2 se puede adelantar el dato a escribir en el caso de las instrucciones de store. Los saltos se predicen no efectivos y la condición de salto se resuelve en EX.

La caché de instrucciones permite la lectura de dos instrucciones en cada ciclo de reloj, en tanto la caché de datos permite leer y/o escribir hasta dos palabras por ciclo de reloj. El banco de registros evidentemente permite leer cuatro registros y escribir otros dos en cada ciclo. Suponga que se ejecuta el siguiente código.

```
                          Inic: load r1, r2(100)  ; carga A[i] en r1
for (i = 4; i >0 ; i--){        add r6, r6, r1    ; suma a[i] en R6
   sum += A[i];                 add r7, r1, r10   ; A[i] + cte. R10 es la cte
   A[i] = A[i] + cte;           store r7, r2(100) ; guarda A[i]
  }                             sub r2, r2, 4     ; i-- (datos de 4 bytes)
                                bnq r2, r0, inic  ; Salto inicio
```

El siguiente diagrama de tiempos muestra la ejecución del código anterior. Las etapas que aparecen entre paréntesis indican que se ha detenido la instrucción en esa etapa. Las flechas indican los adelantamientos de datos producidos con la semántica de qué etapa produce un dato y quién la utiliza.

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13
load r1, r2(100)	IF1	ID1	EX1	ME1	WR1								
add r6, r6, r1	IF2	ID2	(ID2)	(ID2)	EX2	ME2	WR2						
add r7, r1, r10		IF1	(IF1)	(IF1)	ID1	EX1	ME1	WR1					
store r7, r2(100)		IF2	(IF2)	(IF2)	ID2	EX2	ME2	WR2					
sub r2, r2, 4					IF1	ID1	EX1	ME1	WR1				
bnq r2, r0, inic					IF2	ID2	(ID2)	EX2	ME2	WR2			
Next						IF1	(IF1)	ID1	--				
Next+1						IF2	(IF2)	ID2	--				
Next+2								IF1	--				
Next+3								IF2	--				
load r1, r2(100)									IF1	ID1	EX1	ME1	WR1
add r6, r6, r1									IF2	ID2	(ID2)	(ID2)	EX2

a. ¿Cuántos ciclos ha de tardar la ejecución del programa? Justifique brevemente
8.3 + 10 = 8*4 + 2 = 34 ciclos.

Las 3 primeras iteraciones necesitan 8 ciclos hasta comenzar la nueva iteración. La última iteración necesita dos ciclos más hasta finalizar.

b. Indique los adelantamientos que se han producido en la ejecución del código anterior. Se muestra el primer adelantamiento

TECNICAS AVANZADAS EN PARALELISMO

A EX2 en T5 desde WR1
A ME2 en T7 desde ME1 (en el store)
A EX2 en T8 desde ME1 (en beq)

El 1er adelantamiento se repite en T13, T21 y T29; el 2do en T15, T23 y T31; el 3ro en T16, T24 y T32

c. Un compilador avanzado permite aplicar las técnicas de desenrollamiento de bucles, renombramiento de registros y actualización de referencias. Cuántos ciclos podría tardar la ejecución del código desenrollado. Suponga por simplicidad que no existe límite en la cantidad de registros pudiendo utilizar cualquier registro de la forma rxx. Escriba el código resultante en el recuadro y complete el diagrama de ejecución.

Código ensamblador tras la optimización:

```
I1:  load r1, r2(100) ;
I2:  load r11, r2(FC) ;
I3:  load r21, r2(F8) ;
I4:  load r31, r2(F4) ;
I5:  add r16, r11, r1  ; A[3] + A[2]
I6:  add r7, r1, r10 ; A[3] + cte
I7:  add r26, r21, r31  ; A[1] + A[0]
I8:  add r17, r11, r10 ; A[2] + cte
I9:   add r36, r16, r26 ; A[3] + A[2] + A[1] + A[0]
I10:  add r27, r21, r10; A[1] + cte
I11:  add r6, r6, r36 ; r6 + A[3] + A[2] + A[1] + A[0]
I12:  add r37, r31, r10; A[0] + cte
I13:  store r7, r2(100)
I14:  store r17, r2(FC)
I15:  store r27, r2(F8)
I16:  store r37, r2(F4)
```

Eventualmente podría agregar “sub r2, r2, 16”

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
I1:load r1, r2(100)	IF1	ID1	EX1	ME1	WR1										
I2:load r11, r2(FC)	IF2	ID2	EX2	ME2	WR2										
I3:load r21, r2(F8)		IF1	ID1	EX1	ME1	WR1									
I4:load r31, r2(F4)		IF2	ID2	EX2	ME2	WR2									
I5:add r16, r11, r1			IF1	ID1	EX1[1]	ME1	WR1								
I6:add r7, r1, r10			IF2	ID2	EX2[2]	ME2	WR2								
I7:add r26, r21, r31				IF1	ID1	EX1[3]	ME1	WR1							
I8:add r17, r11, r10				IF2	ID2	EX2[7]	ME2	WR2							
I9:add r36, r26, r16					IF1	ID1	EX1[4]	ME1	WR1						
I10:add r27, r21, r10					IF2	ID2	EX2	ME2	WR2						
I11:add r6, r6, r36						IF1	ID1	EX1[5]	ME1	WR1					
I12:add r37, r31, r10						IF2	ID2	EX2	ME2	WR2					
I13:store R7, r2(100)							IF1	ID1	EX1	ME1	WR1				
I14:store r17, r2(FC)							IF2	ID2	EX2	ME2	WR2				
I15:store r27, r2(F8)								IF1	ID1	EX1	ME1	WR1			
I16:store r37, r2(F4)								IF2	ID2	EX2[6]	ME2	WR2			

La ejecución tarda 12 ciclos. Una aceleración de 34/12 = 2,83

d. Utilizando la sintaxis del punto b indique los adelantamientos de datos producidos:
1. a EX1 en T5 desde WR1 y WR2
2. a EX2 en T5 desde WR1
3. a EX1 en T6 desde WR1 y WR2
4. a EX1 en T7 desde ME1
5. a EX1 en T8 desde ME1
6. Este adelantamiento es algo más sutil. El dato producido por EX2 en T8 se escribe por WR2 en T10. Sin embargo la instrucción de store lo necesita en ME2 en el ciclo 11. Pero cuando estuvo en ID2 (T9) aun no estaba en los registros. La solución de esto es adelantarlo a la etapa EX2 en T10 desde WR2 y que avance en el ciclo 11 tanto la dirección calculada en EX2 como el dato a guardar que se ha adelantado.

i. No es necesario adelantar. En T5 se puede coger el dato del banco de registros ya que se escribe y lee en el mismo ciclo.

TECNICAS AVANZADAS EN PARALELISMO

4.2.7. Se ha desarrollado un MIPS superescalar de grado 4 con planificación estática para aplicaciones DSP (Digital Signal Processing – Procesado digital de la señal). El procesador está segmentado en las mismas 5 etapas del MIPS original estudiado en clase (IF, ID, EX, MEM, WB). El procesador dispone de 2 unidades para ejecutar operaciones aritmético-lógicas (U1 y U2) que se ejecutan en la etapa EX; una unidad para load/store (U3) (calcula dirección en EX y lee/escribe en MEM) y una unidad para multiplicaciones (U4). La unidad de multiplicación esta segmentada en 2 etapas, usando las etapas EX y MEM para su ejecución. Para los saltos condicionales e incondicionales es necesario utilizar la U2. El salto ha sido adelantado a la etapa ID de modo que solo pierde un ciclo de reloj. El procesador dispone de todos los adelantamientos posibles y que tienen sentido.

Notas:

El procesador captura 4 instrucciones por ciclo de reloj (32*4 = 128 bits) y no se realiza ningún tipo de control adicional (la planificación es estática realizada por el compilador).

En la tabla se muestran de 4 en 4 instrucciones (00-03, 04-07, etc.), las instrucciones NOP correspondientes se notan con "-" por simplicidad.

Los datos son siempre de 32 bits. El resultado de Sum1 y Sum2 queda en los registros r6 y r8 respectivamente. El registro r30 posee la constante cte a multiplicar y el registro r31 el número 40000 (decimal) (4*10000). El resto de los registros están inicialmente a cero.

Considerando el siguiente código ensamblador que itera diez mil veces:

```
for (i = 0; i <10000; i++)     L1: Lw r5, base(r1)        ; Carga elemento A[i]
{ Sum1 += A[i];                    Add r6, r6, r5         ; acumular sum 1
  Sum2 += A[i]*A[i];               Mul r7, r5, r5         ; realiza el cuadrado de A[i]
  A[i] *= cte; }                   Add r8, r8, r7         ; acumula el cuadrado
                                   Mul r9, r5, r30        ; multiplica por la constante (r30 =cte).
                                   Sw r9, base(r1)        ; Almacena resultado
                                   Add r1, r1, 4          ; Apunta a dir de memoria siguiente
                                   Bne r1, r31, L1        ; Salto condicional al comienzo del bucle
```

Un compilador ha realizado la siguiente planificación:

inst	U1	U2	U3	U4
00-03	-	Add r1, r1, 4	Lw r5, base(r1)	-
04-07	-	-	-	-
08-0B	Add r6, r6, r5	-	-	Mul r7, r5, r5
0C-0F	-	-	-	Mul r9, r5, r30
10-13	Add r8, r8, r7	-	-	-
14-17	-	Bne r1, r31, L1	Sw r9, base-4(r1)	-
18-1B				

a. ¿Cuantos ciclos de reloj necesita para ejecutarse ese código en el procesador propuesto?

10000*7 + 3 = 70003;
(N-1 iterac)* (Nro Ciclos/iterac + ciclo perdido salto)+última iterac.=9999*(6+1)+10= 70003

*Se capturan 6 grupos de instrucciones (6 ciclos) y el salto pierde un ciclo (7 ciclos) y vuelve a comenzar el bucle. Esto se repite 9999 veces. La última vez no salta y esperamos vaciar el pipeline (7+3 ciclos). No es necesario considerar el vaciado del pipeline.

Para justificar su repuesta utilice la tabla que muestra las ejecuciones del pipeline

	1	2	3	4	5	6	7	8	9	10	11	12
00-03	IF	DE	EX	MEM	WB							
04-07		IF	DE	EX	MEM	WB						
08-0B			IF	DE	EX	MEM	WB					
0C-0F				IF	DE	EX	MEM	WB				
10-13					IF	DE	EX	MEM	WB			
14-17						IF	DE	EX	MEM	WB		
18-1B							IF	-	-	-	-	-
00-03								IF	DE	EX	MEM	WB
04-07									IF	DE	EX	MEM

b. Se necesita aumentar el rendimiento y se permite realizar optimizaciones al compilador que incluyen: desenrollamiento de bucles, renombramiento de registros, actualización de referencias y reordenamiento de código.

Por cuestiones de tamaño de código es requisito que el código no supere las 60 instrucciones (desde 00h hasta 3Bh). Desenrolle el bucle 4 veces (es decir que se ejecute el bucle 2500 veces).

	U1	U2	U3	U4
00-03	-	Add r1, r1, 16d	Lw r5, base(r1)	-
04-07	-	-	Lw r11, base-12(r1)	-
08-0B	Add r6, r6, r5	-	Lw r12, base-8(r1)	Mul r7, r5, r5
0C-0F	Add r6, r6, r11	-	Lw r13, base-4(r1)	Mul r9, r5, r30
10-13	Add r6, r6, r12	Add r8, r8, r7	-	Mul r17, r11, r11
14-17	Add r6, r6, r13	-	Sw r9, base-16(r1)	Mul r19, r11, r30
18-1B	-	Add r8, r8, r17	-	Mul r18, r12, r12
1C-1F	-	-	Sw r19, base-12(r1)	Mul r20, r12, r30
20-23	-	Add r8, r8, r18	-	Mul r21, r13, r13
24-27	-	-	Sw r20, base-8(r1)	Mul r22, r13, r30
28-2B	-	Add r8, r8, r21	-	-
2C-2F	-	Bne r1, r31, L1	Sw r22, base-4(r1)	-
30-33				
34-37				
38-3B				
3C-3F				
40-43				

¿Cuántos ciclos tardará la ejecución de este código?
2500*13 + 3 = 32503; (desenrollando x 4)
2500*12 + 3 = 30003; (desenrollando x 4 adelanto del Bne y adelanto a MEM en SW)
2500*11 + 3 = 27503; (desenrollando x 4 adelanto y suponer Bne funciona con “delay slot”)

Puede utilizar esta segunda tabla en caso de equivocarse en la anterior.

	U1	U2	U3	U4
00-03	-	Add r1, r1, 16d	Lw r5, base(r1)	-
04-07	-	-	Lw r11, base-12(r1)	-
08-0B	Add r6, r6, r5	-	Lw r12, base-8(r1)	Mul r7, r5, r5
0C-0F	Add r6, r6, r11	-	Lw r13, base-4(r1)	Mul r9, r5, r30
10-13	Add r6, r6, r12	Add r8, r8, r7	Sw r9, base-16(r1)	Mul r17, r11, r11
14-17	Add r6, r6, r13	-	-	Mul r19, r11, r30
18-1B	-	Add r8, r8, r17	Sw r19, base-12(r1)	Mul r18, r12, r12
1C-1F	-	-	-	Mul r20, r12, r30
20-23	-	Add r8, r8, r18	Sw r20, base-8(r1)	Mul r21, r13, r13
24-27	-	-	-	Mul r22, r13, r30
28-2B	Add r8, r8, r21	Bne r1, r31, L1	Sw r22, base-4(r1)	-
2C-2F	-	-	-	-
30-33				
34-37				
38-3B				
3C-3F				
40-43				

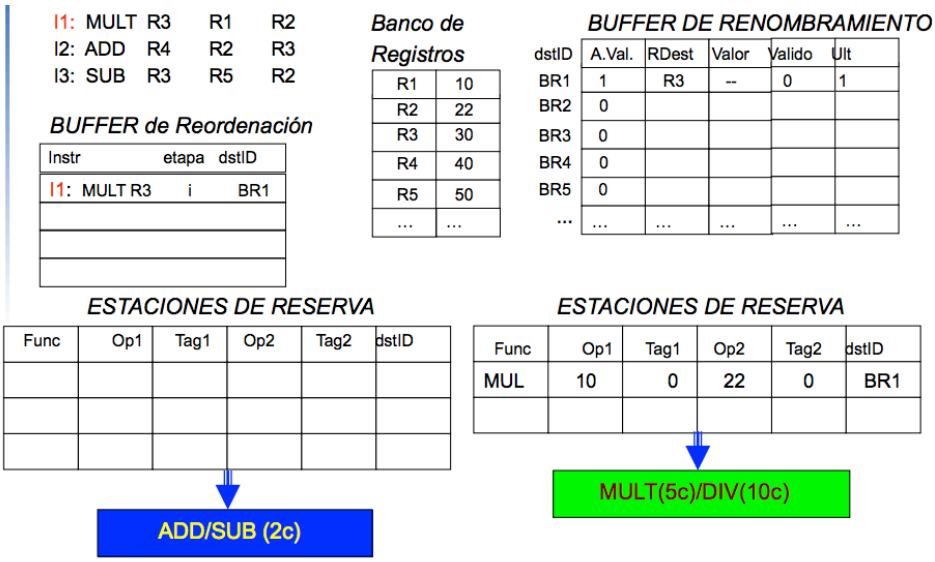
*Posibles Cambios correctos:
Suponer que el adelanto al Store se produce a la etapa MEM. Eso adelanta los SW un ciclo.
Suponer que la ausencia de control lleva a que el BNE no vacía la siguiente instrucción en caso de salto efectivo funcionando como un “salto con delay slot”.

4.2.8. La siguiente figura representa el estado de diferentes componentes de un procesador superescalar después de la emisión de la instrucción I1 de la siguiente secuencia de código

I1 : MULT R3 R1 R2 ; R3 = R1 * R2
I2 : ADD R4 R2 R3 ; R4 = R2 + R3
I3 : SUB R3 R5 R2 ; R3 = R5 - R2

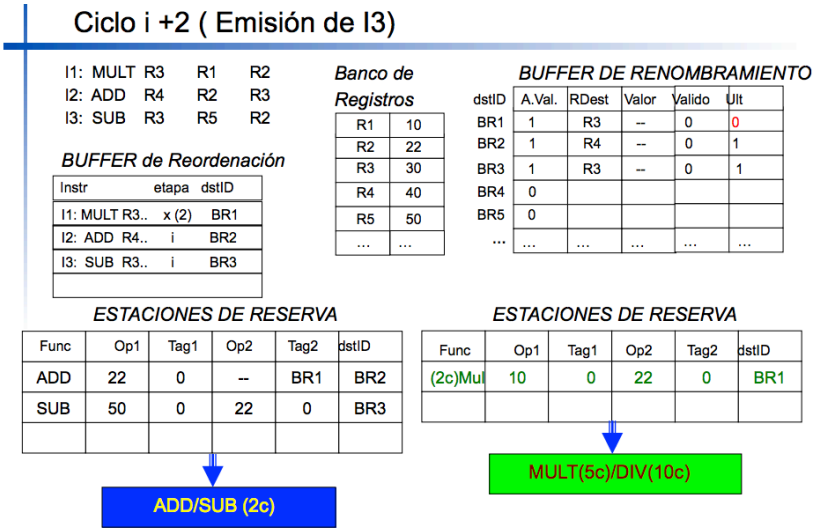
Teniendo en cuenta que el procesador tiene las siguientes Unidades Funcionales (UF) con ejecución segmentada:

- 3 UF para operaciones ADD/SUB que tardan 2 ciclos en ejecución.
- 2 UF para realizar operaciones MULT/DIV que tardan 5 ciclos en operaciones MULT y 10 ciclos en operaciones DIV



Se pide: Completar sobre la figura el estado de los componentes una vez que se haya emitido la instrucción I3.

Solución:



TECNICAS AVANZADAS EN PARALELISMO

4.2.9. - Se quiere ejecutar el siguiente programa en una versión segmentada del procesador MIPS estudiado en clase (5 etapas: IF, ID, EX, MEM y WB. Arquitectura Harvard):

```
Instrucción                          Comentario
I1    inic: load r1, 100(r2)         ; load A[i] into r1
I2          add r6, r6, r1           ; add A[i] to R6
I3          add r7, r1, r10          ; A[i] + cte. R10 contains cte
I4          store r7, 100(r4)        ; store B[j]
I5          sub r2, r2, 4            ; i-- (data size is 4 bytes)
I6          sub r4, r4, 4            ; j-- (data size is 4 bytes)
I7          bnq r2, r0, inic         ; Jump to inic
```

a) Identifique todos los riesgos del programa.

b) El procesador no dispone de unidad de predicción de saltos ni de unidad de adelantamiento, pero permite leer y escribir del mismo registro en el mismo ciclo. Completa el diagrama de ejecución. ¿Cuántos ciclos de reloj son necesarios para la ejecución de una iteración del bucle?

Instrucción / Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
I1: load r1, 100(r2)																
I2: add r6, r6, r1																
I3: add r7, r1, r10																
I4: store r7, 100(r4)																
I5: sub r2, r2, 4																
I6: sub r4, r4, 4																
I7: bnq r2, r0, inic																

c) Al diseño anterior se le añade adelantamiento de datos (el mejor posible). Completa el diagrama de ejecución. Indica los adelantamientos mediante flechas cuyo origen es la etapa que genera el dato, y el destino, la etapa que lo necesita. ¿Cuántos ciclos de reloj son necesarios para la ejecución de una iteración del bucle?

Instrucción / Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
I1: load r1, 100(r2)																
I2: add r6, r6, r1																
I3: add r7, r1, r10																
I4: store r7, 100(r4)																
I5: sub r2, r2, 4																
I6: sub r4, r4, 4																
I7: bnq r2, r0, inic																

d) Se desarrolla una versión superescalar de grado 2 del procesador MIPS anterior con emisión y finalización fuera de orden. Cada cauce superescalar es un sistema segmentado con las mismas características que el diseño anterior, permitiendo adelantamiento de datos en cada ruta de datos y entre los dos cauces. Ambos cauces comparten las mismas unidades funcionales (UF) a utilizar durante la etapa de ejecución y se dispone de una unidad funcional load/store/bcc y una unidad funcional ALU para enteros pero se dispone de estaciones de reserva suficientes en cada UF para permitir emisión fuera de orden. En la etapa de ejecución cada unidad funcional utiliza 1 ciclo de reloj. Completa el diagrama de ejecución. Indica los adelantamientos mediante flechas cuyo origen es la etapa que genera el dato, y el destino, la etapa que lo necesita. ¿Cuántos ciclos de reloj son necesarios para la ejecución de una iteración del bucle?

Instrucción / Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
I1: load r1, 100(r2)																
I2: add r6, r6, r1																
I3: add r7, r1, r10																
I4: store r7, 100(r4)																
I5: sub r2, r2, 4																
I6: sub r4, r4, 4																
I7: bnq r2, r0, inic																

e) La arquitectura anterior hace uso de planificación dinámica, por lo que se dispone en la ruta de datos de un buffer de renombramiento y un buffer de reordenación, además de las estaciones de reserva y unidades funcionales ya indicadas. Muestre el contenido de los siguientes elementos, justo cuando finaliza la ejecución de la primera instrucción (use cualquier valor para el contenido de la memoria).

Banco de registros	
R1	*0*
R2	*16*
R4	*16*
R6	*0*
R7	*0*
R10	*5*

Buffer de reordenación		
Instrucción	*Etapa*	*dstID*

Buffer de renombramiento					
dstID	*A.Val.*	*RDest*	*Valor*	*Válido*	*Ult.*
BR1					
BR2					
BR3					
BR4					
BR5					
BR6					

Estación de reserva LD/ST/Bcc					
Func	*Op1*	*Tag1*	*Op2*	*Tag2*	*dstID*

Estación de reserva ALU enteros					
Func	*Op1*	*Tag1*	*Op2*	*Tag2*	*dstID*

f) A la arquitectura anterior se le incorpora una unidad de predicción de saltos que consta de un predictor correlacionado con información de un sólo salto anterior (BHR de 1 bit) y un BHT de 4 entradas (predictor (1,1)). Si el valor de R2 es 16, analice el rendimiento del predictor tras la ejecución del programa. Inicialmente la predicción es no-efectivo y todos los valores de BHT y BHR son 0 (NE). Muestre los contenidos del BHT y BHR en cada iteración del bucle. La instrucción I1 se encuentra en la dirección 0xA000.

g) Un compilador avanzado para la arquitectura anterior permite aplicar las técnicas de desenrollamiento de bucles, renombramiento de registros y actualización de referencias. Escriba el código resultante de aplicar las técnicas anteriores en el recuadro si se sabe que el bucle se ejecuta 4 veces. Suponga por simplicidad que no existe límite en la cantidad de registros pudiendo utilizar cualquier registro de la forma rxx. ¿Cuántos ciclos tarda la ejecución del código desenrollado? Complete la tabla para justificar el número de ciclos. Sólo es necesario indicar en que ciclo se ejecuta cada instrucción.

Código ensamblador tras la optimización:

Ciclo	LD/ST/Bcc	ALU enteros
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		

SOLUCION 4.2.9.- Se quiere ejecutar el siguiente programa en una versión segmentada del procesador MIPS estudiado en clase (5 etapas: IF, ID, EX, MEM y WB. Arquitectura Harvard):

```
Instrucción                          Comentario
I1    inic: load r1, 100(r2)         ; load A[i] into r1
I2          add r6, r6, r1           ; add A[i] to R6
I3          add r7, r1, r10          ; A[i] + cte. R10 contains cte
I4          store r7, 100(r4)        ; store B[j]
I5          sub r2, r2, 4            ; i-- (data size is 4 bytes)
I6          sub r4, r4, 4            ; j-- (data size is 4 bytes)
I7          bnq r2, r0, inic         ; Jump to inic
```

h) Identifique todos los riesgos del programa.

Estructurales: Al disponer de memorias separadas no hay riesgo.
Control: Existe un riesgo de control debido a la instrucción de salto.
Datos:
RAW: I2 con I1 por R1, I3 con I1 por R1, I4 con I3 por R7, I7 con I5 por R2
WAR: I5 con I1 por R2, I6 con I4 por R4
WAW:

i) El procesador no dispone de unidad de predicción de saltos ni de unidad de adelantamiento, pero permite leer y escribir del mismo registro en el mismo ciclo. Completa el diagrama de ejecución. ¿Cuántos ciclos de reloj son necesarios para la ejecución de una iteración del bucle?

Instrucción / Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
I1: load r1, 100(r2)	IF	ID	EX	ME	WB											
I2: add r6, r6, r1		IF	(ID)	(ID)	ID	EX	ME	WB								
I3: add r7, r1, r10			IF	(IF)	(IF)	ID	EX	ME	WB							
I4: store r7, 100(r4)						IF	(ID)	(ID)	ID	EX	ME	WB				
I5: sub r2, r2, 4							IF	(IF)	(IF)	ID	EX	ME	WB			
I6: sub r4, r4, 4										IF	ID	EX	ME	WB		
I7: bnq r2, r0, inic											IF	(ID)	ID	EX	ME	WB

j) Al diseño anterior se le añade adelantamiento de datos (el mejor posible). Completa el diagrama de ejecución. Indica los adelantamientos mediante flechas cuyo origen es la etapa que genera el dato, y el destino, la etapa que lo necesita. ¿Cuántos ciclos de reloj son necesarios para la ejecución de una iteración del bucle?

Instrucción / Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
I1: load r1, 100(r2)	IF	ID	EX	ME	WB											
I2: add r6, r6, r1		IF	(ID)	ID	EX	ME	WB									
I3: add r7, r1, r10			IF	(IF)	ID*	EX	ME	WB								
I4: store r7, 100(r4)					IF	ID	EX	ME	WB							
I5: sub r2, r2, 4						IF	ID	EX	ME	WB						
I6: sub r4, r4, 4							IF	ID	ME	EX	WB					
I7: bnq r2, r0, inic								IF	ID**	EX	ME	WB				

* Se puede leer y escribir en el mismo registro en el mismo ciclo
** Si el comparador está ubicado en la salida del bloque de registros y permitimos adelantamiento de datos a ese comparador desde la salida de la ALU. Sino de ME de I5 a EX de I7. Mismo resultado.

k) Se desarrolla una versión superescalar de grado 2 del procesador MIPS anterior con emisión y finalización fuera de orden. Cada cauce superescalar es un sistema segmentado con las mismas características que el diseño anterior, permitiendo adelantamiento de datos en cada ruta de datos y entre los dos cauces. Ambos cauces comparten las mismas unidades funcionales (UF) a utilizar durante la etapa de ejecución y se dispone de una unidad funcional load/store/bcc y una unidad funcional ALU para enteros pero se dispone de estaciones de reserva suficientes en cada UF para permitir emisión fuera de orden. En la etapa de ejecución cada unidad funcional utiliza 1 ciclo de reloj. Completa el diagrama de ejecución. Indica los adelantamientos mediante flechas cuyo origen es la etapa que genera el dato, y el destino, la etapa que lo necesita. ¿Cuántos ciclos de reloj son necesarios para la ejecución de una iteración del bucle?

Instrucción / Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
I1: load r1, 100(r2)	IF	ID	EX	ME	WB											
I2: add r6, r6, r1	IF	ID^W	ID^W	ID	EX	ME	WB									
I3: add r7, r1, r10		IF	ID^W	ID^W	ID^W	EX	ME	WB								
I4: store r7, 100(r4)		IF	ID^W	ID^W	ID^W	ID^W	EX	ME	WB							
I5: sub r2, r2, 4			IF	ID^W	ID^W	ID^W	EX	ME	WB							
I6: sub r4, r4, 4			IF	ID^W	ID^W	ID^W	ID^W	EX	ME	WB						
I7: bnq r2, r0, inic				IF	ID^W	ID^W	ID^W	EX	ME	WB	WB					

WB en I11 si el adelantamiento es a la etapa ID. Esto supone ID^W en ciclo 8 y retrasar el resto un ciclo.
En la solución propuesta se usa adelantamiento EX → EX dejando el comparador en ALU.
Variantes:

- En el ciclo 3 se puede optar por ejecutar I3 en vez de I2
- En el ciclo 5 se puede optar por ejecutar I5 en vez de I3

l) La arquitectura anterior hace uso de planificación dinámica, por lo que se dispone en la ruta de datos de un buffer de renombramiento y un buffer de reordenación, además de las estaciones de reserva y unidades funcionales ya indicadas. Muestre el contenido de los siguientes elementos, justo cuando finaliza la ejecución de la primera instrucción (use cualquier valor para el contenido de la memoria).

Banco de registros	
R1	*0*
R2	*16*
R4	*16*
R6	*0*
R7	*0*
R10	*5*

Buffer de reordenación		
Instrucción	*Etapa*	*dstID*
I1: LD	*f*	*BR1*
I2: ADD	*i*	*BR2*
I3: ADD	*i*	*BR3*
I4:store	*i*	*memoria*

Buffer de renombramiento					
dstID	*A.Val.*	*RDest*	*Valor*	*Válido*	*Ult.*
BR1	*1*	*R1*	*AA*	*1*	*1*
BR2	*1*	*R6*	*-*	*0*	*1*
BR3	*1*	*R7*	*-*	*0*	*1*
BR4					
BR5					
BR6					

Estación de reserva LD/ST/Bcc					
Func	*Op1*	*Tag1*	*Op2*	*Tag2*	*dstID*
ld	*16*	*0*	*116*	*0*	*BR1*

Estación de reserva ALU enteros					
Func	*Op1*	*Tag1*	*Op2*	*Tag2*	*dstID*
add	*0*	*-*	*-*	*BR1*	*BR2*
add	*-*	*BR1*	*5*	*0*	*BR3*

m) A la arquitectura anterior se le incorpora una unidad de predicción de saltos que consta de un predictor correlacionado con información de un sólo salto anterior (BHR de 1 bit) y un BHT de 4 entradas (predictor

(1,1)). Si el valor de R2 es 16, analice el rendimiento del predictor tras la ejecución del programa. Inicialmente la predicción es no-efectivo y todos los valores de BHT y BHR son 0 (NE). Muestre los contenidos del BHT y BHR en cada iteración del bucle. La instrucción I1 se encuentra en la dirección 0xA000.

La instrucción I7 se encuentra en la dirección 0xA000 + 6*4 = 0xA018, por tanto, ocupa la posición 3 del BHT.
Es la Posición 3 indice 2. porque los 2 bits LSB no se deben considerar al ser siempre 0.
Inicialmente BHT(0,2) = NE, BHT(1,2) = NE, BHR = 0

Primer ciclo: R2 = 16 - 4 = 12 → Salto efectivo → BHT(0,2) = NE → Fallo
BHT(0,2) = E, BHR = 1
Segundo ciclo: R2 = 12 – 4 = 8 → Salto efectivo → BHT(1,2) = NE → Fallo
BHT(1,2) = E, BHR = 1
Tercer ciclo: R2 = 8 - 4 = 4 → Salto efectivo → BHT(1,2) = E → Acierto
BHT(1,2) = E, BHR = 1
Cuarto ciclo: R2 = 4 - 4 = 0 → Salto no efectivo → BHT(1,2) = E → Fallo
BHT(1,2) = NE, BHR = 0
Tras 4 iteraciones hemos tenido 1/4 aciertos = 25% y 3/4 fallos = 75%.

n) Un compilador avanzado para la arquitectura anterior permite aplicar las técnicas de desenrollamiento de bucles, renombramiento de registros y actualización de referencias. Escriba el código resultante de aplicar las técnicas anteriores en el recuadro si se sabe que el bucle se ejecuta 4 veces. Suponga por simplicidad que no existe límite en la cantidad de registros pudiendo utilizar cualquier registro de la forma rxx. ¿Cuántos ciclos tarda la ejecución del código desenrollado?

o)

Código ensamblador tras la optimización:

load r1, 100(r2)	*load r1, 100(r2)*	*load r1, 100(r2)*
add r6, r6, r1	*load r11, 96(r2)*	*load r11, 96(r2)*
add r7, r1, r10	*load r21, 92(r2)*	*load r21, 92(r2)*
store r7, 100(r4)	*load r31, 88(r2)*	*load r31, 88(r2)*
load r11, 96(r2)	*add r6, r6, r1*	*add r7, r1, r10*
add r6, r6, r11	*add r7, r1, r10*	*add r17, r11, r10*
add r17, r11, r10	*add r6, r6, r11*	*add r27, r21, r10*
store r17, 96(r4)	*add r17, r11, r10*	*add r37, r31, r10*
load r21, 92(r2)	*add r6, r6, r21*	*add r1, r1, r11*
add r6, r6, r21	*add r27, r21, r10*	*add r21, r21, r31*
add r27, r21, r10	*add r6, r6, r31*	*add r6, r1, r21*
store r27, 92(r4)	*add r37, r31, r10*	*store r7, 100(r4)*
load r31, 88(r2)	*store r7, 100(r4)*	*store r17, 96(r4)*
add r6, r6, r31	*store r17, 96(r4)*	*store r27, 92(r4)*
add r37, r31, r10	*store r27, 92(r4)*	*store r37, 88(r4)*
store r37, 88(r4)	*store r37, 88(r4)*	

NOTA: Dado que solo se almacena r7 en memoria, damos prioridad a esas instrucciones en la versión final

Ciclo	LD/ST/Bcc	ALU enteros
1	load r1, 100(r2)	nop
2	load r11, 96(r2)	nop
3	load r21, 92(r2)	add r7, r1, r10
4	load r31, 88(r2)	add r17, r11, r10
5	store r7, 100(r4)	add r27, r21, r10
6	store r17, 96(r4)	add r37, r31, r10
7	store r27, 92(r4)	add r1, r1, r11
8	store r37, 88(r4)	add r21, r21, r31
9	nop	add r6, r1, r21