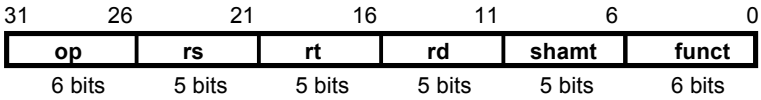
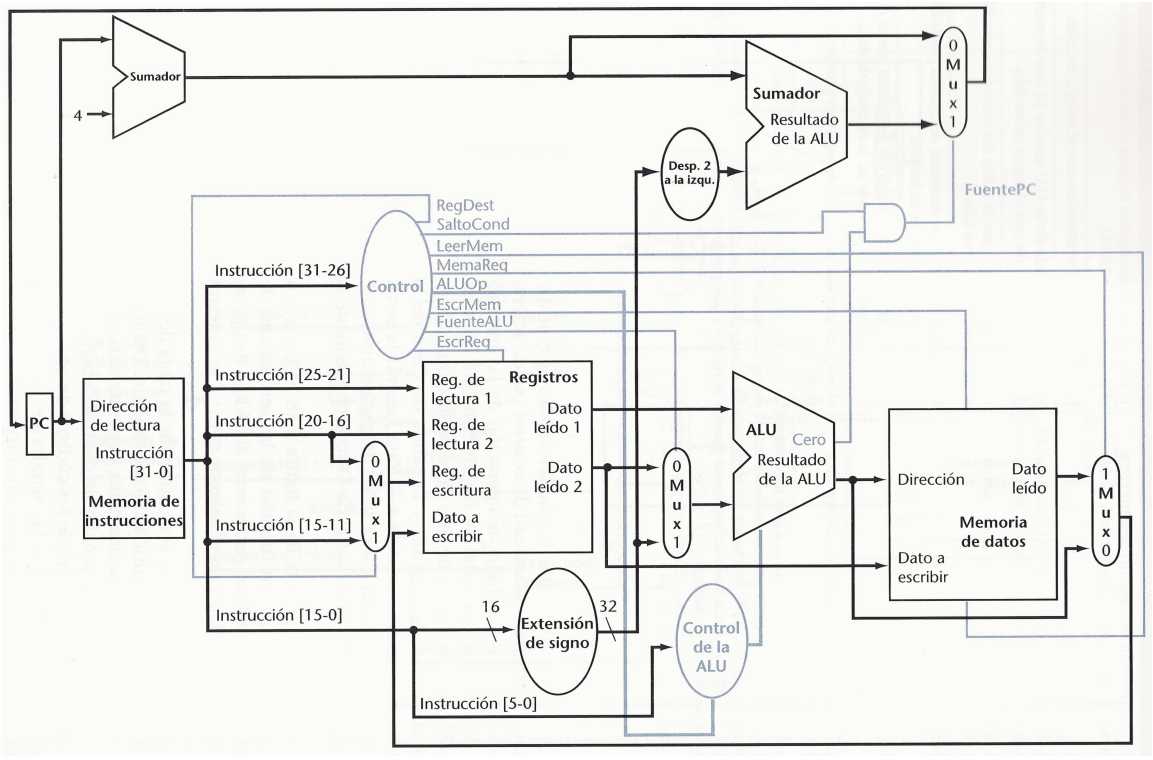
ARQUITECTURA DE COMPUTADORES
CAPÍTULO 2. PROCESADORES SEGMENTADOS

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
ADD	1	0	0	1	0	0	0	1	0
LOAD									
STORE									
BEQ									
AND									

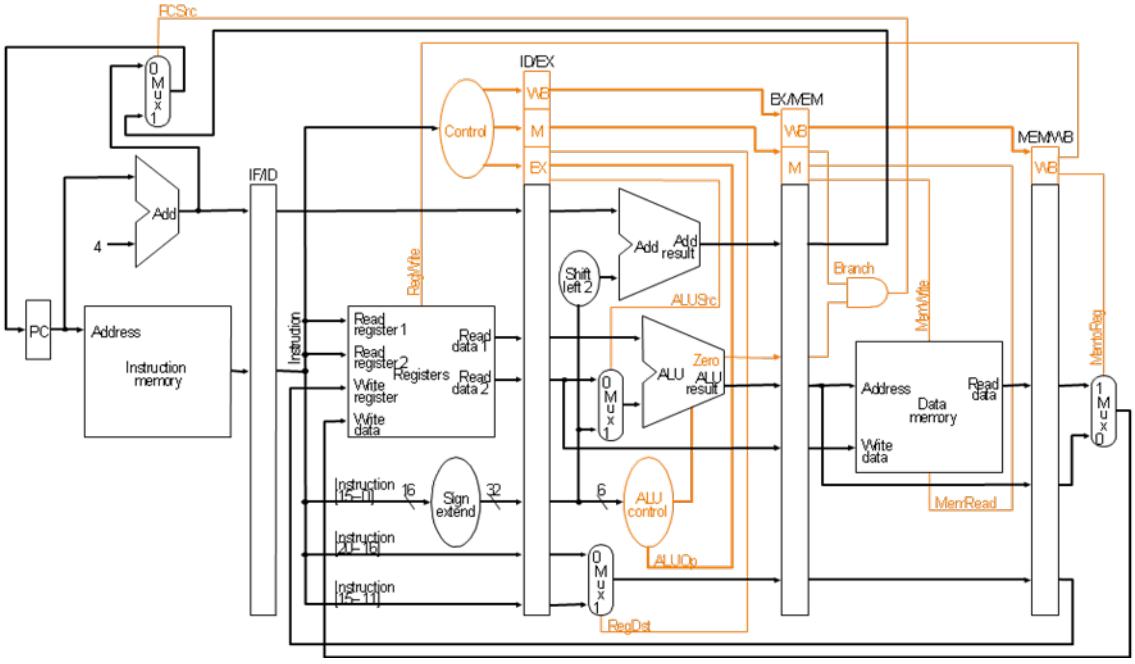
Nota: Descripción de las instrucciones:

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?

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

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"; -- j e instrucciones no validas

  RegDst <= '1' when (opcode="000000") -- add, sub, and, or, slt
          else '0'; -- resto de instrucciones no tienen 3 registros

  BrCond <= '1' when (opcode="000100") -- beq
          else '0'; -- resto de instrucciones

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

  MemWrite <= '1' when (opcode="101011") -- sw
            else '0'; -- 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)
```

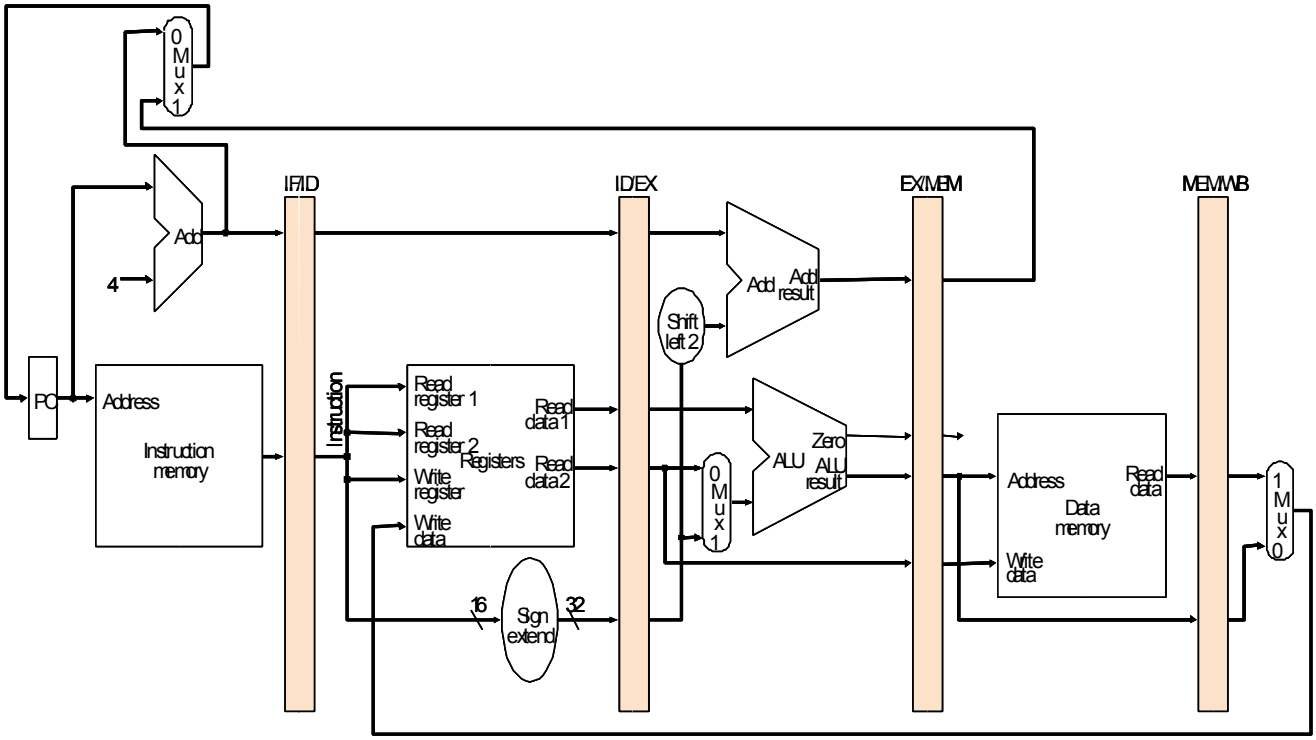
Se pide completar la tabla a partir del momento en que se decodifica el segundo load (lw).

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

Donde el primer grupo de señales son las que genera la unidad de control, las que tienen el prefijo IDEX son las señales de control registradas en el registro ID/EX de la figura, las que tienen el prefijo EXMEM son las señales de control registradas en el registro EX/MEM de la figura, y las que tienen el prefijo MEMWB son las señales de control registradas en el registro MEM/WR de la figura.

Nota: Suponga que los adelantamientos de datos están completamente resueltos en hardware.

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.

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.

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.

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.

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
```

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?

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.

a)

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																
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																			
I2: Loop : LD R1, 0(R2)																			

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														
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																	
I2: Loop : LD R1, 0(R2)																	

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.

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

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.

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).

	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																
I2: LOAD R4, 8(R1)																						
I3: AND R2, R1, R4																						
I4: SUB R5,R5,1																						
I5: BEQ R5,R0,200																						
I6: BEQ R4 ,R6,-24																						
I7: STORE 8(R7), R6																						
I8: BNE R4,R6, -32																						

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

	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													
I2: LOAD R4, 8(R1)																			
I3: AND R2, R1, R4																			
I4: SUB R5,R5,1																			
I5: BEQ R5,R0,200																			
I6: BEQ R4 ,R6,-24																			
I7: STORE 8(R7), R6																			
I8: BNE R4,R6, -32																			

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.

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.

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

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																			
I2: JMP L1																			
I3: L0: XOR R1, R1, R1																			
I4: ADD R1, R1, 4																			
I5: JMP L2																			
I6: L1: STORE 200(R4),R4																			
I7: ADD R4, R4, 4																			
I8: LOAD R5, (R4)																			
I9: SUB R6, R5, 4																			
I10: MUL R7, R5, 3																			
I11: L2: STORE (R6),R7																			

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																			
I2: JMP L1																			
I3: L0: XOR R1, R1, R1																			
I4: ADD R1, R1, 4																			
I5: JMP L2																			
I6: L1: STORE 200(R4),R4																			
I7: ADD R4, R4, 4																			
I8: LOAD R5, (R4)																			
I9: SUB R6, R5, 4																			
I10: MUL R7, R5, 3																			
I11: L2: STORE (R6),R7																			

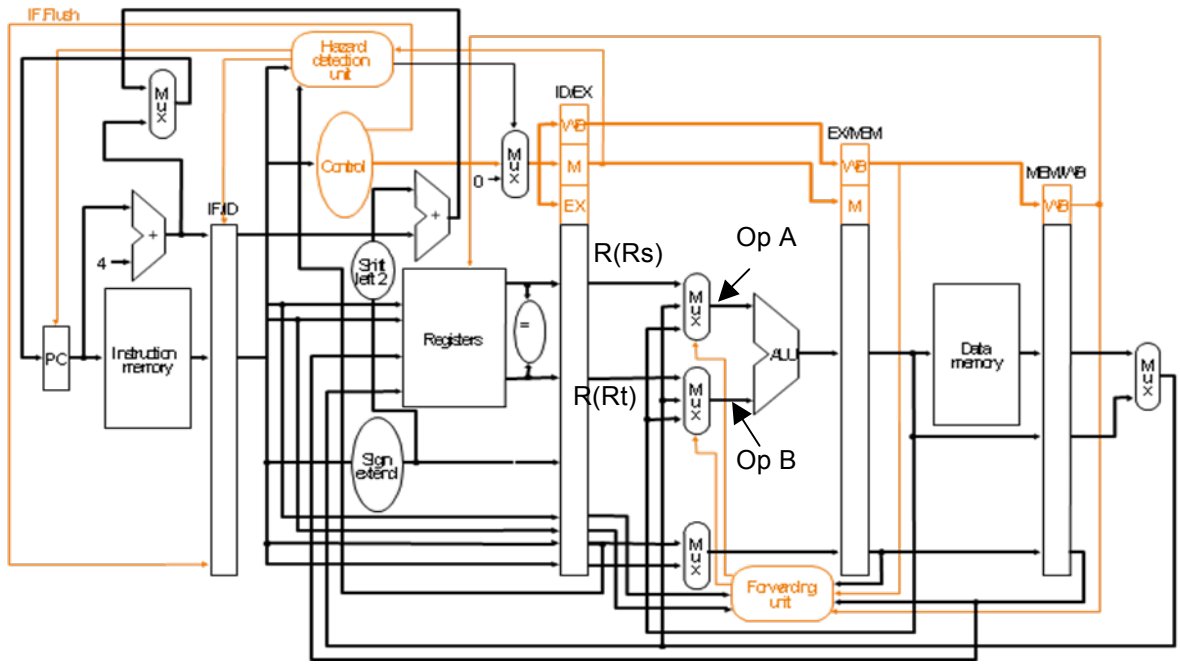
d) Sin adelantamiento de datos y arquitectura Harvard:

Instrucciones	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19
I1: AND R4, R4, R0																			
I2: JMP L1																			
I3: L0: XOR R1, R1, R1																			
I4: ADD R1, R1, 4																			
I5: JMP L2																			
I6: L1: STORE 200(R4),R4																			
I7: ADD R4, R4, 4																			
I8: LOAD R5, (R4)																			
I9: SUB R6, R5, 4																			
I10: MUL R7, R5, 3																			
I11: L2: STORE (R6),R7																			

Con adelantamiento de datos y arquitectura Harvard:

Instrucciones	01	02	03	04	05	06	07	80	09	10	11	12	13	14	15	16	17	18	19
I1: AND R4, R4, R0																			
I2: JMP L1																			
I3: L0: XOR R1, R1, R1																			
I4: ADD R1, R1, 4																			
I5: JMP L2																			
I6: L1: STORE 200(R4),R4																			
I7: ADD R4, R4, 4																			
I8: LOAD R5, (R4)																			
I9: SUB R6, R5, 4																			
I10: MUL R7, R5, 3																			
I11: L2: STORE (R6),R7																			

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

I1: Add r1, r2, r3
I2: Sub r5, r1, r6
I3: And r6, r5, r1
I4: Add r4, r1, r3

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 anticiparA = anticiparB = ID/EX.Reg.Rs = ID/EX.Reg.Rt = EX/MEM.EscrReg = EX/MEM.Reg.Rd = MEM/WB.EscrReg = MEM/WB.Reg.Rd =	Add r1, r2, r3	XXX r10,r1, r2
5		Add r4, r1, r3	And r6, r5, r1 anticiparA = anticiparB = ID/EX.Reg.Rs = ID/EX.Reg.Rt = EX/MEM.EscrReg = EX/MEM.Reg.Rd = MEM/WB.EscrReg = MEM/WB.Reg.Rd =	Sub r5, r1, r6	Add r1, r2, r3
6			Add r4, r1, r3 anticiparA = anticiparB = ID/EX.Reg.Rs = ID/EX.Reg.Rt = EX/MEM.EscrReg = EX/MEM.Reg.Rd = MEM/WB.EscrReg = MEM/WB.Reg.Rd =	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

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.

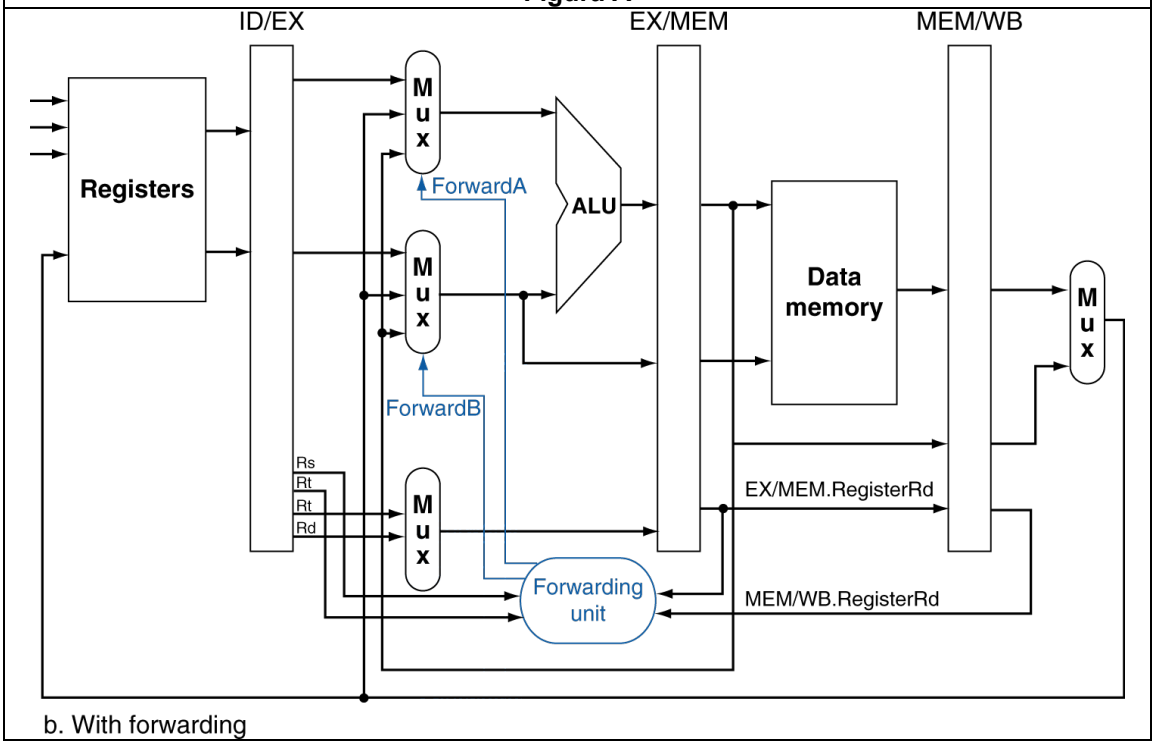
Instrucciones		**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**
ADD	R1, R2, R3	IF	ID	EX	M	W																			
LOAD	R4, 0(R1)																								
MUL	R5, R4, R4																	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

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)																					
MUL	R5, R4, R4																					
BNE	R1, R4, L1																					
SUB	R5, R5, R2																					
ADD	R7, R4, R5																					
L1: ADD	R6, R1, R5																					
OR	R1, R7, R8																					
ST	R6, 0(R1)																					

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 adelantamiento.

ARQUITECTURA DE COMPUTADORES
CAPÍTULO 2. PROCESADORES SEGMENTADOS

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.

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
```

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

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.

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.

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

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

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.

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.

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:

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

Salto efectivo. Total ciclos perdidos:

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

Demora media agregada por los saltos medida en 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.

Demora media agregada por los saltos usando "*Delay Slot"* (medido en ciclos):

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).

ARQUITECTURA DE COMPUTADORES
CAPÍTULO 2. PROCESADORES SEGMENTADOS

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														
Add														
Store														
Mult														
Sub														
BLTD														
Add														
Store														
Mult														

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.

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.

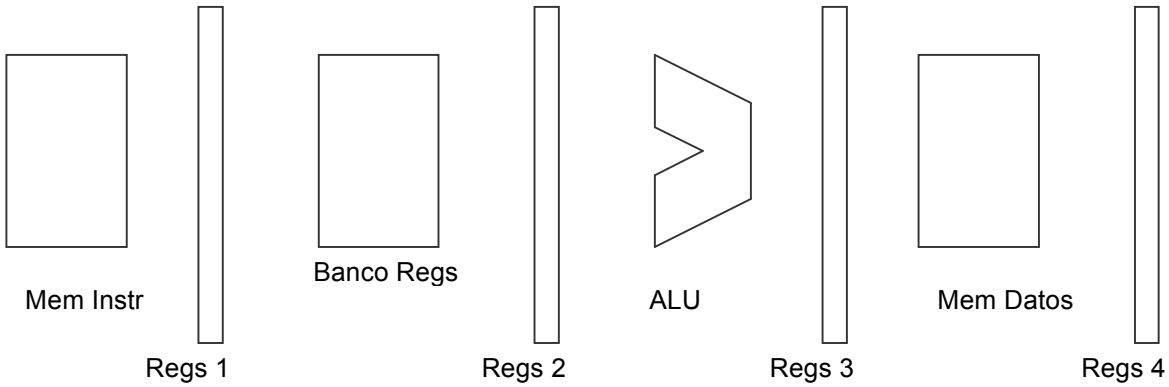
Instrucción \ tiempo	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
I1: Loop: SUB R1, R1, #4	F	D	E	M	W															
I2: LD R2, $1000(R1)																				
I3: ADD R3, R3, R2																				
I4: LD R4, $2000(R1)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

d) ¿Cuántos ciclos tarda en finalizar las 10 instrucciones

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.

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