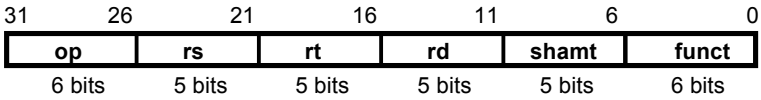
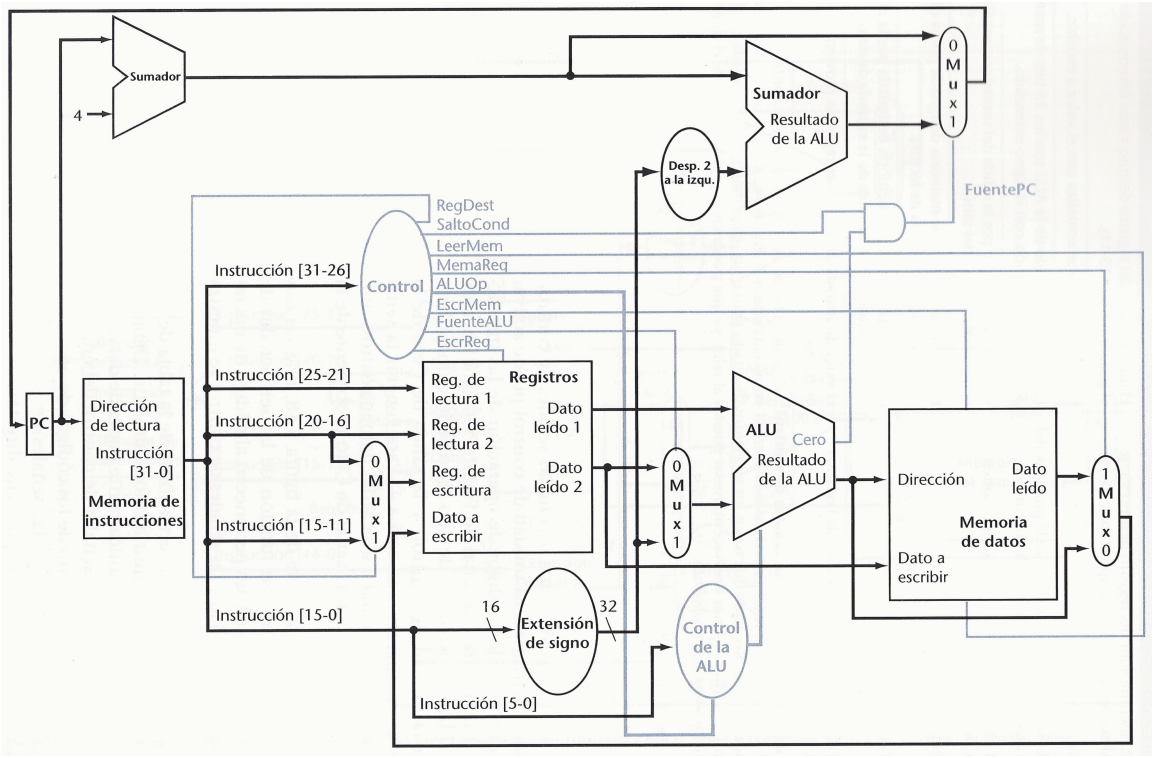
P2.1. Se tiene un sencillo sistema RISC con control en un único ciclo (similar al descrito en la asignatura). Las instrucciones son de 32 bits y los bits se distribuyen como en la figura.



La unidad de control decodifica la instrucción en función del los 6 bits de mayor peso (31:26) generando las 9 señales de control que se muestran en el gráfico. La acción de cada señal se detalla a continuación:

Señal	Acción
RegDst	Control del multiplexor de registro de destino
SaltoCond (*Branch*)	Señal de habilitación del multiplexor de salto. Si además el resultado de la operación es cero carga los 16 bits de menor peo de la instrucción desplazados en el contador de programa PC
LeerMen (*MemRead*)	Señal de habilitación de lectura de la memoria de datos.
MemaReg (*Mem2Reg*)	Control del multiplexor de memoria (1, dato de mem; 0, dato directo de ALU)
ALUOp	2 bits que indican que hará la ALU. 00 se utiliza la ALU para cálculo de direcciones, 10: operación entre registros (depende de los bits 5:0 de la instrucción); 01 si es una operación de salto; 11 prohibido.
EscrMem (*MemWrite*)	Habilitación de escritura en memoria de datos
FuentALU (*ALUScr*)	Multiplexor de la segunda entrada a la ALU
EscrReg (*RegWrite*)	Habilitación de escritura en los registros.



a) Complete en el cuadro adjunto la salida del bloque conbinacional "control" dependiendo de los tipos de instrucciones que se ejecuten. (Utilice el símbolo X para indicar "no importa")

Instrucción	RegDst	ALUSrc	Mem2Reg	RegWrite	MemRd	MemWr	Branch	ALUOp1	ALUOp0

Nota: Descripción de las instrucciones:

ADD rs, rt, rd	Suma el contenido de los registros R(rs) y R(rt) y lo guarda en el registro R(rd)
LOAD rs, rt, inmediato	carga en el registro R(rt) el contenido de memoria de la posición MEM(sign_ext(inmediato) + R(rs))
STORE rs, rt, Inmediato	guarda en la posición MEM(sign_ext(inmediato) + R(rs)), el contenido del registro R(rt)
BEQ rs, rt, Inmediato	si R(rs) = R(rt), el PC (programm counter) se carga con PC+ sign_ext (inmediato) * 4, Si no con PC+4
AND rs, rt, rd	Realiza el AND bit a bit del contenido de los registros R(rs) y R(rt) y guarda el resultado en el registro R(rd)

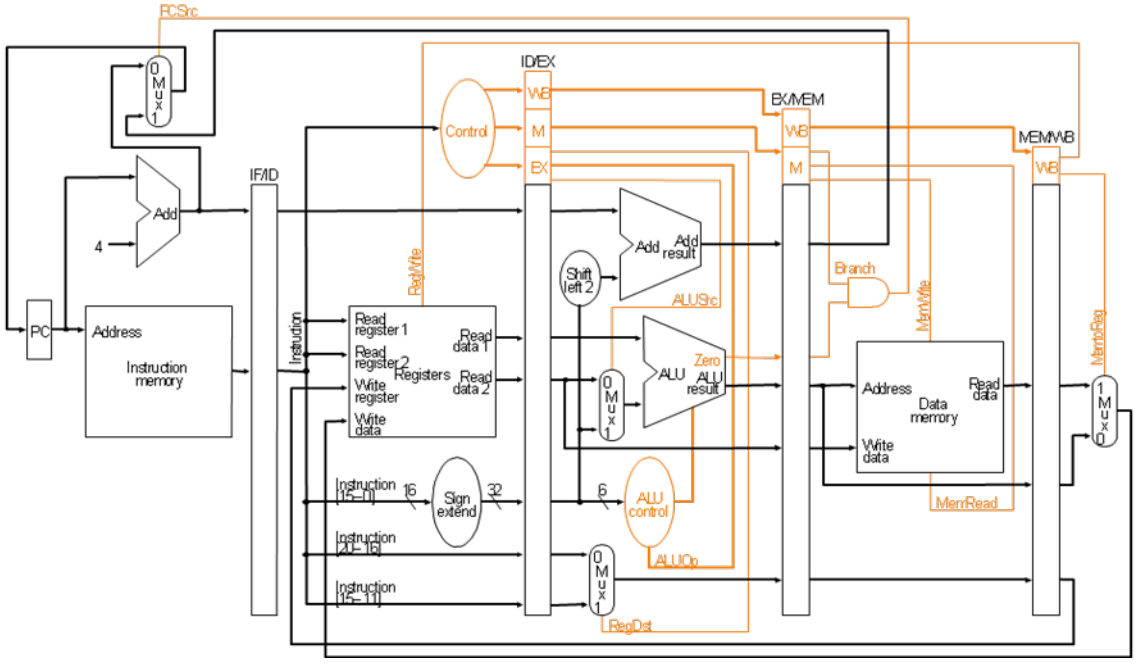
b) ¿Cuál es la principal desventaja del control uniciclo que motiva la utilización de arquitecturas multiciclo?

Típicamente tiempo de ciclo muy largo. Todas las instrucciones utilizan, sin necesidad, tanto tiempo como la instrucción más lenta.

c) ¿Cuál es el CPI en una arquitectura con control uniciclo sin considerar detenciones en la memoria?

Uno.

P2.2. Se tiene un sencillo sistema RISC segmentado en 5 etapas con control en un único ciclo (similar al descrito en la asignatura). Las instrucciones son de 32 bits y los bits se distribuyen como en la figura.



La unidad de control se puede implementar en VHDL de la siguiente forma:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity control is
  port(
    opcode : in STD_LOGIC_VECTOR(5 downto 0);
    I_Rdy : in STD_LOGIC; -- No se usa en esta versión
    D_Rdy : in STD_LOGIC; -- No se usa en esta versión
    ALUSrc : out STD_LOGIC;
    ALUOp : out STD_LOGIC_VECTOR(1 downto 0);
    RegDst : out STD_LOGIC;
    BrCond : out STD_LOGIC;
    MemRead : out STD_LOGIC;
    MemWrite : out STD_LOGIC;
    MemToReg : out STD_LOGIC;
    RegWrite : out STD_LOGIC
  );
end control;

architecture control_arq of control is
begin

  ALUSrc <= '1' when (opcode="100011" or opcode="101011" or opcode="001111")
          -- lw, sw y lui usan el dato inmediato
          else '0';

  ALUOp <= "00" when (opcode="100011" or opcode="101011") -- lw y sw
         else "01" when (opcode="000100") -- beq
         else "10" when (opcode="000000") -- add, sub, and, or, slt
         else "11" when (opcode="001111") -- lui
         else "00"; -- resto de instrucciones
```

```
  MemToReg <= '1' when (opcode="100011") -- lw
            else '0'; -- resto de instrucciones

  RegWrite <= '1' when (opcode="100011" or opcode="000000" or opcode="001111")
            -- escriben en registro lw, add, sub, and, or, slt y lui
            else '0'; -- resto de instrucciones
end control_arq;
```

Si se ejecutan las instrucciones:

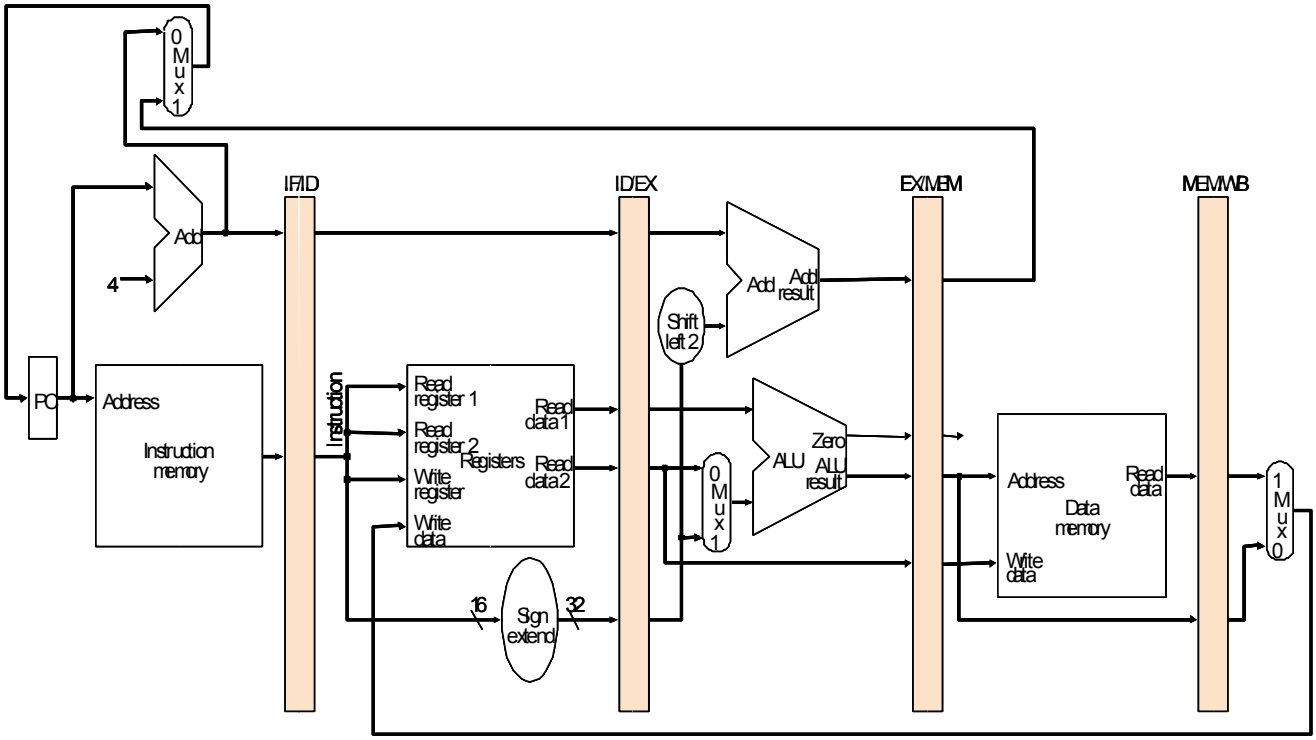
```
lw $t1, 24($zero)   # lw $r9, 24($r0)
lw $t2, 28($zero)   # lw $r10, 28($r0)
add $t7, $t1, $t3    # add $r15, $r9, $r11 => r15=r9+r11
and $s0, $t1, $t2   # and $r16, $r9, $r10 => r16=r9 and r10
sw $t7, 40($zero)  # sw $r10, 40($r0)
sw $s0, 44($zero)  # sw $r11, 44($r0)
```

Se pide completar la tabla a partir del momento en que se decodifica el segundo load (lw).

	T:lw $t1	T+1	T+2	T+3	T+4	T+5	T+6	T+7	T+8
ALUSrc	1	1	0	0	1	1			
ALUOp	00	00	10	10	00	00			
RegDst	0	0	1	1	X	X			
BrCond	0	0	0	0	0	0			
MemRead	1	1	0	0	0	0			
MemWrite	0	0	0	0	1	1			
MemToReg	1	1	0	0	0	0			
RegWrite	1	1	1	1	0	0			
IDEX_ALUSrc	-	1	1	0	0	1	1		
IDEX_ALUOp	-	00	00	10	10	00	00		
IDEX_RegDst	-	0	0	1	1	X	X		
IDEX_BrCond	-	0	0	0	0	0	0		
IDEX_MemRead	-	1	1	0	0	0	0		
IDEX_MemWrite	-	0	0	0	0	1	1		
IDEX_MemToReg	-	1	1	0	0	0	0		
IDEX_RegWrite	-	1	1	1	1	0	0		
EXMEM_BrCond	-	-	0	0	0	0	0	0	
EXMEM_MemRead	-	-	1	1	0	0	0	0	
EXMEM_MemWrite	-	-	0	0	0	0	1	1	
EXMEM_MemToReg	-	-	1	1	0	0	0	0	
EXMEM_RegWrite	-	-	1	1	1	1	0	0	
MEMWR_MemToReg	-	-	-	1	1	0	0	0	0
MEMWR_RegWrite	-	-	-	1	1	1	1	0	0

Donde el primer grupo de señales son las que genera la unidad de control, las que tienen el prefijo IDEX

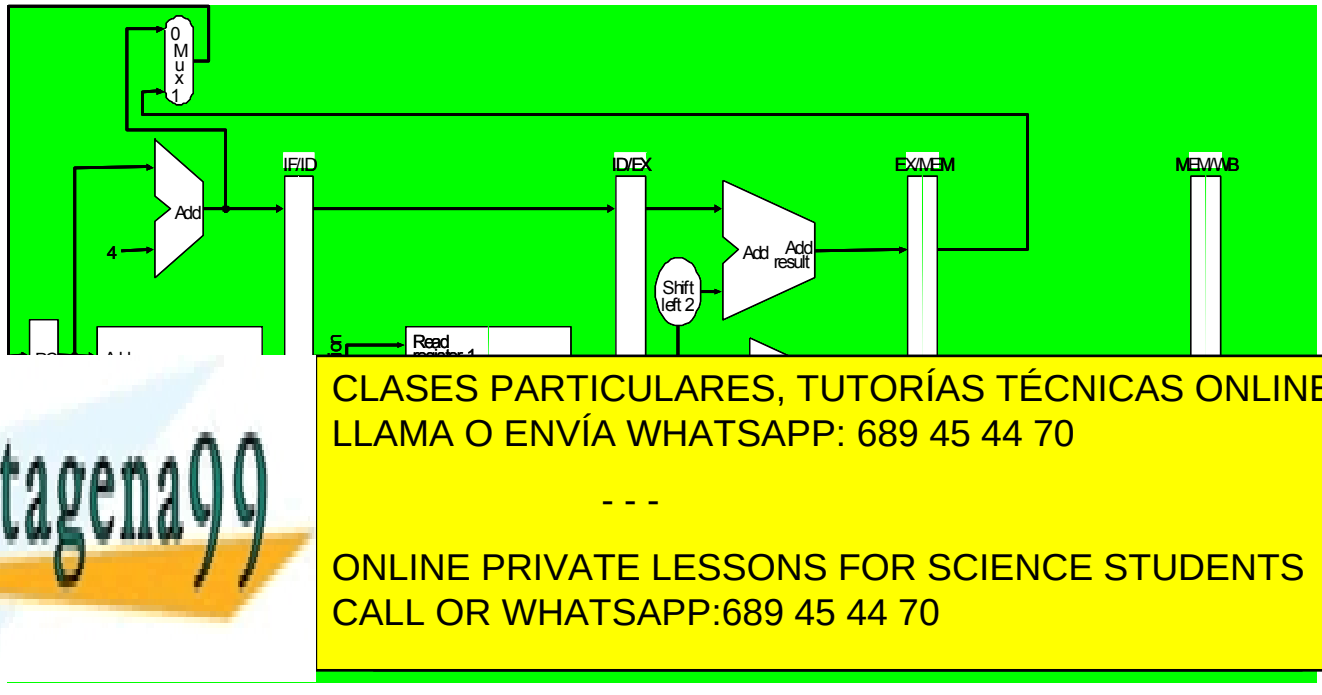
P2.3. La siguiente versión del procesador MIPS, tal como se ha estudiado en clase, tiene un problema a la hora de escribir datos en el banco de registros.



a) Indicar brevemente en palabras cuál es el problema.

El problema es que la señal que dirección de escritura (Write register) pertenece a una instrucción posterior a la del dato que se va a escribir (Write data). La información del registro de escritura debe avanzar con su instrucción correspondiente.

b) Indicar el cambio que se debe realizar sobre el dibujo de arriba.



c) Suponiendo que se tiene el código VHDL correspondiente al dibujo de arriba, escribir en VHDL las líneas para implementar el cambio planteado.

Hay que registrar la señal “Write register” y transmitirla en cada etapa de registros de segmentación:

```
process (clk) -- registro de segmentación entre ID y EX
  if rising_edge (clk) then
    …
    write_register_id_ex <= write_register_if_id;
    …
  end if;
end process;
```

Lo mismo para la etapa EX_MEM y MEM_WB

En el banco de registros, para la entrada de dirección de escritura hay que conectar la señal proveniente de la etapa MEM_WB:

```
port map (…
  write_register => write_register_mem_wb,
  …);
```

P2.4. Indique todos los riesgos que presenta el siguiente fragmento de código si se ejecuta en un sistema con arquitectura Von Neumann (memoria común para instrucciones y datos). El procesador esta segmentado con cauce único y como máximo captura una instrucción por ciclo de reloj.

```
L1:   I1 LD R8, 4(R5)        ; R8  = Mem[R5+4]
      I2 ADD R6, R3, R8              ; R6 =  R3 + R8
      I3 AND R7, R8, R6              ; R7 =  R8 & R6
      I4 BEQ R8, R7, L1              ; Si R7 = R8 salta a L1
      I5 ST R8, 4(R1)        ; Mem[R1+4] = R8
```

Defina brevemente todos los tipos de riesgos que se deben considerar. Concrete en este código y para este sistema cuales pueden aparecer.

Riesgos estructurales:
Son riesgos debidos a que dos instrucciones intentan acceder en el mismo ciclo al mismo recurso. Como el bus de acceso a memoria es único en la etapa que I1 e I5 accedan a memoria de datos no será posible capturar una nueva instrucción.

Riesgos de Control:
Son riesgos debidos a cambios en la secuencia de ejecución de instrucciones
La I4 es un salto y dependiendo de la condición será necesario ejecutar I1 o I5. Dependiendo de la etapa donde se calcule el destino de salto, de donde se evalúe la condición y si realiza predicción correctamente, se necesitarán diferentes ciclos de espera.

Riesgos por dependencias de datos:
RAW Son riesgos debidos a que la instrucción posterior necesita (lee) el dato que genera (escribe) una instrucción anterior.

Riesgos RAW del código:

I2 con I1 por R8
I3 con I2 por R6
I3 con I1 por R8
I4 con I2 por R6
I4 con I3 por R7
I5 con I4 por R8

Al ser de cauce simple no se presentan riesgos por dependencias de datos WAR ni WAW

P2.5. Se dispone de un procesador segmentado de cinco estados: (1) BI: Búsqueda de la instrucción; (2) DI: Decodificación de la instrucción y búsqueda de datos en los registros internos; (3) (EX) Ejecución en la ALU de la instrucción; (4) (MEM) Leer/escribir en memoria y (5) (REG) Escritura en los registros internos. Aunque una instrucción no necesite utilizar un determinado segmento, consume un ciclo de reloj para atravesar dicho segmento.
La sintaxis de las instrucciones es: "OP RDESTINO,RFUENTE,RFUENTE" y se dispone de la siguiente secuencia de cinco instrucciones:

I1: SUB R2, R1, R3
I2: AND R4, R2, R5
I3: OR R8, R2, R6
I4: ADD R9, R4, R2
I5: SUB R1, R6, R7

a) Detectar los posibles riesgos por dependencia de datos, señalando el tipo.
b) Escribir el cronograma del procesador para los siguientes casos:

i) Al detectar un riesgo de datos en una determinada instrucción el procesador detiene el procesado de esa instrucción y de todas las posteriores hasta que el riesgo sea solucionado.
ii) Se introduce una mejora en la que es posible adelantar datos una vez terminado el ciclo de ejecución en la ALU.

Nota: se supone que no se puede escribir y leer en el mismo registro en un solo ciclo de reloj.

SOLUCIÓN:

a) Se detectan riesgos de tipo RAW en: I2 con I1 por R2, I3 con I1 por R2, I4 con I1 por R2 e I4 con I2 por R4. Se detectan riesgos de tipo WAR en: I5 con I1 por R1.

b)

i)	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	14
BI	I1	I2	I3	I3	I3	I3	I4	I5	I5	I5	-	-	-	-
DI	-	I1	I2D	I2D	I2D	I2	I3	I4D	I4D	I4	I5	-	-	-
EX	-	-	I1	-	-	-	I2	I3	-	-	I4	I5	-	-
MEM	-	-	-	I1	-	-	-	I2	I3	-	-	I4	I5	-
REG	-	-	-	-	I1	-	-	-	I2	I3	-	-	I4	I5

ii)	T1	T2	T3	T4	T5	T6	T7	T8	T9
BI	I1	I2	I3	I4	I5	-	-	-	-
DI	-	I1	I2	I3	I4	I5	-	-	-
EX	-	-	I1	I2	I3	I4	I5	-	-
MEM	-	-	-	I1	I2	I3	I4	I5	-
REG	-	-	-	-	I1	I2	I3	I4	I5

P2.6. Se dispone de un procesador RISC segmentado en cuatro estados, S1: búsqueda de la instrucción. S2: Decodificación y detección de los riesgos de datos y/o control, cálculo de la dirección efectiva y en su caso captura de los operandos desde registros internos. S3: Ejecución en la ALU y en su caso lectura y/o escritura de datos en memoria y S4: Escritura del resultado en un registro interno. El sistema dispone de un único bus para instrucciones y datos y sólo se permite una operación (leer/escribir) por ciclo en un mismo registro. Dibujar los cronogramas correspondientes para calcular la mejora en ciclos en el programa dado cuando se dota al sistema con la capacidad de adelantar datos "*Internal Forwarding*" en el caso de que dichos datos sean generados en la ALU.

NOTA: Todas las instrucciones pasan por los cuatro estados. Cuando se detecta un riesgo, el sistema se detiene hasta su solución. No se puede escribir y leer un dato en el banco de registros en el mismo ciclo. El primer operando de una instrucción indica siempre el destino de la misma.

```
LOAD    R3, (R8)        ; Direccionamiento indirecto por registro.
ADD     R1, R2, R3
INC     R3
STORE   (R9), R1        ; Direccionamiento indirecto por registro.
SUB     R4, R3, R1
AND     R5, R4, R3
OR      R6, R5, R4
```

SOLUCIÓN:

Con * se indica el uso de los buses con memoria por alguno de los segmentos. En paréntesis se indican las instrucciones en espera de resolver el riesgo encontrado.

a) Sin usar "Internal Forwarding":

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17
LOAD	S1*	S2	S3*	S4													
ADD		S1*	(S2)	(S2)	S2	S3	S4										
INC				S1*	(S1)	S2	S3	S4									
STORE						S1*	(S2)	S2	S3*	S4							
SUB							S1*	(S1)	S2	S3	S4						
AND										S1*	(S2)	S2	S3	S4			
OR											S1*	(S1)	(S2)	(S2)	S2	S3	S4

b) Usando "Internal Forwarding":

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
LOAD	S1*	S2	S3*	S4										
ADD		S1*	(S2)	(S2)	S2	S3	S4							
INC				S1*	(S1)	S2	S3	S4						
STORE						S1*	S2	S3*	S4					
SUB							S1*	S2	S3	S4				
AND									S1*	S2	S3	S4		
OR										S1*	S2	S3	S4	

El ahorro con la mejora para el programa dado supone cuatro ciclos de reloj.

P2.7. Un procesador segmentado con 5 etapas

IF: Lectura de instrucción
ID: Decodificación de instrucción y lectura de operandos
EX: Ejecución de operación y cálculo de la dirección efectiva.
DM: Acceso a la memoria
WB: Escritura en los registros. (Asumir que se puede escribir y leer en el mismo ciclo)

Ejecuta el siguiente código

Instrucción	Código	Función
1	LW R1,2000 (R0)	R1 ← M[R0+2000]
2	LW R2,2004 (R0)	R2 ← M[R0+2004]
3	ADD R3,R2,R1	R3 ← R2+R1
4	ADDI R1,R2,8	R1 ← R2+8
5	SUBI R4,R0,2	R4 ← R0-2
6	AND R5,R3,R2	R5 ← R3 and R2
7	SW R4,2000 (R0)	M[R0+2000] ← R4
8	SW R5,2004 (R0)	M[R0+2004] ← R5
9	SW R1,2008 (R0)	M[R0+2008] ← R1

a) Identificar, señalando el tipo, todos los posibles conflictos que pueden surgir por dependencias entre operandos.

b) Suponiendo que el procesador no tiene la posibilidad de parar el *pipeline*, insertar en el programa fuente el mínimo número de instrucciones NOP que se consideren necesarias para eliminar las dependencias de datos. Asumir que el procesador no dispone de mecanismos de "*Forwarding*" (avance de resultado en la salida de la ALU) y que no se puede cambiar el orden de la secuencia de instrucciones

c) Representar gráficamente paso a paso el estado del *pipeline*. Rellenar el cuadro adjunto de la misma manera a como se muestra para la primera instrucción.

d) ¿Cuál es la mejora producida por utilizar segmentación?

SOLUCIÓN:

a) Los riesgos posibles por dependencia de datos son:

RAW	RAW	WAR	WAW
I3 con I1 por R1	I3 con I2 por R2	I4 con I3 por R1	I4 con I1 por R1
I4 con I2 por R2	I6 con I2 por R2	I7 con I1 por M[...]	
I6 con I3 por R3	I7 con I5 por R4	I8 con I2 por M[...]	
I8 con I6 por R5	I9 con I1 por R1		
I9 con I4 por R1			

b) y c) Se supone un único bus para instrucciones y datos. Se permite leer/escribir de/en un mismo registro en el mismo ciclo de reloj.

Instr.	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17
1	IF1*	ID	EX	DM*	WB												
2		IF2*	ID	EX	DM*	WB											
3			NOP														
4				NOP													
5					NOP												
6						IF3*	ID	EX	DM	WB							

7 = 2,65. La segmentación es mejor en un 165%.

P2.8. Un sistema RISC con arquitectura Harvard segmenta la ejecución de cada una de las instrucciones en las siguientes etapas: IF (captura), ID (decodificación, captura de operandos y detección de riesgos), EX (opera en la ALU, calcula la dirección y calcula la condición en saltos), M (acceso a memoria) y WB (escritura de registro). El procesador emite una instrucción por ciclo con ejecución en orden y todas las instrucciones pasan por todas las etapas. Los saltos condicionales se manipulan con la técnica de predecir-no-efectivo pero el contador de programa se actualiza en las instrucciones de salto en la etapa M.

Considere el siguiente fragmento de código, donde se conoce que los 6 primeros saltos en I8 son efectivos. La sintaxis de los operandos es en todos los casos $R_{DESTINO}$, R_{FUENTE}, R_{FUENTE}.

I1: ADD R5, R0, R0
I2: Loop: LD R1, 0(R2)
I3: ADD R6, R1, R5
I4: ADD R5, R1, R0
I5: STO 0(R2), R6
I6: ADDI R2, R2, #4
I7: SUB R4, R3, R2
I8: BNZ R4, Loop

a) Señalar los riesgos de datos.
b) Dibujar una tabla en la que se muestre la evolución temporal de la secuencia de instrucciones en la segmentación sin considerar ningún tipo de adelantamiento salvo que las etapas ID y WB pueden acceder en el mismo ciclo de reloj al banco de registros (la etapa WB accede en la primera mitad y la etapa ID en la segunda mitad).
c) Dibujar una tabla similar a la del apartado (a) pero considerando adelantamiento de datos entre etapas.
d) Calcular para cada uno de los dos supuestos anteriores cuántos ciclos de reloj tarda en ejecutarse el código anterior para los 6 primeros saltos.

SOLUCIÓN:

a) Los riesgos de datos que se detectan son:

RAW	WAR	WAW
I3 con I1 por R5 I3 con I2 por R1 I4 con I2 por R1 I5 con I3 por R6 I7 con I6 por R2 I8 con I7 por R4	I4 con I3 por R5 I5 con I2 por acceso a M[0+R2] I6 con I2 por R2 I6 con I5 por R2	I4 con I5 por R1

b) SIN adelantamiento de datos.

Instr. \ Ciclo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
I1: ADD R5, R0, R0	IF	ID	EX	M	WB														
I2: Loop: LD R1, 0(R2)		IF	ID	EX	M	WB													
I3: ADD R6, R1, R5			IF	ID	ID	ID	EX	M	WB										
I4: ADD R5, R1, R0				IF	IF	IF	ID	EX	M	WB									
I5: STO 0(R2), R6							IF	ID	ID	EX	M	WB							

Instr. \ Ciclo	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
I7: SUB R4, R3, R2	WB							
I8: BNZ R4, Loop	ID	EX	M	WB				
I2: **Loop : LD R1, 0(R2)**				IF	ID	EX	M	WB

Ciclo 6: I2 escribe R1 e I3 puede leerlo

Ciclo 9: I3 escribe R6 e I5 puede leerlo

Ciclo 13: I6 escribe R2 e I7 puede leerlo

Ciclo 16: I7 escribe R4 e I7 puede comprobar si su valor no es cero.

En el Ciclo 18 se actualiza en la etapa Mem el contador de programa con la dirección destino

d1) Ejecución de 8 iteraciones: Tiempo = 1 + 18 + 5 x 17 = 104 ciclos

La instrucción I1 inicial añade un ciclo.
La primera iteración de bucle I2-I8 tarda 18 ciclos
Las siguientes iteraciones tarda 17 ciclos por el solape de 1 ciclo (ver ciclo 19).

c) CON Adelantamiento de datos.

Instr. \ Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1: ADD R5, R0, R0	IF	ID	EX	M	WB												
I2: Loop: LD R1, 0(R2)		IF	ID	EX	M	WB											
I3: ADD R6, R1, R5			IF	ID	ID	EX	M	WB									
I4: ADD R5, R1, R0					IF	ID	EX	M	WB								
I5: STO 0(R2), R6						IF	ID	EX	M	WB							
I6: ADDI R2, R2, #4							IF	ID	EX	M	WB						
I7: SUB R4, R3, R2								IF	ID	EX	M	WB					
I8: BNZ R4, Loop									IF	ID	EX	M	WB				
I2: Loop : LD R1, 0(R2)													IF	ID	EX	M	WB

Adelantamientos: Acceso a Memoria de I2 adelanta su resultado (valor de r1) a Ejecución de I3

Ejecución de I3 adelanta su resultado (valor de r6) a Decodificación de I5

Ejecución de I6 adelanta su resultado (valor de r2) a Ejecución de I7

Ejecución de I7 adelanta su resultado (valor de r4) a Ejecución de I8

d2) Tiempo = 1 ciclo (I1) + 12 ciclos (1ª iteración) + 5 x 11 (otras iteraciones) = 68 ciclos.

P2.9. Se ejecutan el siguiente código en un procesador RISC con arquitectura Harvard, segmentado en 5 etapas: **F:** Captura; **DE:** Decodificación y captura de operandos desde registros; **EX**: Ejecución; **MEM**: Memoria (de datos) y **WR**: Escritura (en registro). Las instrucciones se ejecutan en orden y sin adelantamiento de datos. Suponer que las escrituras se producen en la primera mitad de la etapa WR y que las lecturas en la etapa DE se producen en la segunda mitad de la misma.

I1:	ADD R3, R1, R2	;R3 <- R1+R2	**I5:**	OR R1, R3, R6	;R1 <- R3 or R6
I2:	LD R1, 0(R4)	;R1 <- M[0+R4]	**I6:**	ST R1, 4(R4)	;M[4+R4] <- R1
I3:	AND R5, R3, R4	;R5 <- R3 and R4	**I7:**	LD R2, 4(R4)	;R2 <- M[4+R4]
I4:	AND R6, R1, R2	;R6 <- R1 and R2	**I8:**	SUB R3, R5, R6	;R3 <- R5 – R6

a) Indicar todos los riesgos potenciales de datos entre cualesquiera dos instrucciones. Señalar de los anteriores cuáles causan riesgos reales en el procesador planteado y el número de ciclos necesarios para completar la ejecución.
b) Indicar cuántas operaciones NOP habría que intercambiar para no provocar detenciones y qué mejora se obtiene en dicho caso.
c) Indicar la mejora que se alcanza permitiendo el adelantamiento de datos entre etapas, suponiendo que se hace de la mejor forma posible. Indicar expresamente en cada caso la instrucción que adelanta el dato, en qué etapa se genera el dato que se adelanta, qué otra instrucción está implicada y en qué etapa.

SOLUCIÓN:

a) Los riesgos posibles por dependencia de datos son:

RAW	WAR	WAW
I3 con I1 por R3 (real)	I2 con I1 por R1	I8 con I1 por R3
I4 con I2 por R1 (real)	I5 con I1 por R1	I5 con I2 por R1
I5 con I1 por R3	I5 con I4 por R1	
I5 con I4 por R6 (real)	I7 con I1 por R2	
I6 con I2 por R1	I7 con I4 por R2	
I6 con I5 por R1(real)	I8 con I3 por R3	
I7 con I6 por M[4+R4]	I8 con I5 por R3	
I8 con I3 por R5		
I8 con I4 por R6		

Si no se aplica ninguna mejora los riesgos reales implican ejecutar el programa en 17 ciclos.

Instrucción		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I1:	ADD R3, R1, R2	F	D	E	M	W												
I2:	LD R1, 0(R4)		F	D	E	M	W											
I3:	AND R5, R3, R4			F	(D)	D	E	M	W									
I4:	AND R6, R1, R2				(F)	F	D	E	M	W								
I5:	OR R1, R3, R6						F	(D)	(D)	D	E	M	W					
I6:	ST R1, 4(R4)							(F)	(F)	F	(D)	(D)	D	E	M	W		
I7:	LD R2, 4(R4)										(F)	(F)	F	D	E	M	W	
I8:	SUB R3, R5, R6													F	D	E	M	W

b) Una solución posible supone intercalar 5 operaciones NOP (una entre I2:I3; dos entre I4:I5 y otras dos entre I5:I6). Esto implica mantener los mismos 17 ciclos para el total de la ejecución pero se ahorra en hardware de control.

c) Si se permite el adelantamiento de datos, la mejora es de un 41% (17/12 =1,41).

P2.10. Suponga que la secuencia de código se ejecuta en un sistema con arquitectura Von Neumann, con una estructura de segmentación en las siguientes etapas: **IF** (captura), **ID** (decodificación, captura de operandos y detección de riesgos), **E1** (opera en la ALU en operaciones aritméticas y calcula PC + desplazamiento en las instrucciones de salto), **E2** (utiliza la ALU para calcular la condición en saltos), **M** (acceso a memoria) y **W** (escritura de registro).

```
I1: SUB    R1, R2, R3          ; R1 = R2 – R3
I2: LOAD   R4, 8(R1)           ; R4 = M(R1 + 8)
I3: AND    R2, R1, R4          ; R2 = R1 and R4
I4: SUB    R5, R5, 1           ; R5 = R5 –1
I5: BEQ    R5, R0, 200         ; Si R5 = R0 =0 ir a (PC=PC+200) => (terminar)
I6: BEQ    R4, R6, -24         ; Si R4 = R6 ir a I1 (PC=PC-24)
I7: STORE 8(R7), R6            ; M(R7+8)=R6
I8: BNE    R4, R6, -32         ; Si R4 es distinto de R6 ir a I1
```

Inicialmente el contador de programa contiene la dirección de la instrucción I1. El formato de las instrucciones es de 32 bits, el procesador emite una instrucción por ciclo con ejecución en orden, y todas las instrucciones pasan por todas las etapas. Se pide:

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

SOLUCIÓN:

RAW	WAR	WAW
I2 con I1 por R1	I3 con I1 por R2	No hay
I3 con I1 por R1	I7 con I2 por memoria	
I3 con I2 por R4		
I5 con I4 por R5		
I6 con I2 por R4		
I8 con I2 por R4		

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

b) En el cronograma adjunto muestre la evolución temporal de la primera iteración de la secuencia de instrucciones suponiendo que R5 es distinto de cero (p.ej. R5 = 10), y el salto de la instrucción 6 no es efectivo porque R4 y R6 son distintos. 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).

SOLUCIÓN:

Nota: Esta solución es suponiendo que las ALU de E1 y de E2 son independientes y por ello se permite que instrucciones distintas estén en E1 y E2 en un mismo ciclo. Es también válido dar la solución suponiendo que la ALU es única y que se evita la coincidencia en las etapas E1 y E2 de instrucciones que usan estas etapas de manera consecutiva (riesgo estructural).

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22
I1: SUB R1,R2,R3	IF*	ID	E1	E2	M	W															IF*	ID
I2: LOAD R4, 8(R1)		IF*	(ID)	id	id	ID	E1	E2	M*	W												
I3: AND R2, R1, R4			IF*	if	if	if	(ID)	id	id	ID	E1	E2	M	W								
I4: SUB R5,R5,1							IF*	if	if	if	ID	E1	E2	M	W							

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
I1: SUB R1,R2,R3	IF*	ID	**E1**	E2	M	W								**IF***	**ID**	**E1**	**E2**	**M**	**W**
I2: LOAD R4, 8(R1)		**IF***	**ID**	**E1**	**E2**	**M**	**W**												
I3: AND R2, R1, R4			**IF***	**(ID)**	**id**	**ID**	**E1**	**E2**	**M**	**W**									
I4: SUB R5,R5,1				**IF***	**if**	**if**	**ID**	**E1**	**E2**	**M**	**W**								
I5: BEQ R5,R0,200							**IF***	**ID**	**E1**	**E2**	**M**	**W**							
I6: BEQ R4 ,R6,-24								**IF***	**ID**	**E1**	**E2**	**M**	**W**						
I7: STORE 8(R7), R6									**IF***	**ID**	**E1**	**E2**	**M***	**W**					
I8: BNE R4,R6, -32										**IF***	**ID**	**E1**	**E2**	**M**	**W**				

d) Para cada uno de los dos apartados anteriores indique la variación que se produce en el cronograma si se modifica el sistema con una arquitectura Harvard.

No hay mejora.

P2.11. Suponga que no conoce el procesador que va a ejecutar el fragmento de código dado.

NOTA: la sintaxis de los operandos es en todos los casos $R_{DESTINO}$, R_{FUENTE}, R_{FUENTE}.

a) Analice todos los riesgos de datos presentes en el código.

Suponga ahora que la secuencia de código se ejecuta en un sistema con arquitectura Von Neumann que segmenta la ejecución de cada una de las instrucciones en las siguientes etapas: **IF** (captura), **ID** (decodificación, captura de operandos, calculo de la dirección efectiva y detección de riesgos), **EX** (opera en la ALU y calcula la condición en saltos), **M** (acceso a memoria) y **WB** (escritura de registro).

Inicialmente el contador de programa contiene la dirección de la instrucción I1. El procesador dispone de un cauce único, emite una instrucción por ciclo con ejecución y finalización en orden y todas las instrucciones pasan por todas las etapas. También se pide: **a)** Dibujar un cronograma en el que se muestre la evolución temporal de la secuencia de instrucciones en la segmentación, sin considerar ningún tipo de adelantamiento salvo que las etapas **ID** y **WB** pueden acceder en el mismo ciclo de reloj al banco de registros (la etapa **WB** accede en la primera mitad y la etapa **ID** en la segunda mitad). **b)** Dibujar un cronograma similar al del apartado b pero considerando que existe adelantamiento de datos entre etapas de la manera más eficiente posible. **c)** A la vista de los cronogramas anteriores, señalar la mejora que se produciría por disponer en el sistema de una arquitectura Harvard.

SOLUCIÓN:

	Instrucciones	Dependencias
I1	AND R4, R4, R0	
I2	JMP L1	
I3	**L0:** XOR R1, R1, R1	
I4	ADD R1, R1, 4	RAW con I3 por R1; WAR con I3 por R1 WAW con I3 por R1,
I5	JMP L2	
I6	**L1:** STORE 200(R4),R4	RAW con I1 por R4
I7	ADD R4, R4, 4	RAW con I1 por R4 WAR con I1/R4 y con I6/R4 WAW con I1 por R4,
I8	LOAD R5, (R4)	RAW con I7 por R4 y con I1 por R4 ,
I9	SUB R6, R5, 4	RAW con I8 por R5
I10	MUL R7, R5, 3	RAW con I8 por R5
I11	**L2:** STORE (R6),R7	RAW con I9 por R6 y con I10 por R7

Además de los marcados en la tabla

- I8 tiene riesgo RAW con I6 por la posibilidad de acceso a la misma posición de memoria en un procesador con planificación estática como por ejemplo un VLIW.
- I11 tiene riesgo WAR con I8 por la posibilidad de acceso a la misma posición de memoria en un procesador con planificación estática como por ejemplo un VLIW.
- I11 tiene riesgo WAW con I6 por la posibilidad de acceso a la misma posición de memoria en un procesador con planificación estática como por ejemplo un VLIW.

b) Sin adelantamiento de datos y arquitectura Von Neumann

Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I1: AND R4, R4, R0	**IF**	**ID**	**EX**	**M**	**WB**														
I2: JMP L1		**IF***	**ID**	**EX**	**M**	**WB**													
I3: L0: XOR R1, R1, R1			**IF***	**x**	**x**	**x**	**x**												
I4: ADD R1, R1, 4																			

c) Con Adelantamiento de datos y arquitectura Von Neumann

Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I1: AND R4, R4, R0	IF	ID	EX	M	WB														
I2: JMP L1		IF*	ID	EX	M	WB													
I3: L0: XOR R1, R1, R1			IF*	x	x	x	x												
I4: ADD R1, R1, 4																			
I5: JMP L2																			
I6: L1: STORE 200(R4),R4				IF*	ID	EX	M*	WB											
I7: ADD R4, R4, 4					IF*	ID	EX	M	WB										
I8: LOAD R5, (R4)						IF*	ID	EX	M*	WB									
I9: SUB R6, R5, 4							*	IF*	ID	EX	M	WB							
I10: MUL R7, R5, 3									*	IF*	ID	EX	M	WB					
I11: L2: STORE (R6),R7											IF*	ID	EX	M*	WB				

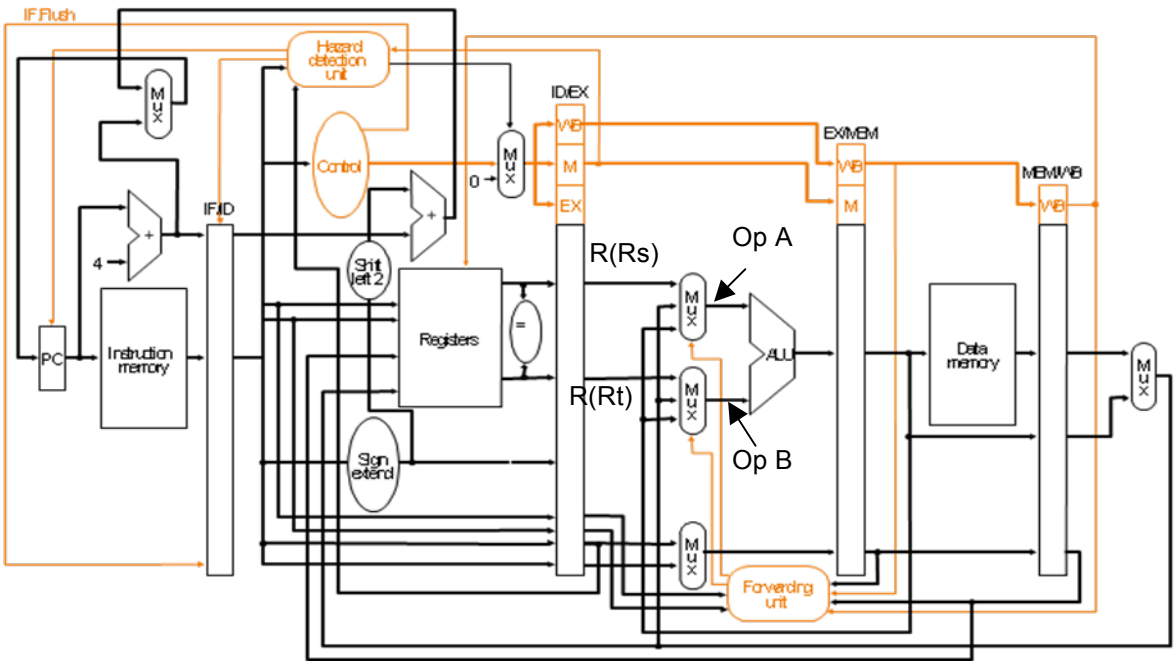
d) Sin adelantamiento de datos y arquitectura Harvard: NO MEJORA

Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I1: AND R4, R4, R0	IF	ID	EX	M	WB														
I2: JMP L1		IF*	ID	EX	M	WB													
I3: L0: XOR R1, R1, R1			IF*	x	x	x	x												
I4: ADD R1, R1, 4																			
I5: JMP L2																			
I6: L1: STORE 200(R4),R4				IF*	ID	EX	M*	WB											
I7: ADD R4, R4, 4					IF*	ID	EX	M	WB										
I8: LOAD R5, (R4)						IF*	ID	(ID)	(ID)	EX	M*	WB							
I9: SUB R6, R5, 4							IF*	IF	(IF)	(ID)	(ID)	(ID)	EX	M	WB				
I10: MUL R7, R5, 3										IF*	(IF)	(IF)	ID	EX	M	WB			
I11: L2: STORE (R6),R7													IF*	ID	ID	ID	EX	M*	WB

Con adelantamiento de datos y arquitectura Harvard: MEJORA 1 ciclo

Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I1: AND R4, R4, R0	IF	ID	EX	M	WB														
I2: JMP L1		IF*	ID	EX	M	WB													
I3: L0: XOR R1, R1, R1			IF*	x	x	x	x												
I4: ADD R1, R1, 4																			
I5: JMP L2																			

P2.12. El microprocesador MIPS estudiado en teoría y realizado en las prácticas de la asignatura posee una unidad de adelantamiento de datos (data forwarding) que permite adelantar los datos de las etapas MEM y WB a la etapa de ejecución (EX):



Los registros de las diferentes etapas se llaman PC, IF/ID, ID/EX, EX/MEM y MEM/WB respectivamente. La lógica implementada en la unidad de adelantamientos es:

Adelantamientos desde la etapa MEM	*Adelantamientos desde la etapa WB*
if (EX/MEM.EscrReg **and** EX/MEM.Reg.Rd ≠ 0 **and** EX/MEM.Reg.Rd = ID/EX.Reg.Rs) **then** AnticiparA = 10 **else** AnticiparA = 00 **if** (EX/MEM.EscrReg **and** EX/MEM.Reg.Rd ≠ 0 **and** EX/MEM.Reg.Rd = ID/EX.Reg.Rt) **then** AnticiparB = 10 **else** AnticiparB = 00	**if** (MEM/WB.EscrReg **and** MEM/WB.Reg.Rd ≠ 0 **and** EX/MEM.Reg.Rd ≠ ID/EX.Reg.Rs **and** MEM/WB.Reg.Rd = ID/EX.Reg.Rs) **then** AnticiparA = 01 **else** AnticiparA = 00 **if** (MEM/WB.EscrReg **and** MEM/WB.Reg.Rd ≠ 0 **and** EX/MEM.Reg.Rd ≠ ID/EX.Reg.Rt **and** MEM/WB.Reg..Rd = ID/EX.Reg.Rt) **then** AnticiparB = 01 **else** AnticiparB = 00

Se ejecutan en el procesador las siguientes 4 instrucciones

ARQUITECTURA DE COMPUTADORES
CAPÍTULO 2. PROCESADORES SEGMENTADOS

Determine el valor de las señales correspondientes a la ejecución del anterior código ensamblador. Complete en la tabla adjunta donde se puede ver la evolución de las instrucciones dentro del pipeline. Se debe completar solamente las señales indicadas de la etapa EX en los ciclos 4, 5 y 6.

Ciclo	IF	ID	EX	MEM	WB
1	Add r1, r2, r3	XXX r10, r1, r2			
2	Sub r5, r1, r6	Add r1, r2, r3	XXX r10, r1, r2		
3	And r6, r5, r1	Sub r5, r1, r6	Add r1, r2, r3	XXX r10, r1, r2	
4	Add r4, r1, r3	And r6, r5, r1	Sub r5, r1, r6 **(1)** anticiparA = **10** anticiparB = **00** ID/EX.Reg.Rs = **r1** ID/EX.Reg.Rt = **r6** EX/MEM.EscrReg = **1** EX/MEM.Reg.Rd = **r1** MEM/WB.EscrReg = **1** MEM/WB.Reg.Rd = **r10**	Add r1, r2, r3	XXX r10,r1, r2
5		Add r4, r1, r3	And r6, r5, r1 **(2)** anticiparA = **10** anticiparB = **01** ID/EX.Reg.Rs = **r5** ID/EX.Reg.Rt = **r1** EX/MEM.EscrReg = **1** EX/MEM.Reg.Rd = **r5** MEM/WB.EscrReg = **1** MEM/WB.Reg.Rd = **r1**	Sub r5, r1, r6	Add r1, r2, r3
6			Add r4, r1, r3 **(3)** anticiparA = **00** anticiparB = **00** ID/EX.Reg.Rs = **r1** ID/EX.Reg.Rt = **r3** EX/MEM.EscrReg = **1** EX/MEM.Reg.Rd = **r6** MEM/WB.EscrReg = **1** MEM/WB.Reg.Rd = **r5**	And r6, r5, r1	Sub r5, r1, r6
7				Add r4, r1, r3	And r6, r5, r1
8					Add r4, r1, r3

Utilice este recuadro en caso que necesite realizar alguna aclaración

(1) Se adelanta el operando A (r1) desde la etapa MEM

P2.13. Se tiene un microprocesador segmentado en las siguientes 5 etapas. **IF**: captura de instrucción. **ID**: decodificación de instrucción, detección de riesgos y captura de operandos. **EX**: ejecución en ALU, cálculo de la dirección de salto y de la dirección de acceso a memoria de datos. **M**: acceso a memoria de datos y resolución de la condición de salto. **W**: escritura en registros internos. El banco de registros es tal que permite en el mismo ciclo de reloj escribir un dato y luego leerlo. Se utiliza arquitectura Harvard y no hay detenciones en memoria. La ejecución es siempre en orden. El código a ejecutar es el siguiente, en el que se sabe que el salto condicional es efectivo:

Instr./Etiq.	**Ensamblador**		*Explicación*
I1	**ADD**	**R1, R2, R3**	*; R1 = R2+R3*
I2	**LOAD**	**R4, 0(R1)**	*; R4 = M[0+R1]*
I3	**MUL**	**R5, R4, R4**	*; R5 = R4·R4*
I4	**BNE**	**R1, R4, L1**	*; Si R1 /= R4, salta a L1 (sí se salta)*
I5	**SUB**	**R5, R5, R2**	*; R5 = R5-R2*
I6	**ADD**	**R7, R4, R5**	*; R7 = R4+R5*
I7 / L1:	**ADD**	**R6, R1, R5**	*; R6 = R1+R5*
I8:	**OR**	**R1, R7, R8**	*; R1 = R7 or R8*
I9:	**ST**	**R6, 0(R1)**	*; M[0+R1] = R6*

a) Suponiendo que no hay adelantamiento de datos (*forwarding*) y que los saltos condicionales se predicen como no efectivos, rellenar el siguiente cronograma.

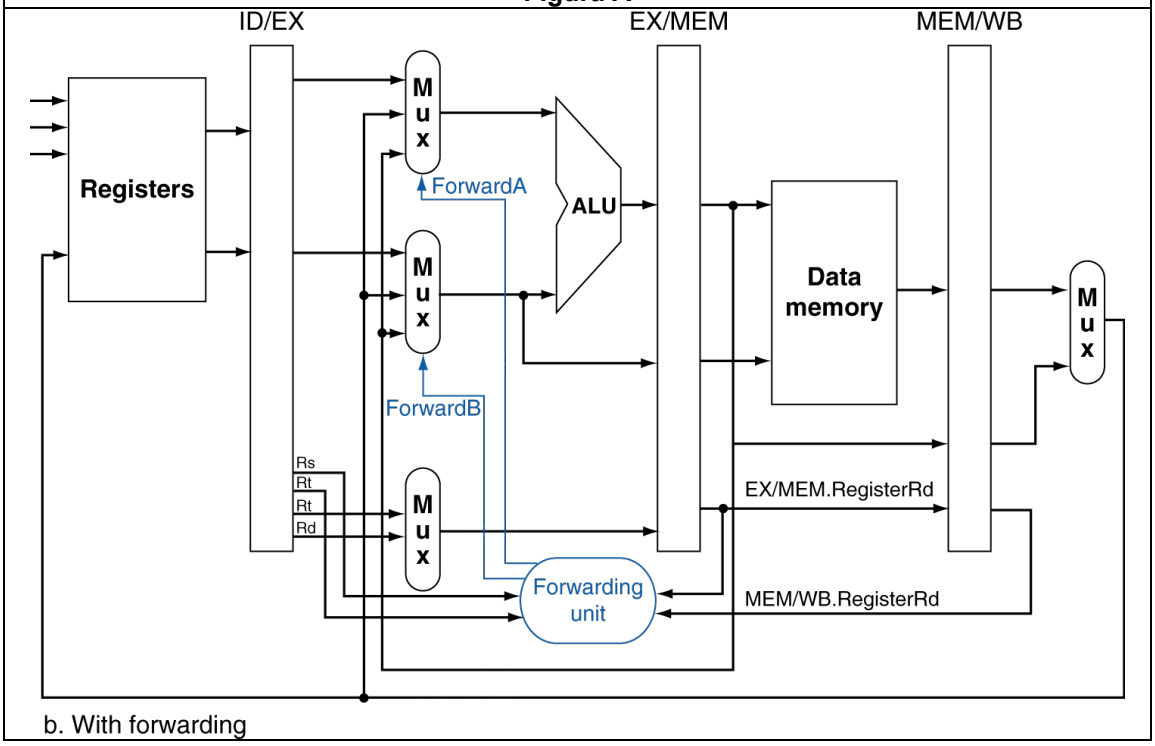
Instrucciones		**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
ADD	R1, R2, R3	IF	ID	EX	M	W																			
LOAD	R4, 0(R1)		IF	(ID)	(ID)	ID	EX	M	W																
MUL	R5, R4, R4			(IF)	(IF)	IF	(ID)	(ID)	ID	EX	M	W													
BNE	R1, R4, L1						(IF)	(IF)	IF	ID	EX	M	W												
SUB	R5, R5, R2									IF	(ID)	ID	-												
ADD	R7, R4, R5										(IF)	IF	-												
L1: ADD	R6, R1, R5												IF	ID	EX	M	W								
OR	R1, R7, R8													IF	ID	EX	M	W							
ST	R6, 0(R1)														IF	(ID)	(ID)	ID	EX	M	W				

b) Suponiendo que hay adelantamiento de datos implementado de la mejor forma posible y que los saltos condicionales se predicen como efectivos, rellenar el siguiente cronograma. Indique sobre el mismo los adelantamientos de datos realizados (flecha de la unidad que genera el dato a la que lo utiliza).

Instrucciones		**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
ADD	R1, R2, R3	IF	ID	EX	M	W																
LOAD	R4, 0(R1)		IF	(ID)	EX	**M**	W															
MUL	R5, R4, R4			IF	(ID)	ID	**EX**	M	W													
BNE	R1, R4, L1				(IF)	IF	ID	EX	M	W												

P2.14. Para realizar adelantamiento de datos en el procesador descrito durante el curso se diseñan caminos de adelantamiento de acuerdo al esquema de la figura A:

Figura A



En la unidad de adelantamiento (Forwarding Unit), entre otros, está implementado el adelantamiento correspondiente al siguiente pseudocódigo:

```
if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0
            and EX/MEM.RegisterRd ≠ ID/EX.RegisterRt
            and MEM/WB.RegisterRd = ID/EX.RegisterRt)
    then  ForwardB = 01      /* si cumple condición se elige la segunda entrada del multiplexor */
else ForwardB= 00            /* en caso contrario no se adelanta */
```

Se pide:

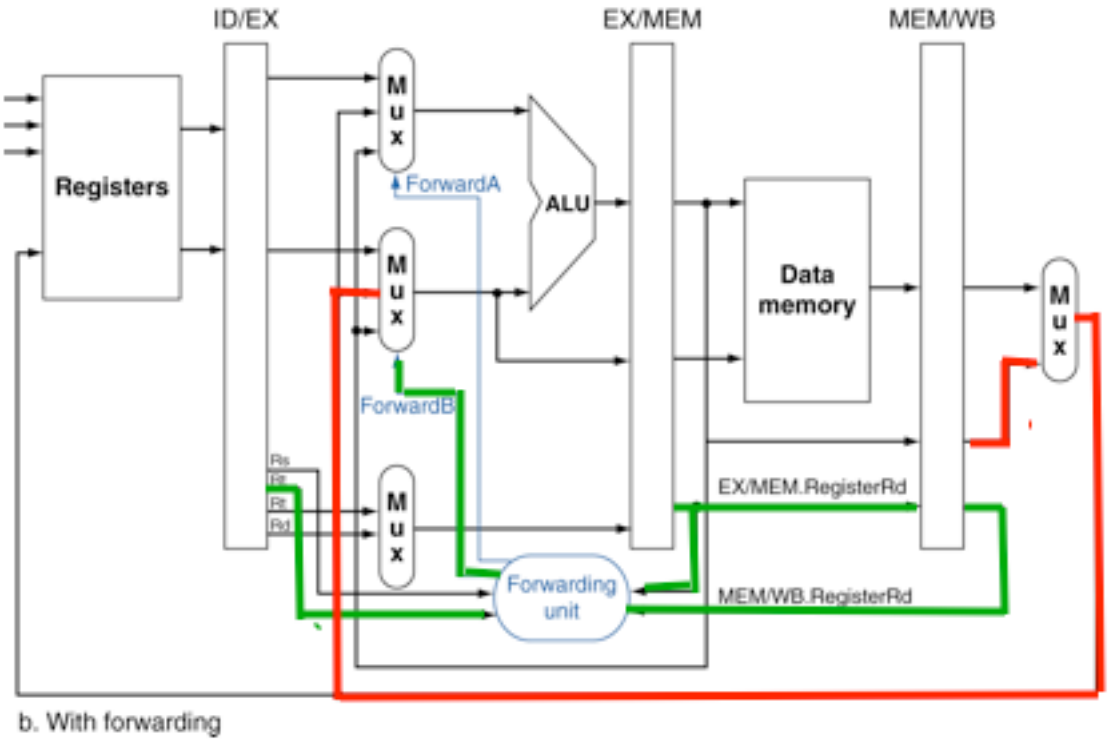
- Dar una secuencia de instrucciones lo más corta posible, en la que sea necesario utilizar el adelantamiento correspondiente al pseudo-código anterior

- Marcar sobre la figura TODAS las líneas que han intervenido para detectar y realizar el

Solución:
I1: ADD r1,r2,r3
I2: SUB r4,r8,r5
I3: XOR r6,r2,r1

Se activa AnticiparB (líneas verdes) para anticipa el resultado de I1 porque el fuente (ID/Ex Rt) de la instrucción (I3) en proceso (etapa EX) coincide con el destino (MEM/WB Rd) de la instrucción previa a la anterior (I1) y se adelanta el resultado guardado en los regs MEM/WB (lineas rojas). Salvo que la instrucción anterior (I2) tenga el mismo registro destino (por ejemplo si I2 fuese SUB r4,r1,r5).

Marcar sobre la figura TODAS las líneas que han intervenido para detectar y realizar el adelantamiento.



b. With forwarding

P2.15. Se tiene un procesador segmentado de cuatro etapas: captura, decodificación, ejecución y escritura. En la etapa de decodificación se calcula la dirección de salto. En la etapa de ejecución se examina la condición de salto. Suponer que la frecuencia para saltos condicionales es del 30% mientras que los incondicionales son del 7%. Suponer también que el porcentaje de condiciones que se satisfacen es del 70%. Se piden ciertos resultados para dos de las estrategias de saltos: predecir efectivo y predecir no efectivo.

a) Considerar la ejecución de todos los casos relevantes (tres) para ambas estrategias que pueden ocurrir dibujando los correspondientes diagramas del procesador, indicando el número de ciclos perdidos en cada caso.
b) Calcular el CPI para ambas estrategias, considerando que los únicos riesgos posibles son los debidos a los problemas de control.

SOLUCIÓN:

a1) Estrategia de predecir no efectivo

1: Salto incondicional	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
CAPTURA	JMP	Next	Target	-	-	-
DECOD.	-	JMP	----	Target	-	-
EJEC.	-	-	JMP	-	Target	-
ESCRIT.	-	-	-	JMP	-	Target

Se produce un retardo de 1 ciclo.

2: Salto condicional NO efectivo	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**
CAPTURA	Bcc	Next	Next+1	-	-	-
DECOD.	-	Bcc	Next	Next+1	-	-
EJEC.	-	-	Bcc	Next	Next+1	-
ESCRIT.	-	-		Bcc	Next	Next+1

Se produce un retardo de 0 ciclos.

3) Salto condicional SÍ efectivo.

3: Salto condicional SÍ efectivo	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**
CAPTURA	Bcc	Next	Next+1	Target	-	-	-
DECOD.	-	Bcc	Next	---	Target	-	-
EJEC.	-	-	Bcc	---	---	Target	-
ESCRIT.	-	-	-	Bcc	-	-	Target

Se produce un retardo de 2 ciclos.

a2) Estrategia de predecir efectivo

1.- Salto incondicional. Igual que en el caso anterior, se produce un retardo de 1 ciclo.

2: Salto condicional NO efectivo	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**
CAPTURA	Bcc	Next	Target	Next+1	-	-	-
DECOD.	-	Bcc	Next	---	Next+1	-	-
EJEC.	-	-	Bcc	Next	---	Next+1	-
ESCRIT.	-	-	-	Bcc	Next	---	Next+1

Se produce un retardo de 1 ciclo.

3: Salto condicional SÍ efectivo	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**
CAPTURA	Bcc	Next	Target	Target+1			

Considerando un CPI ideal de 1 ciclo y que las detenciones son sólo debidas a problemas de control, se obtiene para este ejemplo que la técnica de predecir efectivo es un 8,8% mejor que la de predecir no efectivo (1,49/1,37 = 1,088).

P2.16. Considerar un sistema RISC dotado con arquitectura Harvard y un procesador de 8 segmentos:

F1 Captura-1: Envía la dirección a la caché de instrucciones
F2 Captura-2: Recibe la instrucción desde la caché
D1 Decodifica-1: Se obtiene la dirección de saltos
D2 Decodifica-2: Lectura de registros internos. Calcula la condición en saltos
EX Ejecución: Ejecuta las operaciones aritméticas. Calcula la dirección efectiva en instrucciones Load y Store
MEM1 Memoria-1: Envía la dirección a la caché de datos
MEM2 Memoria-2: Lee/Escribe el dato en la caché
WR Escritura: Escribe el resultado en el registro interno que corresponda

a) Suponer la posibilidad de mejorar el sistema al incorporar caminos para el adelantamiento de datos "*Internal Forwarding*", señalar cuántos son posibles y entre qué segmentos.
En el caso de encontrase un riesgo de datos, el sistema añade ciclos de retardo o en su caso utiliza uno de los caminos anteriores.

b) Dado el siguiente programa y suponiendo cachés ideales, señalar los riesgos que existen y los ciclos necesarios antes y después de la mejora.

I1: LOAD R2, 8(R4)
I2: ADD R3, R2, R5
I3: SUB R4, R1, R3
I4: STORE 0(R6), R4

Nota1: Todas las instrucciones pasan por todos los segmentos. No es posible leer y escribir simultáneamente de un mismo registro.
Nota2: La sintaxis de las instrucciones es: FUNCION Destino, Fuente1, Fuente2.

c) Dada la siguiente secuencia de código que incluye un salto condicional, indicar cuántos ciclos se ejecutan desde la captura de SUB hasta la ejecución final de STORE. Suponer que el salto es efectivo y que se emplean las estrategias **c1)** Parar el sistema hasta conocer la dirección del salto. **c2)** Predecir no efectivo **c3)** Predecir efectivo y **c4)** Predecir efectivo utilizando un BTB.
Nota3: Suponer que en este caso se aplica la mejora del apartado **a)**

```
            SUB R3, R7, R8
            BEZ   R3, END        ; Salta a END R3 =0, (Z=1)
.           …
END:  STORE 0(R6), R4
```

SOLUCIÓN:

a) Para optimizar el sistema, se deben interconectar todos los segmentos que producen datos (EX y MEM2) con aquellos que los utilizan (D2, EX y MEM2), es decir:

El segmento EX con los segmentos D2, EX y MEM2 y el segmento MEM2 con los segmentos D2, EX y MEM2.

MEM1 No necesita adelantamiento ya que la dirección efectiva se calcula en la etapa anterior EX.

D1 no necesita datos, ya que la dirección efectiva se obtiene con el dato inmediato que acompaña a la instrucción.

b1) Sin adelantamiento de datos

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOAD	F1*	F2*	D1	D2	EX	M1*	M2*	**WR**							
ADD		F1*	F2*	D1	$D2^W$	w	w	w	D2	EX	M1	M2	**WR**		
SUB			F1*	F2*	D1	d	d	d	d	$D2^W$	w	w	w	D2	EX

b2) Con adelantamiento de datos

	1	2	3	4	5	6	7	8	9	10	11	12	13
LOAD	F1*	F2*	D1	D2	EX	M1*	M2*	**WR**					
ADD		F1*	F2*	D1	D2	EX^{W}	w	EX	M1	M2	**WR**		
SUB			F1*	F2*	D1	D2	d	D2	EX	M1	M2	**WR**	
STORE				F1*	F2*	D1	d	D1	D2	EX	M1	M2	**WR**

c1) Con adelantamiento de datos y Parada del sistema

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SUB	F1*	F2*	D1	D2	EX	M1*	M2*	**WR**						
BEZ		F1*	F2*	D1(T)	$D2^{W}$	D2(cc)	EX	M1	M2	**WR**				
NEXT			F1*	F2*	**STALL (F2)**	**STALL (F2)**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**		
NEXT+1				F1*	**STALL (F1)**	**STALL (F1)**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	
STORE							F1*	F2*	D1	D2	EX	M1*	M2*	**WR**

c2) Con adelantamiento de datos y Predicción no efectiva. Captura 3 instrucciones especulativas

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
SUB	F1*	F2*	D1	D2	EX	M1*	M2*	**WR**						
BEZ		F1*	F2*	D1(T)	$D2^{W}$	D2(cc)	EX	M1	M2	**WR**				
NEXT			F1*	F2*	D1	$D1^{D}$	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**			
NEXT+1				F1*	F2*	$F2^{D}$	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**		
NEXT+2					F1*	$F1^{D}$*	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	
STORE							F1*	F2*	D1	D2	EX	M1*	M2*	**WR**

c3) Con adelantamiento de datos y Predicción efectiva

	1	2	3	4	5	6	7	8	9	10	11	12	13
SUB	F1*	F2*	D1	D2	EX	M1*	M2*	**WR**					
BEZ		F1*	F2*	D1(T)	$D2^{W}$	D2(cc)	EX	M1	M2	**WR**			
NEXT			F1*	F2*	D1	$D1^{D}$	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**		
NEXT+1				F1*	F2*	$F2^{D}$	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	**FLUSH**	
STORE					F1*	$F1^{D}$	F2*	D1	D2	EX	M1*	M2*	**WR**

c4) Con adelantamiento de datos y Predicción dinámica con BTB

	1	2	3	4	5	6	7	8	9	10	11
SUB	F1*	F2*	D1	D2	EX	M1*	M2*	**WR**			
BEZ		F1*	F2*	D1(T)	$D2^{W}$	D2(cc)	EX	M1	M2	**WR**	
STORE			F1*	F2*	D1	$D1^{D}$	D2	EX	M1*	M2*	**WR**
Next st				F1*	F2*	$F2^{D}$	D1	D2	EX	M1*	M2*
Next+1 st					F1*	$F1^{D}$	F2	D1	D2	EX	M1*

P2.17. Considerar el siguiente procesador RISC con un CPI ideal de 1,5 segmentado en 9 estados:

- **F1** Captura de instrucción 1. Envía la dirección de la instrucción a la I-caché.
- **F2** Captura de instrucción 2. Lee la instrucción de la I-caché.
- **D1** Decodificación de la instrucción.
- **R1** Lectura de operandos desde registros. Calcula la dirección destino en saltos.
- **A1** Comienza la operación en la ALU. Calcula la dirección efectiva. Resuelve la condición.
- **A2** Termina la operación con la ALU.
- **M1** Captura de datos 1. Envía la dirección efectiva a la D-caché.
- **M2** Captura de datos 2. Lee/escribe el dato de/en la D-caché.
- **W1** Escribe resultado en un registro.

Con un porcentaje de saltos del 20%, de los cuales el 70% son efectivos, y suponiendo que no existen otros riesgos que los de control, se pide: **a)** Determinar el CPI real si la predicción es estática **i)** efectiva o **ii)** no efectiva. En ambos casos, cuando se detecta la instrucción de salto, el procesador se detiene hasta poder ejecutar la predicción elegida.

Suponer ahora una secuencia de saltos de la forma E-E-E-NE-E-E-NE-E-E-E-NE-E-E-NE...(E: Efectivo, NE: No Efectivo). Se pide **b)** Determinar el CPI real si la predicción es dinámica con dos bits de control donde la predicción cambia tras dos fallos consecutivos. Suponer que por defecto se predice efectivo y que la ejecución no se detiene al detector un salto.

c) Determinar el CPI real si se mejora el caso anterior con una estructura BTB.

SOLUCIÓN:

ai) Predicción estática Efectiva (E): Si acierta +3, si falla +2: CPI = 1,5 + 0,2 x (0,7 x 3 + 0,3 x 2) = 2,04

F1	Bcc	N	(N+1)	N+1	T	T+1 N+2								
F2		Bcc	(N)	N	N+1	T ---	T+1 N+2							
D1			Bcc	---	N	--- N+1	T ---	T+1 N+2						
R1(T)				Bcc	---	--- N	--- N+1	T ---	T+1 N+2					
A1(cc)					Bcc	---	--- N	--- N+1	T ---	T+1 N+2				
A2						Bcc	---	--- N	--- N+1	T ---	T+1 N+2			
M1							Bcc	---	--- N	--- N+1	T ---	T+1 N+2		
M2								Bcc	---	--- N	--- N+1	T ---	T+1 N+2	
W1									Bcc	---	--- N	--- N+1	T ---	T+1 N+2

aii) Predicción No Efectiva (NE). Si acierta +0, si falla +4: CPI = 1,5 + 0,2 x (0,7 x 4 + 0,3 x 0) = 2,06

F1	Bcc	N	N+1	N+2	N+3	N+4 T								
F2		Bcc	N	N+1	N+2	N+3 ---	N+4 T							
D1			Bcc	N	N+1	N+2 ---	N+3 ---	N+4 T						
R1(T)				Bcc	N	N+1 ---	N+2 ---	N+3 ---	N+4 T					

b) Predice dinámica: siempre Efectivo, la predicción no cambia.

F1	Bcc	N	N+1	N+2	T	T+1 N+3							
F2		Bcc	N	N+1	N+2	T ---	T+1 N+3						
D1			Bcc	N	N+1	--- N+2	T ---	T+1 N+3					
R1(T)				Bcc	N	--- N+1	--- N+2	T ---	T+1 N+3				
A1(cc)					Bcc	--- N	--- N+1	--- N+2	T ---	T+1 N+3			
A2						Bcc	--- N	--- N+1	--- N+2	T ---	T+1 N+3		
M1							Bcc	--- N	--- N+1	--- N+2	T ---	T+1 N+3	
M2								Bcc	--- N	--- N+1	--- N+2	T ---	T+1 N+3
W1									Bcc	--- N	--- N+1	--- N+2	T ---

Si acierta en la predicción (E) +3, si falla +1: CPI = 1,5 + 0,2 x (5/7 x 3 + 2/7 x 1) = 1,9857

c) Predice siempre Efectiva pero con BTB

F1	Bcc	T	T+1	T+2	T+3	T+4 N						
F2		Bcc	T	T+1	T+2	T+3 ---	T+4 N					
D1			Bcc	T	T+1	T+2 ---	T+3 ---	T+4 N				
R1(T)				Bcc	T	T+1 ---	T+2 ---	T+3 ---	T+4 N			
A1(cc)					Bcc	T ---	T+1 ---	T+2 ---	T+3 ---	T+4 N		
A2						Bcc	T ---	T+1 ---	T+2 ---	T+3 ---	T+4 N	
M1							Bcc	T ---	T+1 ---	T+2 ---	T+3 ---	T+4 N
M2								Bcc	T ---	T+1 ---	T+2 ---	T+3 ---
W1									Bcc	T ---	T+1 ---	T+2 ---

Si acierta en la predicción (E) +0, si falla +4 ciclos: CPI = 1,5 + 0,2 x (5/7 x 0 + 2/7 x 4) = 1,7286

P2.18. Tenemos un procesador RISC segmentado con las siguientes 6 etapas: Captura instrucciones (CI): en donde se lee la instrucción a ejecutar. Decodificación (DE): en donde se decodifica la instrucción. Captura de operandos (CO): en donde se cogen los operandos de los registros (si los hay), se calcula la dirección de memoria en las instrucciones con memoria, o se calcula la dirección de salto en las instrucciones de salto. Ejecución (EJ): en donde se ejecuta la operación aritmética o lógica. Se da la orden de lectura en las instrucciones de carga. También se determina si los saltos son o no efectivos. La fase de ejecución de las instrucciones se realiza en un ciclo para todas las instrucciones excepto las de punto flotante, que necesitan 3 ciclos. En este caso la ALU queda ocupada durante los tres ciclos y se producen las detenciones pertinentes. Memoria (ME): en donde se da la orden de escritura en las instrucciones de almacenamiento o se recibe el dato en las de lectura. Escritura (ES): en donde se escriben los resultados en los registros o se almacena el dato en memoria. Se dispone de los circuitos necesarios que permiten el adelantamiento de datos "*Internal Fordwarding*". El procesador sólo tiene un puerto de memoria. Para una determinada carga de trabajo se tiene la frecuencia de uso de instrucciones que se muestra en la tabla adjunta. Todos los saltos son condicionales y el 57% de ellos resultan efectivos. Tabla de frecuencia de Instrucciones:

Instrucción	Saltos	Enteras	Punto Flotante	Lectura de datos	Escritura de datos
Frecuencia	15%	35%	17%	21%	12%

Para esta carga de trabajo se pide: **a)** ¿Cuál es la CPI de ejecución suponiendo que al detectarse una instrucción de salto se detiene el sistema hasta que esta se resuelve? **b)** ¿Cuál es la CPI de ejecución si hacemos la predicción de no efectivo? **c)** ¿Cuál es la CPI de ejecución si hacemos la predicción de efectivo? **d)** ¿Cuál de las dos predicciones es más rentable y en cuánto? **NOTA:** En el cálculo del CPI no considerar los periodos de llenado y vaciamiento del procesador.

SOLUCIÓN:

1) Las operaciones en coma flotante utilizan el segmento de ejecución durante tres ciclos, es decir por cada instrucción de CF (17%) hay dos ciclos de ejecución adicionales.

2) En las operaciones de lectura (LD) y escritura (ST) de datos en memoria, se utiliza el único bus de direcciones/datos, en segmentos diferentes al de captura de instrucciones. Esto supone una pérdida de dos ciclos en cada caso (ver tablas).

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11
CI	LD*	Next*	N+1*	---	---	N+2*					
DE		LD									
CO			LD								
EJ				LD*							
ME					LD*						
ES						LD	Next	N+1	---	---	N+2

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
CI	ST*	Next*	N+1*	N+2*	---	---	N+3*					
DE		ST										
CO			ST									
EJ				ST								
ME					ST*							

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
CI	Bcc	Next	---	---	(N+1)/Tg					
DE		Bcc			Next					
CO			Bcc(T)							
EJ				Bcc(cc)						
ME					Bcc					
ES						Bcc	---	---	Next	(N+1)/Tg

b) El sistema predice no efectivo. La retención es de 0 ciclos en el caso de condición no efectiva (43%) y de 3 ciclos en el caso de efectiva (57%).

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
CI	Bcc	Next	N+1	N+2	(N+3)/Tg					
DE		Bcc								
CO			Bcc(T)							
EJ				Bcc(cc)						
ME					Bcc					
ES						Bcc	Next	N+1	N+2	(N+3)/Tg

c) El sistema predice efectivo. La retención es de 1 ciclo en el caso de condición no efectiva (43%) y de 2 ciclos en el caso de efectiva (57%).

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
CI	Bcc	Next	N+1	Target	(T+1)/(N+2)					
DE		Bcc								
CO			Bcc(T)							
EJ				Bcc(cc)						
ME					Bcc					
ES						Bcc	Next	N+1	Target	(T+1)/(N+2)

La ecuación que define el rendimiento es:

CPI = %Ent x 1 + %PF x 3 + %(LD+ST) x 3 + %Bcc{%Efec x (1+Pen) + %No_Efec x (1+ Pen)}

En base a los datos anteriores, el CPI para cada uno de los casos es:

a) CPI = 0,35x1 + 0,17x3 + 0,33x 3 + 0,15 {(0,57x4) + (0,43x3)} = 2,39 ciclos.

b) CPI = 0,35x1 + 0,17x3 + 0,33x 3 + 0,15 {(0,57x4) + (0,43x1)} = 2,26 ciclos.

c) CPI = 0,35x1 + 0,17x3 + 0,33x 3 + 0,15 {(0,57x3) + (0,43x2)} = 2,24 ciclos.

d) Comparando los apartados b y c resulta más rentable la predicción efectiva en un ~1% (2,26/2,24=1,01).

P2.19. Se dispone de un procesador segmentado en cinto etapas, captura (CI), decodificación (DI), ejecución en la ALU (EX), ejecución en memoria (ME) y escritura de resultados (WE). La dirección de salto se sabe en (DI) y la condición se resuelve en (EX). En un supuesto sistema ideal (sin riesgos), todas las instrucciones tardan 1 ciclo, excepto las de coma flotante que tardan 3 ciclos. Las únicas perdidas del rendimiento ideal, son debidas a los riesgos de control y se sabe que el porcentaje de saltos condicionales efectivos es del 70%.

TIPO	JMP	Bcc	C.FLOTANTE	OTRAS
% USO	5	20	20	55
CPI_TOTAL (ciclos)1	1	1	3	1

Se pide conocer cuantitativamente, la pérdida de rendimiento frente al sistema ideal, si la estadística de instrucciones es como la que se indica en la tabla superior y se diseñan las siguientes estrategias para el tratamiento de saltos: **a)** Parar el sistema hasta resolver el salto. **b)** Predecir estática no efectiva.
c) Predicción dinámica sin BTB, para la secuencia **NE-E-E-NE-E-E-E-E**, suponiendo dos bits de control que cambia la predicción cada dos fallos y que por defecto comienza prediciendo efectivo.

SOLUCIÓN:

CPI^{IDEAL} = 0,8*1 + 0,2*3 = 1,4

a) Detiene el procesador

CI	JMP	Next	T		Bcc	Next	--	N+1/T				
DI(T)		JMP		T		Bcc	--	N	N+1/T			
EX(cc)			JMP		T		Bcc	--	N	N+1/T		
ME				JMP		T		Bcc	--	N	N+1/T	
WE					JMP	---	T		Bcc	--	N	N+1/T

b) Predice no efectivo

CI	Bcc	Next	N+1	N+2/T				
DI(T)		Bcc	N	N+1	N+2/T			
EX(cc)			Bcc	N	N+1	N+2/T		
ME				Bcc	N	N+1	N+2/T	
WE					Bcc	N	N+1	N+2/T

c) Predice efectivo

CI	Bcc	Next	T	T+1/N+1				
DI(T)		Bcc	N	T	T+1/N+1			
EX(cc)			Bcc	N	T	T+1/N+1		
ME				Bcc	N	T	T+1/N+1	
WE					**Bcc**	N	T	T+1/N+1

Por cada JMP (5%) se pierde 1 ciclo.

Por cada Bcc (20%), si se para se pierden 1 ciclos si NE (30%) y 2 ciclos si E (70%).

Por cada Bcc (20%), si se predice NE se pierden 0 ciclos si se acierta (30%) y 2 ciclos si falla (70%).

Por cada Bcc (20%), si se predice E se pierde 1 ciclo si se acierta (6/8) y 1 ciclo si falla (2/8).

a) $CPI^{RETARDO}$ = 0,05*1 + 0,2*(0,3*1 + 0,7*2) = 0,39

Perdida de rendimiento CPI^{IDEAL} / $CPI^{RETARDO}$ + CPI^{IDEAL} = 1,40 / 1,79 = 0,78. 22% de pérdida

b) $CPI^{RETARDO}$ = 0,05*1 + 0,2*(0,3*0 + 0,7*2) = 0,33

P2.20. Un procesador segmentado de seis etapas, que funciona de la siguiente manera: en la etapa 2 decodifica la instrucción y detecta riesgos. En la en la etapa 3 calcula la dirección destino y la condición de saltos. Además dispone de de un predictor dinámico de saltos BTB (Branch Target Buffer). La probabilidad de que un salto se encuentre en la tabla es del 80 % y de que se acierte en la predicción es del 90%. Para un programa que del total de saltos condicionales tiene un 60% de saltos efectivos en su ejecución. Se piden los ciclos de penalización, ayudándose de un cronograma, en los siguientes casos y considerando tanto la posibilidad de que el salto sea efectivo o que no lo sea:

a) El salto no se encuentra en el BTB.
b) Esta en el BTB y se predice no efectivo
c) Esta en el BTB y se predice efectivo
d) Calcular el CPI promedio de las instrucciones de salto para este programa

SOLUCIÓN:

a) No esta en BTB: Si salto efectivo penaliza 2 ciclos y si NE 1 ciclo

	01	02	03	04	05	06	07	08	09
S1 captura	Bcc	N	-	T/N+1					
S2 dec		Bcc	-	N	T/N+1				
S3 destino y. condición			Bcc	-	N	T/N+1			
S4				Bcc	-	N	T/N+1		
S5					Bcc	-	N	T/N+1	
S6						Bcc	-	N	T/N+1

b) Esta en BTB y predice NO efectivo: Si salto efectivo penaliza 2 ciclos y si NE 0 ciclos

	01	02	03	04	05	06	07	08	09
S1 captura	Bcc	N	N+1	T/N+2					
S2 dec		Bcc	N	N+1	T/N+2				
S3 destino y. condición			Bcc	N	N+1	T/N+2			
S4				Bcc	N	N+1	T/N+2		
S5					Bcc	N	N+1	T/N+2	
S6						Bcc	N	N+1	T/N+2

c) Esta en BTB y predice efectivo: Si salto efectivo penaliza 0 ciclos y si NE 2 ciclos

	01	02	03	04	05	06	07	08	09
S1 captura	Bcc	T	T+1	T+2/N					
S2 dec		Bcc	T	T+1	T+2/N				
S3 destino y. condición			Bcc	T	T+1	T+2/N			
S4				Bcc	T	T+1	T+2/N		
S5					Bcc	T	T+1	T+2/N	

P2.21. Considere un procesador RISC, segmentado en 8 etapas con cauce único de ejecución. El CPI sin tener en cuenta los saltos es de 1,2 ciclos. Las etapas del pipeline son:

F1 Captura de instrucción 1. Envía la dirección de la instrucción a la I-caché.
F2 Captura de instrucción 2. Lee la instrucción de la I-caché.
DE Decodificación de la instrucción. Calcula la dirección destino en saltos.
RE Lectura de operandos desde registros.
EX Operación en la ALU. Calcula la dirección efectiva para Mem. Resuelve la condición.
M1 Captura de datos 1. Envía la dirección de la instrucción a la D-caché.
M2 Captura de datos 2. Lee/escribe el dato de/en la D-caché.
WR Escribe resultado en un registro.

Para un programa determinado con un porcentaje de saltos condicionales del 15%, de los cuales el 75% son efectivos, calcular suponiendo que no existen otros riesgos que los de control:

a) El **CPI real** si la predicción es estática **i)** efectiva o **ii)** no efectiva. En ambos casos, cuando se detecta la instrucción de salto, el procesador no se detiene. Para justificar su respuesta utilice las tablas adjuntas.

b) El CPI real si la predicción es dinámica con dos bits de control donde la predicción cambia tras dos fallos consecutivos. Suponer que por defecto se predice no efectivo.

La secuencia de saltos es de la forma E-E-E-E-NE-E-E-NE-E-E-E-NE-E-NE-E-NE-NE...(E: efectivo, NE: no efectivo)

c) Suponga el siguiente trozo de código, y que el procesador no dispone de hardware para el adelantamiento de datos. Tenga en cuenta además que todas las etapas funcionan con flanco de subida, es decir en el mismo ciclo que se escribe un registro dado (etapa WR), éste no puede ser leído por otra instrucción en la etapa RE.

c.1) Reescriba el código agregando las instrucciones NOP necesarias y reordenando el código para que se ejecute en el menor tiempo posible. Cuál sería el CPI para este código.

c.2) Si existiese adelantamiento de datos y respetando la emisión en orden. ¿Cuántos ciclos tardaría en ejecutarse este trozo de código?

```
I1:  ADD R3, R1, R2      ;R3 <= R1+R2          I5:  OR R1, R1, R6      ;R1 <= R1 or R6
I2:  LD R1, 100(R6)      ;R1 <= M[100+R6]      I6:  ST R1, 4(R4)       ;M[4+R4] <= R1
I3:  AND R5, R3, R4      ;R5 <= R3 and R4      I7:  ST R3, 104(R4)     ;M[104+R4] <= R3
I4:  NAND R6, R4, R2     ;R6 <= R4 nand R2     I8:  SUB R3, R5, R6     ;R3 <= R5 - R6
```

SOLUCIÓN:

a) Predicción estática efectiva/salto no efectivo:

	T1	T2	T3	T4	T5	T6	T7	T8	T9
Bcc	F1	F2	DE	RE	EX	M1	M2	WR	
N		F1	F2	DE	RE	EX	M1	M2	WR
N+1			F1	F2	DE	RE	EX	M1	M2
N+2				-	-	F1	F2	DE	RE
…	…	…	…	…	…	…	…	…	…
T				F1	F2	-	-	-	-
T+1					F1	-	-	-	-

Predicción estática efectiva/salto efectivo:

	T1	T2	T3	T4	T5	T6	T7	T8	T9
Bcc	F1	F2	DE	RE	EX	M1	M2	WR	
N		F1	F2	DE	RE	-	-		
N+1			F1	F2	DE	-	-		
…	…	…	…	…	…	…	…	…	…

Predicción estática no efectiva/salto no efectivo:

	T1	T2	T3	T4	T5	T6	T7	T8	T9
BCC	F1	F2	DE	RE	EX	M1	M2	WR	
N		F1	F2	DE	RE	EX	M1	M2	WR
N+1			F1	F2	DE	RE	EX	M1	M2
N+2				F1	F2	DE	RE	EX	M1
N+3					F1	F2	DE	RE	EX
…	…	**…**	**…**	**…**	**…**	**…**	**…**	**…**	**…**
T									

Predicción estática no efectiva/salto efectivo:

	T1	T2	T3	T4	T5	T6	T7	T8	T9
BCC	F1	F2	DE	RE	EX	M1	M2	WR	
N		F1	F2	DE	RE	-			
N+1			F1	F2	DE	-			
N+2				F1	F2	-			
N+3					F1	-			
…	…	…	…	…	…	…	…	…	…
T						F1	F2	DE	RE
T+1							F1	F2	DE

Total CPI agregado con la estrategia de predicción No efectiva:

Si salto no efectivo agrega 0 ciclos, si salto efectivo agrega 4 ciclos. Luego CPI adicional:

$CPI^{SALTOS} = 0,15*(0,25 * 0 + 0,75 * 4) = 0,45$ ciclos

$CPI = CPI^{Sin\ SALTOS} + CPI^{SALTOS} = 1,2 + 0,45 = 1,65$ ciclos

b)

Saltos:	E	E	E	E	NE	E	E	NE	E	E	E	NE	E	NE	E	NE	NE	-
Predicc:	NEf	NEd	EFf	EFf	EFf	EFd	EFf	EFf	EFd	EFf	EFf	EFf	EFd	EFf	EFd	EFf	EFd	NEf
Acieto/Fallo	F	F	A	A	F	A	A	F	A	A	A	F	A	F	A	F	F	-
Recuento	1	2	1	2	1	3	4	2	5	6	7	3	8	4	9	5	6	

Hay 2 predicciones NE con salto E que penaliza 4 ciclos. Luego con predicción efectiva hay 6/17 fallos y 9/17 aciertos que penalizan 2 ciclos.

$CPI^{SALTOS} = 0,15*(2/17*4 + 6/17*2 + 9/17*2) = 0,335$

$CPI = CPI^{Sin\ SALTOS\ s} + CPI^{SALTOS} = 1,2 + 0,335 = 1,535$ ciclos

c1)

I1:	ADD R3, R1, R2
I2:	LD R1, 100(R6)
I3:	NAND R6, R4, R2
I4:	nop
I5:	nop
I6:	AND R5, R3, R4
I7:	ST R3, 104(R4)
I8:	OR R1, R1, R6

I9:	nop
I10:	nop
I11:	SUB R3, R5, R6
I12:	nop
I13:	ST R1, 4(R4)
I14:	
I15:	
I16:	

Total ciclos en ejecutarse el código: Nº Inst + profPipe - 1 = 13 + 8 – 1 = 20

Esta respuesta no es necesaria. Sólo se agrega para aclarar el resultado.

		T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
I1	ADD R3, R1, R2	F1	F2	DE	RE	EX	M1	M2	WR[1]				
I2	LD R1, 100(R6)		F1	F2	DE	M1	EX	M1	M2	WR[2]			
I3	NAND R6,R4,R2			F1	F2	DE	M1	EX	M1	M2	WR[2,3]		
I4	Nop				F1	-	-	-	-	-	-	-	
I5	Nop					F1	-	-	-	-	-	-	-
I6	AND R5, R3, R4						F1	F2	DE	RE[1]	EX	M1	M2
I7	ST R3, 104(R4)							F1	F2	DE	RE	EX	M1
I8	OR R1, R1, R6								F1	F2	DE	RE[2]	EX
I9	Nop									F1	-	-	-
I10	Nop										F1	-	-
I11	SUB R3, R5, R6											F1	F2
I12	Nop												F1
I13	ST R1, 4(R4)												

		T12	T13	T14	T15	T16	T17	T18	T19	T20
I1	ADD R3, R1, R2									
I2	LD R1, 100(R6)									
I3	NAND R6,R4,R2									
I4	Nop									
I5	Nop	-								
I6	AND R5, R3, R4	M2	WR[3]							
I7	ST R3, 104(R4)	M1	M2	WR						
I8	OR R1, R1, R6	EX	M1	M2	WR[4]					
I9	Nop	-	-	-	-	-				
I10	Nop	-	-	-	-	-	-			
I11	SUB R3, R5, R6	F2	DE	RE[3]	EX	M1	M2	WR		
I12	Nop	F1	-	-	-	-	-	-	-	
I13	ST R1, 4(R4)		F1	F2	DE	RE[4]	EX	M1	M2	WR

Notas:

1. Al siguiente ciclo de la escritura del reg R3 por I1, puedo leerlo en la I6

2. LD escribe R1 en T9 y NAND R6 en T10. Luego OR lo puede leer en el siguiente ciclo (T10)

3. La instrucción SUB debe esperar la escritura de R5 y R6. Al R5 lo escribe I6 en T13.

4. ST R1, se puede ejecutar después que OR de la I8 haya escrito el R1 en T15.

I1, I2 e I3: se puede ejecutar ya que no depende de ninguna otra instrucción.

I7: puedo ejecutar ST R3, 104(R4) ya que R3 está disponible desde la ejec de I1 (desde T9). Se puede

P2.22. Se tiene un sistema RISC segmentado con 5 etapas, en donde el salto se detecta en la segunda y la condición se averigua en la tercera. Se ejecuta la siguiente secuencia de saltos condicionales E-E-E-NE-E-E-NE-NE-NE-E-E-NE, donde E significa salto efectivo y NE no efectivo. Se pide:

a) Los ciclos que se pierden por riesgos de control en esta secuencia, suponiendo que el sistema se detiene hasta conocer la condición.

b) La mejora porcentual en la pérdida de ciclos por riesgos de control si se dispone de un BTB con predicción histórica. La predicción cambia tras dos fallos consecutivos y por defecto es efectiva.

SOLUCIÓN:

a) Detiene el sistema: Pierde 1 ciclo si no efectivo (5/12) y dos si efectivo (7/12).

	Bcc	N	--	N+1/T					
Decodifica		Bcc	--	N	N+1/T				
Condición			Bcc	--	N	N+1/T			
				Bcc	--	N	N+1/T		
					Bcc	--	N	N+1/T	

Penalización en ciclos: 5*1 + 7*2 =19 ciclos.

b1) Predicción BTB Efectiva: Pierde 0 ciclos si acierta y 2 ciclos si falla

	Bcc	T	T+1	N/T+2					
Decodifica		Bcc	T	T+1	N/T+2				
Condición			Bcc	T	T+1	N/T+2			
				Bcc	T	T+1	N/T+2		
					Bcc	T	T+1	N/T+2	

b2) Predicción BTB No Efectiva: Pierde 0 ciclos si acierta y 2 ciclos si falla

	Bcc	N	N+1	N+2/T					
Decodifica		Bcc	N	N+1	N+2/T				
Condición			Bcc	N	N+1	N+2/T			
				Bcc	N	N+1	N+2/T		
					Bcc	N	N+1	N+2/T	

Secuencia	E	E	E	NE	E	E	NE	NE	NE	E	E	NE
Predicción	E	E	E	E	E	E	E	E	NE	NE	NE	E
A/F	A	A	A	F	A	A	F	F	A	F	F	F

Penalización en ciclos: 6*0 + 6*2 =12 ciclos.

Mejora 19/12 = 1,583 ⇒ Con la mejora se pierden menos ciclos con un porcentaje del 58,3%

P2.23. Se dispone de un procesador RISC, con un CPI ideal de 1,6 ciclos (si se tiene en cuenta los fallos de caché y TLB), segmentado en las siguientes 10 etapas:

F1: Envío de dirección a la I-caché
F2: Recepción de la instrucción desde la I-caché
D1: Decodificación
RE: Lectura de operandos desde registros internos. Cálculo de dirección destino en saltos
A1: Comienza la operación en la ALU. Calcula la dirección efectiva para acceso a memoria.
A2: Ciclo intermedio de ejecución en operación con la ALU.
A3: Termina la operación con la ALU. Resuelve la condición.
M1: Envía la dirección a la D-caché
M2: Lee/Escribe el dato de/en la D-caché
WR: Escritura de resultados en registro

El procesador posee una instrucción de salto incondicional **JUMP** y cuatro instrucciones de saltos condicionales **JUMPZ**, **JUMPNZ**, **JUMPC**, **JUMPNC** que saltan en función de los flags de Zero y Carry respectivamente. De forma genérica llamaremos **JUMPX** a los saltos condicionales. Los flags son modificados por todas las instrucciones aritméticas y lógicas.
Considerar el siguiente programa en ensamblador que ha surgido de compilar el programa escrito en C que se muestra a continuación. El programa calcula cuántos números entre 1 y 1000 no son múltiplos de 7:

Código C			Ensamblador	
	I1		load R1, 1	*; R1 <= 1 (indice i)*
	I2		xor R2, R2, R2	*; R2 <= 0 (var nm)*
for (i=1;i<1001; i++)	I3	L1:	Mod R3, R1, 7	*; R3 <= R1 mod 7*
{ if (i%7 != 0)	I4		jumpZ L2:	*; salta si 0*
nm ++;	I5		add R2, R2, 1	*; R2 <= R2+1*
};	I6	L2:	add R1, R1, 1	*; R1 <= R1+1*
	I7		sub R3, R1, #1000	*; 1000 decimal*
	I8		jumpNC L1	*; salta por no carry*
	I9		Stop	

Teniendo en cuenta únicamente los riesgos debidos a los saltos: se piden ciertos cálculos para el control de saltos en este programa usando las siguientes estrategias:
A) Sin predicción de saltos, el procesador se detiene a que se resuelva el salto.
B) Predicción estática: si el salto es hacia delante, se predice no efectivo y si es hacia detrás se predice efectivo.
C) Utilizando un predictor dinámico de saltos basado en 2 bits de control, es decir cambia la predicción cada dos errores de predicción consecutivos, tomando por defecto la opción NO efectiva cuando es hacia adelante y Efectivo cuando el salto hacia atrás (cada instrucción de salto tiene su propio predictor dinámico).
NOTAS:
- No considere los ciclos de llenado del pipeline
- Complete las respuestas i, ii, iii, iv en los espacios reservados a tal efecto
- Para justificar los ciclos de detención utilice las tablas adjuntas.

SOLUCIÓN
El programa contiene 2 instrucciones de salto condicional (jumpZ y jumpNC) y ningún salto incondicional. La instrucción jumpZ (I4) se ejecuta 1000 veces de las cuales es efectivo en 142 y 858 veces es no efectivo. La instrucción jumpNC (I8) se ejecuta igualmente 1000 veces, con 999 saltos efectivos y un salto no efectivo.

i) Número total de instrucciones ejecutadas:
2 + 1000*2 + 858 + 1000*3 + 1 = 5861

iii) Detenciones debidas a la estrategia B

Saltos_Ad_Ef * Dem_Pred_NE_Ef + Saltos_Ad_NEf * Dem_Pred_NE_NE +
Saltos_Atr_Ef * Dem_Pred_E_Ef + Saltos_Atr_NEf * Dem_Pred_E_NE
142 * 6 + 858 * 0 + 999 * 3 + 1* 3 = 3852

iv) Detenciones debido a la estrategia C
Para el primer salto (I4) siempre predecirá NO efectivo, ya que solo una de cada 7 veces será efectiva y no hará cambiar la predicción.
Para el segundo Salto (I8) acertará la predicción Efectiva 999 veces y solo fallará en la última.
El resultado numérico es igual al punto iii

TABLAS EXPLICATIVAS

Predicción Efectiva/salto NO Efectivo; Se pierden **3** ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE[1]	A1	A2	A3[2]	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	A3	M1	M2	WR	
N+1			F1	F2	D1	RE	A1	A2	A3	M1	M2	WR
N+2				F1	F2	D1	RE	A1	A2	A3	M1	M2
N+3								F1	F2	D1	RE	A1
…	…	…	…	…	…	…	…	…	…	…	…	…
T					F1	F2	D1	-	-	-	-	-
T+1						F1	F2	-	-	-	-	-
T+2							F1	-	-	-	-	-

Predicción Efectiva/salto Efectivo; Se pierden **3** ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE[1]	A1	A2	A3[2]	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	-	-	-	-	-
N+1			F1	F2	D1	RE	A1	-	-	-	-	-
N+2				F1	F2	D1	RE	-	-	-	-	-
N+3												
…	…	…	…	…	…	…	…	…	…	…	…	…
T					F1	F2	D1	RE	A1	A2	A3	M1
T+1						F1	F2	D1	RE	A1	A2	A3
T+2							F1	F2	D1	RE	A1	A2

Predicción NO Efectiva/salto NO efectivo; Se pierden **0** ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE[1]	A1	A2	A3[2]	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	A3	M1	M2	WR	
N+1			F1	F2	D1	RE	A1	A2	A3	M1	M2	WR
N+2				F1	F2	D1	RE	A1	A2	A3	M1	M2
N+3					F1	F2	D1	RE	A1	A2	A3	M1
N+4						F1	F2	D1	RE	A1	A2	A3
N+5							F1	F2	D1	RE	A1	A2
N+6								F1	F2	D1	RE	A1
…	…	…	…	…	…	…	…	…	…	…	…	…
T												

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1	RE[1]	A1	A2	A3[2]	M1	M2	WR		
N		F1	F2	D1	RE	A1	A2	-	-	-	-	-
N+1			F1	F2	D1	RE	A1	-	-	-	-	-
N+2				F1	F2	D1	RE	-	-	-	-	-
N+3					F1	F2	D1	-	-	-	-	-
N+4						F1	F2	-	-	-	-	-
N+5							F1	-	-	-	-	-
…	…	…	…	…	…	…	…	…	…	…	…	…
T								F1	F2	D1	RE	A1
T+1									F1	F2	D1	RE
T+2										F1	F2	D1

Sin Predicción y Detención / salto NO Efectivo; Se pierden **4** ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1[0]	RE[1]	A1	A2	A3[2]	M1	M2	WR		
N		F1	F2	F2[D]	F2[D]	F2[D]	F2[D]	D1	RE	A1	A2	A3
N+1			F1	F1[D]	F1[D]	F1[D]	F1[D]	F2	D1	RE	A1	A2
N+2								F1	F2	D1	RE	A1
N+3									F1	F2	D1	RE
…	…	…	…	…	…	…	…	…				…
T												
T+1												
T+2												

Sin Predicción y Detención / salto Efectivo; Se pierden **6** ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
jumpX	F1	F2	D1[0]	RE[1]	A1	A2	A3[2]	M1	M2	WR		
N		F1	F2	F2[D]	F2[D]	F2[D]	F2[D]	-	-	-	-	-
N+1			F1	F1[D]	F1[D]	F1[D]	F1[D]	-	-	-	-	-
N+2												
N+3												
…	…	…	…	…	…	…	…	…	…	…	…	…
T								F1	F2	D1	RE	A1
T+1									F1	F2	D1	RE
T+2										F1	F2	D1

0. Decodifica la instrucción. Determina que se trata de un salto.
1. Obtiene la dirección de salto. En el siguiente ciclo puede cargar esa dirección.
2. Resuelve la condición. Descarga el pipeline si no acertó la predicción.

Si bien este programa no posee saltos incondicionales se presentan los ciclos de demora correspondientes

Ciclos de demora en saltos incondicionales; Se pierden 3 ciclos

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
JUMP	F1	F2	D1	RE1	A1	A2	A3	M1	M2	WR		
N		F1	F2	-	-	-	-	-	-	-	-	-
N+1			F1	-	-	-	-	-	-	-	-	-
N+2				-	-	-	-	-	-	-	-	-

P2.24. Se dispone de procesador RISC de 32 bits (palabras, direcciones de memoria y ruta de datos de 32 bits) segmentado con 5 etapas (IF, OF, EX, MEM, WB). IF: Captura de instrucción (instruction fetching); OF: captura de operandos (operand fetching); EX: ejecución de la instrucción, MEM: acceso a memoria de datos, WB: escritura en registros.

Notas: El procesador guarda muchas similitudes con el descrito en la teoría.

a) Calcule la demora media en ciclos que agregan los saltos, sabiendo que estos representan el 12% de las instrucciones, que la predicción es no efectiva y que la dirección de destino y condición de saltos se evalúan y conocen en el ciclo EX. Suponga además que el 30% de los casos el salto es efectivo. Los saltos incondicionales representan el 2% de las instrucciones. Justifique su respuesta utilizando las tablas:

Salto no efectivo. Total ciclos perdidos: **0 .**

Instr/Ciclo	1	2	3	4	5	6	7	8	9
BNE	IF	OF	EX(1)	MEM	WB				
N		IF	OF	EX	MEM	WB			
N+1			IF	OF	EX	MEM	WB		
N+2				IF	OF	EX	MEM	WB	

Salto efectivo. Total ciclos perdidos: **2 .**

Instr/Ciclo	1	2	3	4	5	6	7	8	9
BNE	IF	OF	EX(1)	MEM	WB				
N		IF	OF	-					
N+1			IF	-					
T				IF	OF	EX	MEM	WB	
T+1					IF	OF	EX	MEM	WB

(1) En el ciclo EX del Branch se conoce dirección de salto y condición

Demora media agregada por los saltos medida en ciclos:

CPIsaltos = 12%cond * (30%ef * 2 ciclos + 70%noEf * 0 ciclos) + 2% incon * 2 ciclos=
= 0,12 * 0,3 * 2 + 0,02 * 2 = 0,112 ciclos.

b) Para minimizar la demora en los saltos, este procesador incorpora "saltos con *delay slot*" (*branches with delayed slot*). En la técnica de "*Delay Slot",* el compilador modifica el código de modo que la instrucción que figura a continuación del salto siempre se ejecuta con independencia de que el salto sea efectivo o no. El mnemónico de estos saltos agrega una D, así el BNE se transforma en BNED. Esta técnica también se aplica a los saltos incondicionales.
¿Qué CPI agregan los saltos con esta técnica? Suponga que siempre se puede efectuar saltos con "*Delay Slot"*. Utilice los datos del apartado anterior.

c) El siguiente código C se compila para este procesador utilizando la técnica de "*Delay Slot*" descrita anteriormente. Se muestran el código C, el ensamblador sin las inicializaciones de R1(índice i), R2(fact) y R4 (valor de N).

Codigo C	*Ensamblador*	*Comentarios*	
for (i = 1; i < N; i++)	L1: Mul R2, R2, R1	; fact = fact*i ;	R2 = fact
fact = fact * i;	Sub R5, R1, R4	; R5 = R1 – R4 = (i – N)	R5 si I < N
	BLTD R5, L1	; si R5 < 0 salta a L1	Salto con Delay Slot
	Add R1, R1, 1	; R1 = R1 +1 = i++	R1 = i
	Store R2, dirMem	; guarda result	

Muestre la ejecución de dos iteraciones del algoritmo en la tabla que representa las etapas de *pipeline* del procesador.
Nota: Para la resolución, considere que el sistema se ha resuelto sin penalización cualquier riesgo de datos que pueda aparecer.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Mult	IF	OF	EX	MEM	WB									
Sub		IF	OF	EX	MEM	WB								
BLTD			IF	OF	EX	MEM	WB							
Add				IF	OF	EX	MEM	WB						
Store					IF	-	-							
Mult						IF	OF	EX	MEM	WB				
Sub							IF	OF	EX	MEM	WB			
BLTD								IF	OF	EX	MEM	WB		
Add									IF	OF	EX	MEM	WB	
Store										IF	-	-		
Mult											IF	OF	EX	MEM

Cuántos ciclos se requieren para la ejecución de este bucle. Expréselo en términos de N y suponga que los accesos a memoria de instrucciones y datos son de un único ciclo.

Nº ciclos = Nº iteraciones * (instrucc. + "descarga del pipeline") + (llenado/vaciado del pipeline) =
= (N-1) * 5 + 4 = 5*N -1 ciclos.

Si no se considera el llenado del pipeline:
Nº ciclos = Nº iteraciones * (instrucc. + retardo/iteración) + (última (store)) = (N-1) * 5 ciclos.

Notas:
- Cada iteración requiere 5 ciclos. 4 instrucciones más la siguiente instrucción (store) que se quita del pipeline cuando realiza el salto.
- La última instrucción (store) estará cargada en caso que el salto no sea efectivo (cuando i = N). Es decir el último ciclo tarda también 5 ciclos.
- El llenado o vaciado del pipeline requiere M-1 ciclos, siendo M el número de etapas.

P2.25. Un sistema *RISC* con arquitectura *Harvard* segmenta la ejecución de cada una de las instrucciones en las siguientes etapas: **F** (captura de instrucciones), **D** (decodificación, detección de riesgos y lectura de registros), **E** (opera en la ALU, calcula la dirección efectiva, evalúa la condición en saltos y actualiza en su caso el PC), **M** (acceso a memoria de datos) y **W** (escritura en el banco de registros).
El banco de registros permite actualizar un registro en el mismo ciclo que sea necesario leerlo. El procesador dispone de adelantamiento de datos lo más efectivo posible, y utiliza predicción estática efectiva si el salto es hacia adelante y No Efectiva si el sato es hacia atrás. En ambos casos, al predecir no se detiene el pipeline y será al evaluar el salto cuando se corrige el trabajo realizado incorrectamente.

Suponga que se ejecuta el siguiente fragmento de código.

I1: Loop:	SUB	R1, R1, #4	; R1 = R1-4
I2:	LD	R2, $1000(R1)	; R2 = Mem (R1 + 1000)
I3:	ADD	R3, R3, R2	; R3 = R3 + R2
I4:	LD	R4, $2000(R1)	; R4 = Mem(R1+2000)
I5:	ADD	R2, R2, R4	; R2= R2+R4
I6:	SUB	R2, R2, R5	; R2 = R2 –R5
I7:	ST	$1000(R1), R2	; Mem (R1+1000) = R2
I8:	BNZ	R1,R0, Loop	; salta a Loop si R1 es distinto de cero.
I9:	XOR	R6, R7.R8	; R6 = R7 xor R8
I10:	….		

a) Enumere todos los riesgos posibles en el código anterior, si se desconoce la arquitectura del sistema que lo va a ejecutar.

b) ¿Cómo se tiene que implementar el banco de registros para que permita actualizar un registro en el mismo ciclo que sea necesario leerlo?

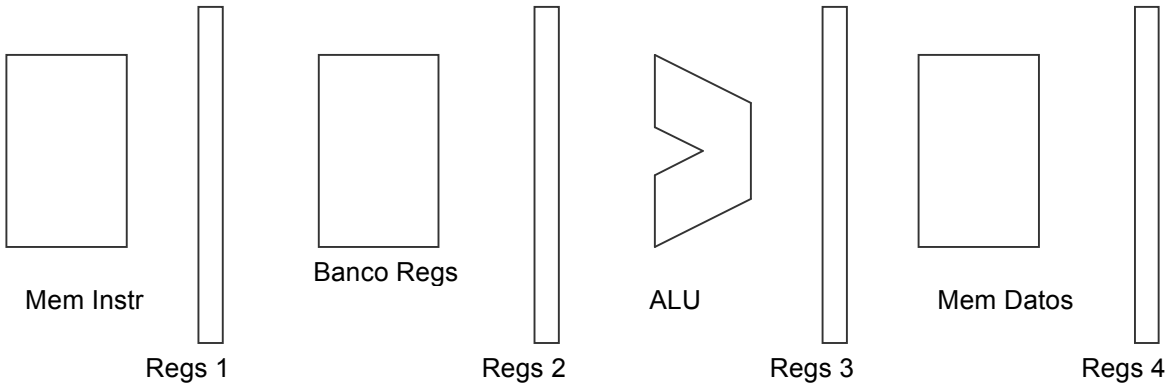
c) Si antes de empezar la ejecución de I1 el registro R1 contiene el valor 8, complete la tabla (ya se ha rellenado la ejecución de la primera instrucción) hasta la finalización de 10 instrucciones en el código anterior, indicando los adelantamientos de datos realizados.

Instrucción \ tiempo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
I1: Loop: SUB R1, R1, #4	F	D	E	M	W															
I2: LD R2, $1000(R1)																				
I3: ADD R3, R3, R2																				
I4: LD R4, $2000(R1)																				
I5: ADD R2, R2, R4																				
I6: SUB R2, R2, R5																				
I7: ST $1000(R1), R2																				
I8: BNZ R1,R0, Loop																				

ciclos adicionales han sido necesarios respecto a la ejecución ideal.

f) ¿Cuándo se captura la instrucción I9? ¿Cuándo se ejecuta?

g) Complete el esquema siguiente con los componentes y conexiones necesarias para que el procesador realice los adelantamientos que son imprescindibles en la ejecución del código anterior, indicando cual se aplica en cada caso.



h) ¿Cómo se modifica el funcionamiento si la arquitectura es Von Neumann?

Instrucción \ tiempo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
I1: Loop: SUB R1, R1, #4	F	D	E	M	W															
I2: LD R2, $1000(R1)																				
I3: ADD R3, R3, R2																				
I4: LD R4, $2000(R1)																				
I5: ADD R2, R2, R4																				
I6: SUB R2, R2, R5																				
I7: ST $1000(R1), R2																				
I8: BNZ R1,R0, Loop																				
I9: XOR R6, R7.R8																				
I10: …																				

ARQUITECTURA DE COMPUTADORES
CAPÍTULO 2. PROCESADORES SEGMENTADOS

SOLUCION

a) Enumere todos los riesgos posibles en el siguiente código si se desconoce la arquitectura del sistema que lo va a ejecutar.

Riesgos de Control : En I8 se salta a I1 o se continua con I9
En Arquitecturas Von Neuman riesgo estructural en acceso a memoria de I2,I4 e I7 (LD/ST) con la instrucción que se esté capturando.
Riesgos de Datos:
RAW : I2 con I1 por R1
RAW : I3 con I2 por R2
RAW : I4 con I1 por R1
RAW : I5 con I2 por R2
RAW : I5 con I4 por R4
RAW : I6 con I5 por R2
RAW: I7 con I1 por R1
RAW : I7 con I6 por R2
RAW : I8 con I1 por R1

WAR : I7 con I2 por acceso a la misma posición de memoria
WAR : I5 con I3 por R2
WAR : I6 con I5 por R2
WAR : I6 con I3 por R2

WAW : I5 con I2 por R2
WAW: I6 con I5 por R2
WAW: I6 con I2 por R2

b) ¿Cómo se tiene que implementar el banco de registros para que permita actualizar un registro en el mismo ciclo que sea necesario leerlo?

Lectura combinacional y escritura síncrona en la mitad del ciclo

c) Si antes de empezar la ejecución de I1 el registro R1 contiene el valor 8, complete la tabla (ya se ha rellenado la ejecución de la primera instrucción) hasta la finalización de 10 instrucciones en el código anterior, indicando con flechas los adelantamientos de datos realizados.

Instrucción \ tiempo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
I1: Loop: SUB R1, R1, #4	F	D	E (1)	M	W								F	D	E	M	W			
I2: LD R2, $1000(R1)		F	D	E (1)	M (2)	W								F	D	E	M	W		
I3: ADD R3, R3, R2			F	D	(d)	E (2)	M	W												
I4: LD R4, $2000(R1)				F	f	D	E	M (2)	W											
I5: ADD R2, R2, R4						F	D	d	E (2)(1)	M	W									
I6: SUB R2, R2, R5							F	f	D	E	M	W								
I7: ST $1000(R1), R2									F	D	E	M (3)	W							
I8: BNZ R1,R0, Loop										F	D	E	M	W						

d) ¿Cuántos ciclos tarda en finalizar las 10 instrucciones
18 ciclos.

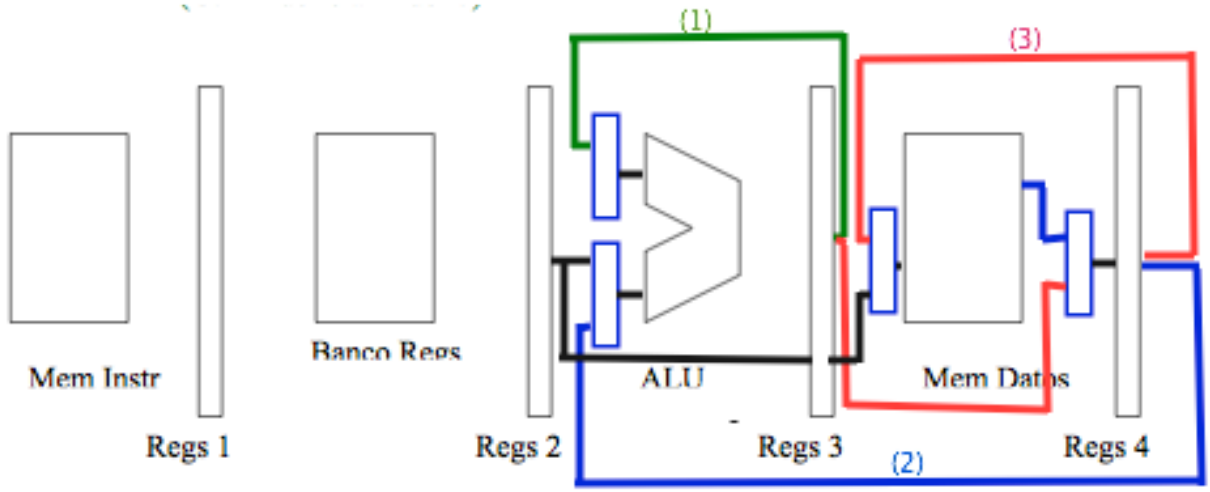
e) ¿Cuántos ciclos tarda en ejecutarse las dos iteraciones del bucle? ¿ Cual es el CPI obtenido? Cuantos ciclos adicionales han sido necesarios respecto a la ejecución ideal.

Dos iteraciones se ejecutan en 26 ciclos.
Numero de instrucciones NI= 8x2= 16 instrucciones
CPI= 26/16
CPI ideal = NI + (número etapas -1) = 16 + 4 = 20 ciclos.
Se ha penalizado 6 ciclos (4 por riesgos de control y 2 por dependencia de datos RAW)

f) ¿Cuándo se captura la instrucción I9? ¿Cuándo se ejecuta?

La primera vez se captura I9 en el ciclo 11 pero no llega a la etapa Ex, después de la segunda iteración se captura en el 23 se ejecuta en el ciclo 25 y finaliza en el ciclo 27.

g) Complete el esquema siguiente con los componentes y conexiones necesarias para que el procesador realice los adelantamientos que son imprescindibles en la ejecución del código anterior, indicando cual se aplica en cada caso.



h) ¿Cómo se modifica el funcionamiento (analice solo la finalización de 10 instrucciones) , si la arquitectura es Von Neumann?
No cambia

Instrucción \ tiempo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
I1: Loop: SUB R1, R1, #4	F*	D	E	M	W								F*	D	E	M	W			
I2: LD R2, $1000(R1)		F*	D	E	M*	W								F*	D	E	M	W		
I3: ADD R3, R3, R2			F*	D	(d)	E	M	W												
I4: LD R4, $2000(R1)				F*	f	D	E	M*	W											