Considera el siguiente código como parte de un bucle principal (10^7 iteraciones) de un cierto programa que se ejecuta sobre un procesador MIPS 3000 con las siguientes características:

Frecuencia = 50MHz
Adelantamiento no incorporado
Unidades funcionales no segmentadas.
Latencia FPU = 2
Acceso simultaneo a etapas MEM y WB permitido
El compilador no aplica optimizaciones

```
(1) addd f4, f4, f0
(2) subd f6, f6, f2
(3) addi r4, r4, r0
(4) subi r6, r6, r4
(5) subi r30, r30, 1
```

Responder a las cuestiones que siguen respecto al mismo:

— Rellenar las filas necesarias de la tabla siguiente con los tipos de riesgos detectados, la primera instrucción que se detiene en cada caso (utilizar la referencia numérica del listado) y los ciclos perdidos por iteración debidos al riesgo.

Tipo de riesgo	Instruc.	Ciclos / iter.
Estructural	2	1
Dependencia de datos	4	2

Explicación: La instrucción (2) debe esperar a que finalice la etapa de EJE de la instrucción (1) para poder entrar ella en esa etapa, dado que la UF de suma flotante no esta segmentada. Por otro lado, la instrucción (4) utiliza r4, que es actualizado por la instrucción (3).

— Considerar ahora que el procesador incorpora adelantamiento. ¿Cuántos ciclos perdidos por iteración tendríamos en ese caso?

1 ciclo

Explicación: El adelantamiento consigue eliminar la perdida de ciclos por dependencia de datos.

Considera ahora el siguiente código como parte de un bucle principal (10^7 iteraciones) de un cierto programa que se ejecuta sobre un procesador MIPS 3000.

```
subui r30, r30, 1
add r2, r2, r6
lw r4, 0(r2)
add r2,r4,r2
sw 0(r4), r2
```

Responder a las cuestiones que siguen respecto al mismo:

— ¿Cuántos ciclos perdidos por iteración tendríamos al ejecutar este código sin y con adelantamiento, respectivamente?

Sin: 6 ciclos Con: 1 ciclo

Explicación: Sin adelantamiento hay 2 ciclos de perdida por cada dependencia de datos. Con adelantamiento se logran eliminar todos los riesgos por dependencias de datos excepto uno de los dos ciclos de perdida provocados por la instrucción LW.

Los fragmentos de código que aparecen a continuación son parte de unos ejercicios que permiten estudiar los riesgos de segmentación en el simulador Windlx configurando la latencia de las unidades funcionales con 2 ciclos la de suma y con 5 ciclos la de multiplicación

A
```
sub r30, r30, 1
addd f4, f0, f2
ld f8, 0(r8)
add r5, r7, r6
multd f6,f8,f10
xor r2,r2,r2
```

B
```
sub r30, r30, 1
multd f4, f0, f2
xor r4, r3, r2
add r5, r7, r6
addd f6, f8, f10
xor r2, r2, r2
```

C
```
sub r30, r30, 1
addd f4, f0, f2
xor r4,r3,r3
multd f6,f8,f10
beqz r30, loop
xor r2,r2,r2
```

D
```
sub r30, r30, 1
multd f4, f0, f2
xor r4,r3,r2
add r5, r7, r6
multd f6,f8,f10
xor r2,r2,r2
```

E
```
sub r30, r30, 1
addd f4, f0, f2
xor r4,r3,r3
ld f8, 0(r8)
multd f6,f8,f10
xor r2,r2,r2
```

Indica para cada una de las siguientes afirmaciones, cuál o cuales de los fragmentos de código anteriores las cumplen:

— No presenta ningún tipo de riesgo

B

Explicación: El codigo del fragmento B es el unico que no presenta ningún tipo de riesgo, ni estructural, ni de dependencia de datos, ni de control.

— ¿Qué fragmento de código es un ejemplo de un riesgo estructural con perdida de ciclos?

D

Explicación: Dado que la latencia de UF de multiplicación es 5, cuando la 2º instrucción MULT de este codigo debe entrar en la etapa EJE, aún la esta utilizando la 1ª instrucción MULT.

— ¿Qué fragmento de código es un ejemplo de un riesgo por dependencia de datos que puede ser eliminado mediante adelantamiento?

A

Explicación: Tanto en el fragmento A como en el E hay dependencia de datos con perdida de ciclos entre las instrucciones [ld f8, 0(f8)] y [multd f6, f8,f10]. Con adelantamiento, en A se elimina totalmente la perdida de ciclos, pues a pesar de ser provocada por una instrucción de carga hay una instrucción intercalada entre ambas. En el fragmento E en cambio, con adelantamiento se eliminaría sólo uno de los dos ciclos de perdida, ya que las instrucciones son consecutivas.

— ¿Qué fragmento de código es un ejemplo de un riesgo de control?

C

Explicación: Es el único donde aparece una instrucción de salto.

El siguiente código es ejecutado en un procesador segmentado como el estudiado en clase con latencias de suma y multiplicación flotante de 2 y 5 ciclos respectivamente.

```
 1  loop:
 2    subui r30, r30, 1
 3
 4    and r3, r4, r5
 5    addd f0, f2, f4
 6    multd f6, f8, f4
 7    xor r6, r7, r8
 8    sub r9, r10, r11
 9    bnez r30, loop
10    trap #0
```

Responder a las tres cuestiones que siguen:

— Indica un ejemplo de instrucción que situada en el hueco 3 de lugar a una depedencia de datos NO superada con adelantamiento.

lw r5, (r1)

Explicación: Ha de plantearse una dependencia de lectura después de escritura entre instrucciones consecutivas y provocada además por una instrucción de carga. Así pues, la instrucción ha de ser una de carga que escriba sobre uno de los registros que lee la siguiente instrucción.

— Indica un ejemplo de instrucción que situada en el hueco 3 de lugar a un riesgo estructural por falta de segmentación de alguna Unidad Funcional que provoque perdida de ciclos.

multd f0, f0, f2

Explicación: Ha de ser una instrucción flotante, ya que son las unidades de ejecución flotantes las que tienen latencias mayores de un ciclo. No puede ser una de suma, ya que la unidad de suma tiene una latencia de solo 2 ciclos y la siguiente instrucción no es otra suma, con lo cual no habría problema estructural. Ha de ser pues una multiplicación, que al ejecutarse en una unidad con latencia de 5 ciclos sí plantearía un conflicto con la otra multiplicación que entra al cauce 3 ciclos después.

— A la vista del programa, indica cual de los métodos de resolución de riesgos de control se deduce que NO está implementado en este procesador y por qué.

El salto retardado, porque en caso de estar implementado se ejecutaría la instrucción trap durante el hueco de retardo de salto y finalizaría el programa después de la primera iteración.

❑ Dado el siguiente código fuente:

```
1 or    r2,   r3,   r4
2 addd  f0,   f2,   f4
3 and   r5,   r2,   r6
4 subd  f6,   f0,   f8
5 xor   r8,   r2,   r10
```

completar el cronograma que resulta al ejecutarlo en una máquina con arquitectura DLX (como los simuladores de prácticas) hasta que finaliza la etapa WB de la instrucción xor

Configuración del procesador:

- Implementa la técnica de adelantamiento
- MEM y WB soportan 2 instrucciones simultaneas
- Suma flotante NO segmentada y con latencia de 2 ciclos

	1	2	3	4	5	6	7	8	9	10	11
or	IF	ID	EXE	MEM	WB						
addd		IF	ID	ADDD	ADDD	MEM	WB				
and			IF	ID	EXE	MEM	WB				
subd				IF	ID	SUBD	SUBD	MEM	WB		
xor					IF	ID	EXE	MEM	WB		