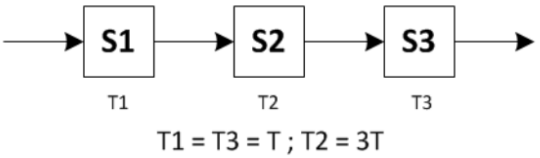
Práctico 4 ADC - Procesador con Pipeline

Ejercicio 1:

En un microprocesador con tres etapas de pipeline: S_1 -> S_2 -> S_3, con tiempos de ejecución $T_1 = T_3 = T$ y $T_2 = 3T$.



T1 = T3 = T ; T2 = 3T

a) ¿Cuál de los tres segmentos o etapas causa la congestión? (el cuello de botella).

b) Asumiendo que el segmento problemático se puede dividir en dos etapas consecutivas, ninguna de ellas con duración menor que T, ¿cuál sería la mejor partición posible? ¿Cuál sería el período de *clock* resultante para este nuevo pipeline de 4 etapas?

c) Asumiendo que el segmento problemático se puede dividir en varias etapas consecutivas de duración T, ¿cuál es el período de *clock* del microprocesador? ¿Cuál es el tiempo de ejecución entre instrucciones?

a. La etapa que genera cuello de botella o congestión, es S2 (ya que la etapa más lenta es la que marca el periodo del clock), haciendo que S1,S1 pasen 2T sin realizar ninguna operación.
b. Si pudiéramos dividir a S2 en dos segmentos, serían cada uno de 1.5T. Siendo que el periodo del clock con este nuevo pipeline de 4 etapas de 1.5T (ya que la etapa más lenta es la que marca el periodo del clock).
c. El periodo del clock con esta nueva división sería de T, ya que todas las etapas tendrán una duración de T. La duración entre instrucciones, también sería de T.

Ejercicio 2:

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	10ns	9ns	40ns	20ns

a) ¿Cuánto tiempo se requiere en un microprocesador sin pipeline para completar la ejecución de la instrucción de mayor latencia, es decir, la latencia del procesador completo?

b) Si se requiere ejecutar esa instrucción en un microprocesador con pipeline, ¿a qué velocidad debería trabajar el *clock*?

c) ¿Cuál es el tiempo de ejecución de una instrucción en un microprocesador con pipeline? ¿Cada cuanto se ejecuta una nueva instrucción en este procesador?

d) Si un microprocesador con pipeline ejecuta 3 instrucciones consecutivas ¿cuál es la ganancia de velocidad de un procesador con pipeline respecto de uno sin pipeline? ¿Y si se ejecutan 1000 instrucciones consecutivas?

$$Ganancia\ de\ velocidad\ = \frac{Tiempo\ de\ ejecución\ sin\ pipline}{Tiempo\ de\ ejecución\ con\ pipline}$$

a. La instrucción de mayor latencia será aquella que necesite de todas las etapas, si las sumamos, obtenemos : 30 + 10 + 9 + 40 + 20 = 109 ns. Por lo tanto, la instrucción de mayor latencia, en un procesador sin pipeline, es de 109.
b. El clock estaría limitado en período por la etapa de mayor latencia, por lo que, el periodo del clock sería de 40 ns. Luego, la velocidad es la inversa, 1/40 = 25MHZ.
c. Una instrucción pasaría por 5 etapas de 40ns hasta ejecutarse en su completitud, por lo que 5 * 40 ns = 200 ns es el tiempo que tarda en ejecutarse una instrucción en este procesador con pipeline. Cada 40 ns se ejecuta una nueva instrucción.
d. Ganancia de velocidad = (109 ns) * 3 / (200 + 40 + 40) = 327 / 280 = 1.16785714. Luego, este procesador con pipeline es 1.16785714 veces más rápido (adimensional), con respecto a 3 instrucciones. Con respecto a 1000 instrucciones, el calculo seria (109 ns) * 1000 / (200 ns + 999 * 40ns) = 2.714 veces más rápido el procesador con pipeline, con respecto a la ejecución de 1000 instrucciones.

Ejercicio 3:

Asumiendo que las etapas individuales del pipeline de dos procesadores ARM distintos tienen las siguientes latencias:

Caso	IF	ID	EX	MEM	WB	Unidad
1	300	400	350	500	100	ps
2	200	150	120	190	140	ps

Arquitectura de Computadoras 2023

a) ¿Cuál es el ciclo de *clock* para la versión con y sin pipeline para cada caso?

b) ¿Cuál es la latencia de la instrucción **LDUR** para ambos, considerando las versiones de procesador con y sin pipeline?

c) Si se pudiera partir una etapa del pipeline en dos nuevas etapas, cada una con la mitad de la latencia de la etapa original, ¿Que etapa elegiría y cuál será el nuevo ciclo de *clock*?

a. Para el primer caso :
   - Con pipeline : 500 ps
   - Sin pipeline : 1650 ps

   Para el segundo caso :
   - Con pipeline : 200ps
   - Sin pipeline : 800 ps

b. Latencia de Ldur para el primer caso :
   - Con pipeline : 2500 ps
   - Sin pipeline : 1650 ps

Latencia de Ldur para el segundo caso :

- Con pipeline : 1000 ps
- Sin pipeline : 800 ps

c. Para el primer caso, partimos la etapa más lenta de 500 ps en dos, generando dos etapas de 250 ps, por lo que el nuevo periodo del ciclo de clock sería de 400 ps. En el segundo caso, análogamente, partimos la etapa de 200 ps en dos de 100, y el nuevo periodo de clock sería de 190 ps.

Ejercicio 4:

Dados los siguientes fragmentos de código de instrucciones LEGv8:

A	B	C
1> SUB X6, X1, X3 2> SUB X7, X6, X5	1> ADDI X10, X1, #8 2> LDUR X3, [X10, #0] 3> SUB X3, X2, X10	1> LDUR X1, [X10, #0] 2> LDUR X2, [X10, #8] 3> ADD X3, X1, X2 4> STUR X3, [X10, #24] 5> LDUR X4, [X10, #16] 6> ADD X5, X1, X4 7> STUR X5, [X10, #32]

a) Analizar en el código las dependencias de datos. En cada caso indicar: los números de las instrucciones involucradas y el operando en conflicto.

b) Mostrar el orden de ejecución de las instrucciones y determinar cuales generan *data hazards*. En cada caso indicar en qué etapa se genera el hazard.

b) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall*.

c) Reescribir la sección de código alterando el orden de las instrucciones para evitar *stalls* innecesarios. Mostrar el nuevo orden de ejecución.

d) ¿Cuántos ciclos toma la ejecución del código en cada caso?

a. Para el programa A :

Dependencia Tipo	Instr. 1, Instr. 2, Reg
Datos	1, 2, X6

Para el programa B :

Dependencia Tipo	Instr. 1, Instr. 2, Reg
Datos	1, 2, X10
Datos	1, 3, X10

Práctico 4 ADC - Procesador con Pipeline

Para el programa C :

Dependencia Tipo	Instr. 1, Instr. 2, Reg
Datos	1, 3, X1
Datos	2, 3, X2
Datos	1, 6, X1
Datos	6, 7, X5
Datos	3, 4, X3
Datos	5, 6, X4

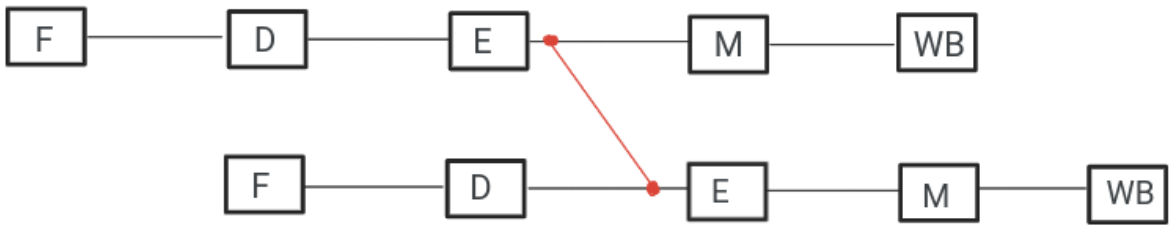
b. Para el programa A :

Sub X6, X1, X3	Fetch	Decode	Execute	Mem	WB	
Sub X7, X6, X5		Fetch	Decode	Execute	Mem	WB

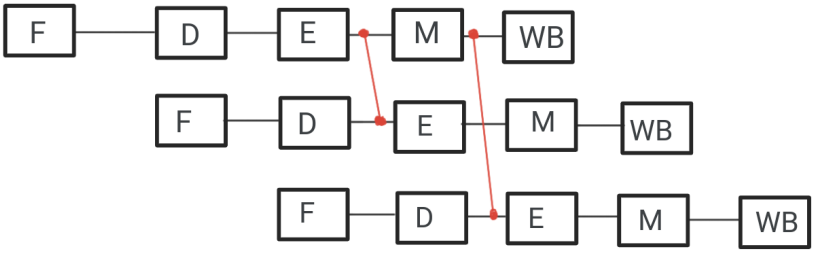
Para el programa B :

Addi X10, X1, #8	Fetch	Decode	Execute	Mem	WB		
Ldur X3, [X10, #0]		Fetch	Decode	Execute	Mem	WB	
Sub X3, X2, X10			Fetch	Decode	Execute	Mem	WB

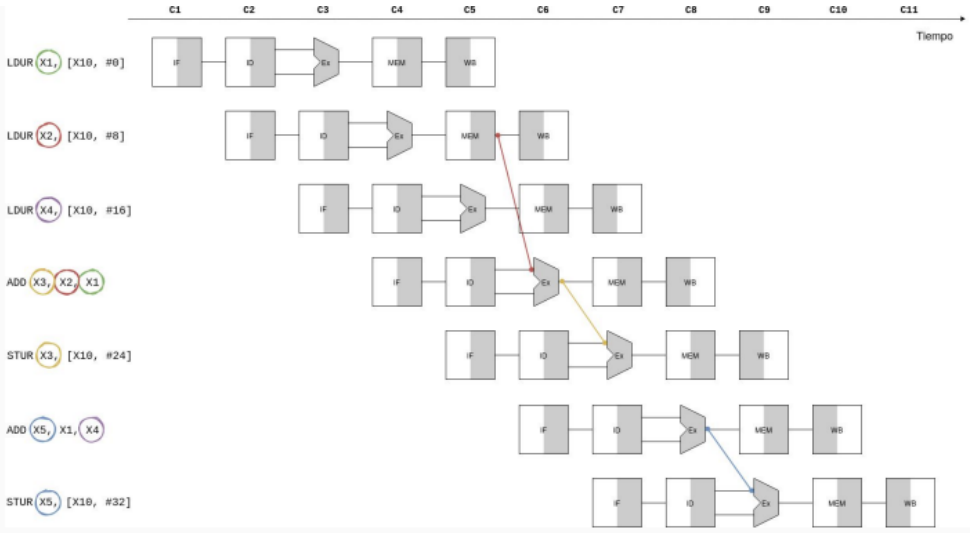
c. Para el programa A (forwarding stall) :



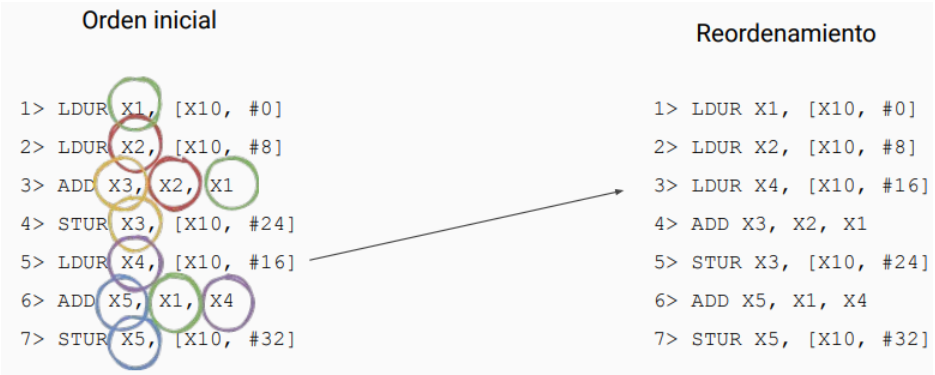
Para el programa B (forwarding stall) :



Práctico 4 ADC - Procesador con Pipeline



d. Ejemplo de reescritura del código C :



e. Para el programa A, se necesitan 6 ciclos de reloj para completar la ejecución del código. Para el programa B, se necesitan 7 ciclos de reloj. Por último, para el programa C, 11 ciclos de reloj hasta completar la ejecución.

Ejercicio 5:

Dados los siguientes fragmentos de código de instrucciones LEGv8:

A	B
1> LDUR X1, [X2, #80] 2> ADD X2, X3, X3 3> ADD X1, X1, X2 4> STUR X1, [X2, #40]	1> ADD X1, X2, X3 2> STUR X2, [X1, #0] 3> LDUR X1, [X2, #8] 4> ADD X2, X2, X1

a) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *stall*.

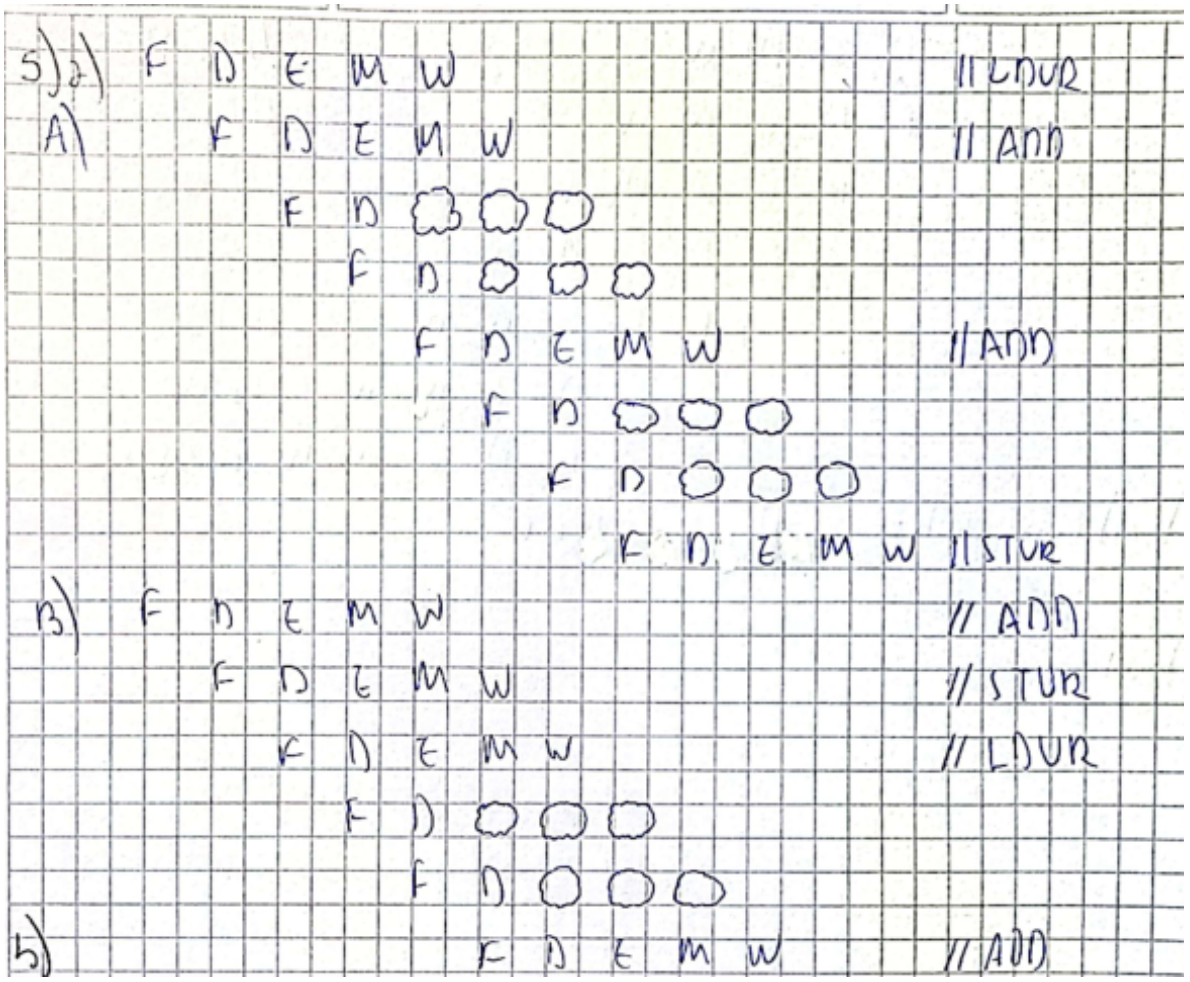
b) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall*.

c) Agregar instrucciones **nop** a las secuencias 'A' y 'B' para asegurar la correcta ejecución en un procesador sin soporte de *forwarding stall*. Mostrar gráficamente el orden de ejecución del programa en el pipeline.

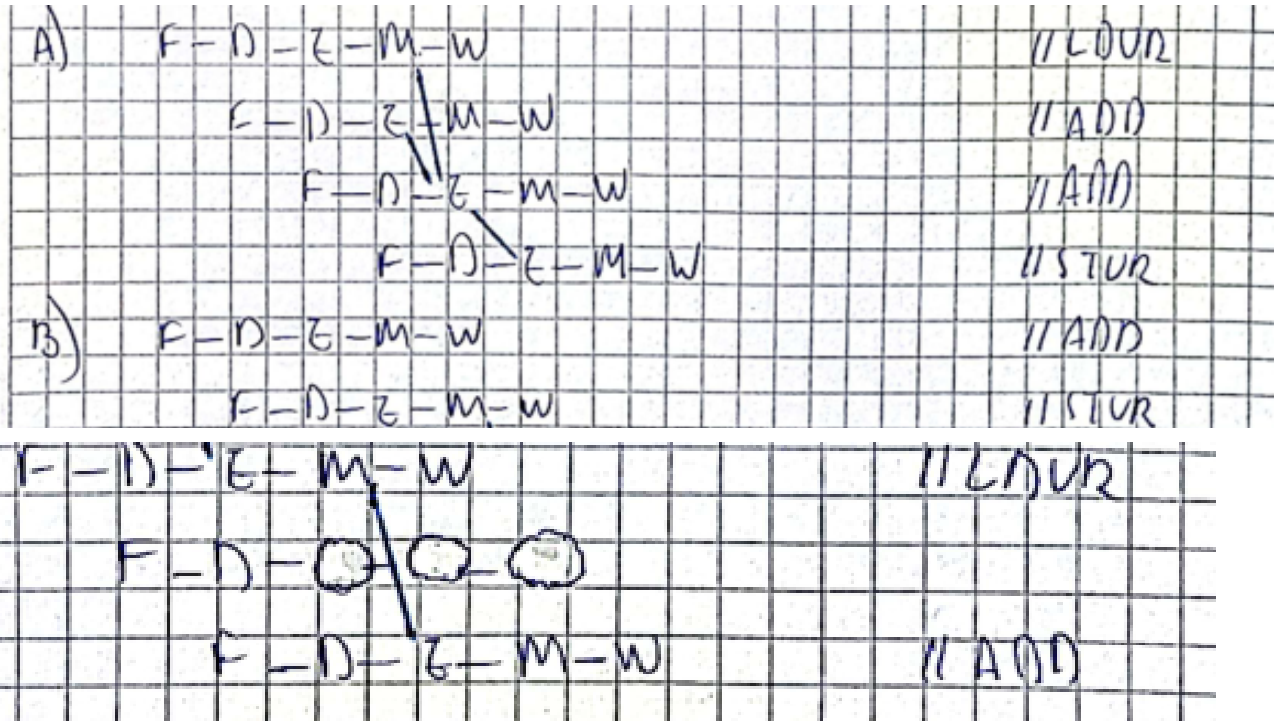
d) Considerando un clock de 1GHz, calcular el tiempo de ejecución de ambos programas para los tres casos anteriores.

Práctico 4 ADC - Procesador con Pipeline

a.



b.



c.

Ldur X1, [X2, #80]	Add X1, X2, X3
Add X2, X3, X3	Stur X2, [X1, #0]
nop();	Ldur X1, [X2, #8]
nop();	nop();
Ldur X1, [X2, #80]	nop();
nop();	Add X2, X2, X1
nop();	
Stur X1, [X2, #40]	

d.

El programa “A” se ejecuta en 8 ciclos de clock → T = 8 / 1.000.000.0000 = 8 NS.
El programa “B” se ejecuta en 10 ciclos de clock → T = 10 / 1.000.000.000 = 10 NS.

Ejercicio 6:

Dado el siguiente fragmento de código de instrucciones LEGv8:

```
1>        SUB X3, X1, X2
2>        CBZ X3, skip
3>        ADD X4, X3, XZR
4>        ORR X7, X1, X2
5>        AND X1, X6, X2
6>        STUR X5, [X2, #40]
7> skip: ADD X5, X3, XZR
```

a) Analizar en el código las dependencias de datos y de control, determinar cuales generan *hazards*. En cada caso indicar: los números de las instrucciones involucradas, en qué etapa se encuentra c/u, el operando en conflicto y el tipo de hazard.

b) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall* suponiendo que X1 != X2. ¿Cuál es la penalidad por el hazard de control?

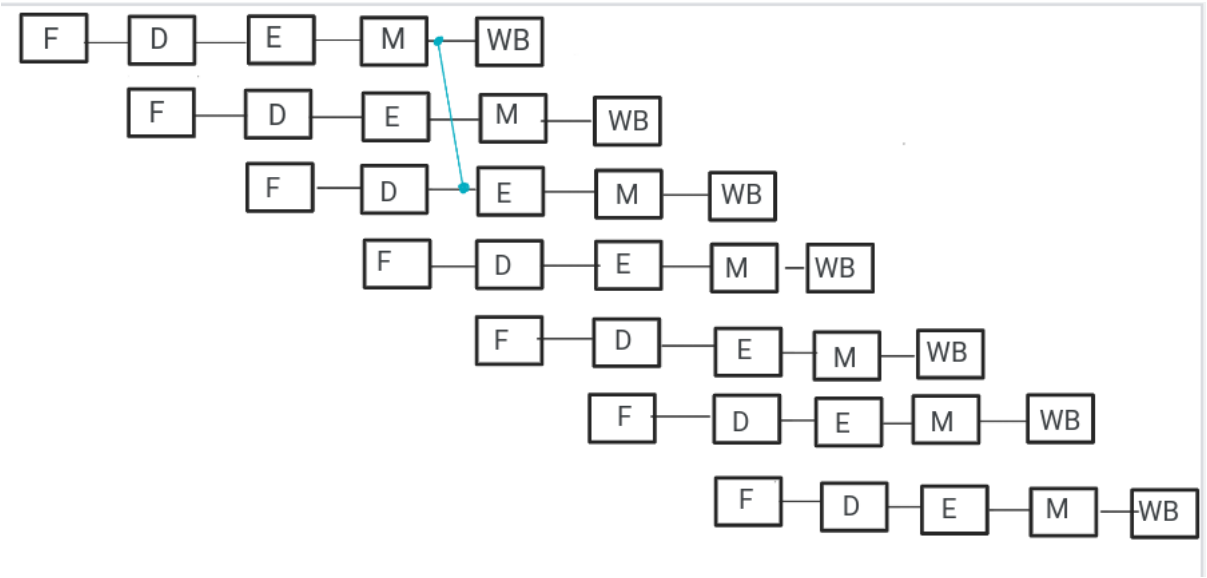
c) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall* suponiendo que X1 = X2. ¿Cuál es la penalidad por el hazard de control?

Práctico 4 ADC - Procesador con Pipeline

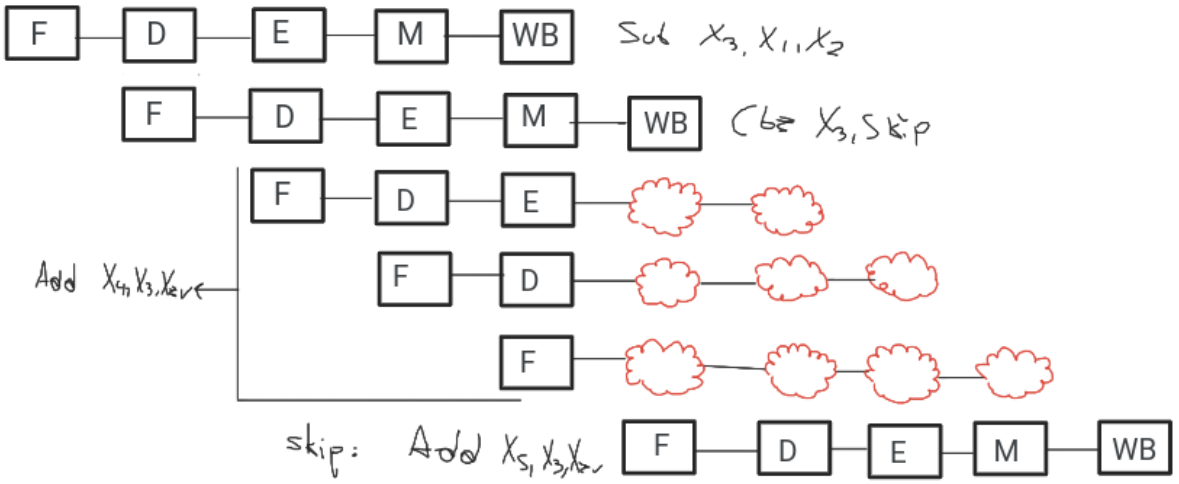
a.

Dependencia Tipo	Instr. 1, Instr. 2, Reg	Genera Hazard? Solución
Control	2	Si salta, no tiene solución.
Datos (condicional)	1, 3, X3	Si, Forwarding.
Datos (condicional)	1, 7, X3	Si, Forwarding.
Datos	1, 7, X3	No

b. Penalidad es de cero ciclos de clock.



c. Penalidad : 3 ciclos de clock.



Práctico 4 ADC - Procesador con Pipeline

Ejercicio 7:

Para el siguiente fragmento de código de instrucciones LEGv8 que se ejecuta en un procesador con forwarding stall:

```
1>        SUB X3,X1,X2
2>        ORR X8,X5,X6
3> L:     LDUR X0, [X3, #0]
4>        ADD X3,X3,X8
5>        CBNZ X0, L
6>        ADD X8,X0,X3
```

a) Analizar en el código las dependencias de datos y de control, determinar cuales generarían *hazards* y mediante qué técnica se evita. En cada caso indicar: los números de las instrucciones involucradas, el operando en conflicto y el tipo de hazard.

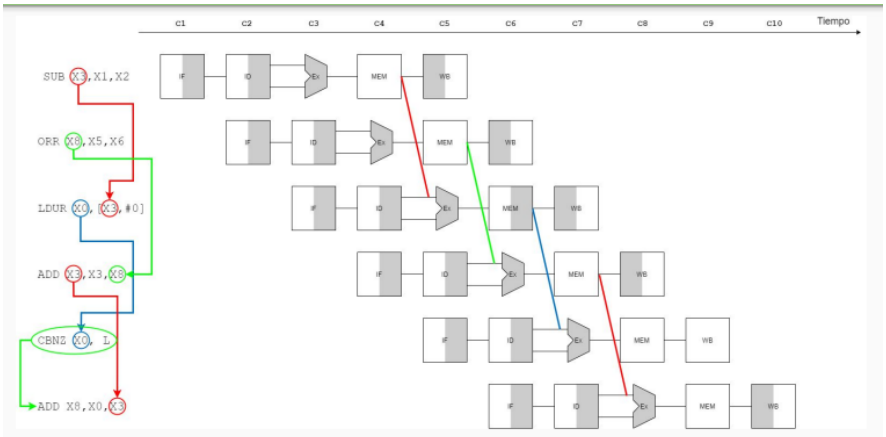
b) Mostrar el orden de ejecución y los recursos utilizados por las instrucciones para el segmento de código dado, suponiendo que CBNZ no se toma.

c) Mostrar el orden de ejecución y los recursos utilizados por las instrucciones para el segmento de código dado, suponiendo que CBNZ se toma una vez.

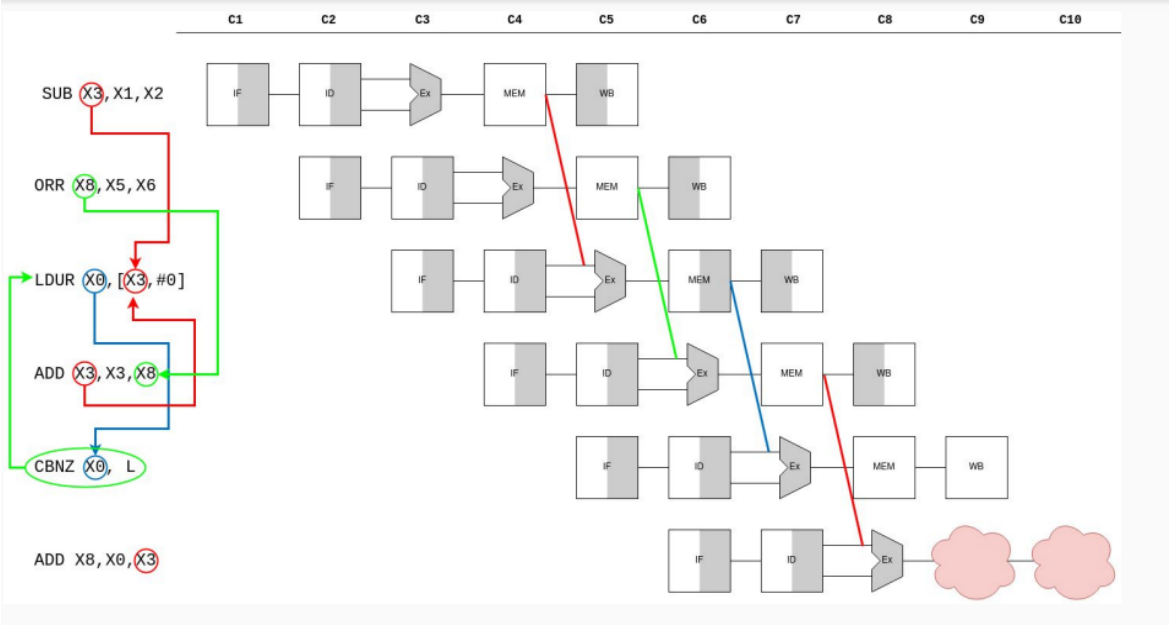
a.

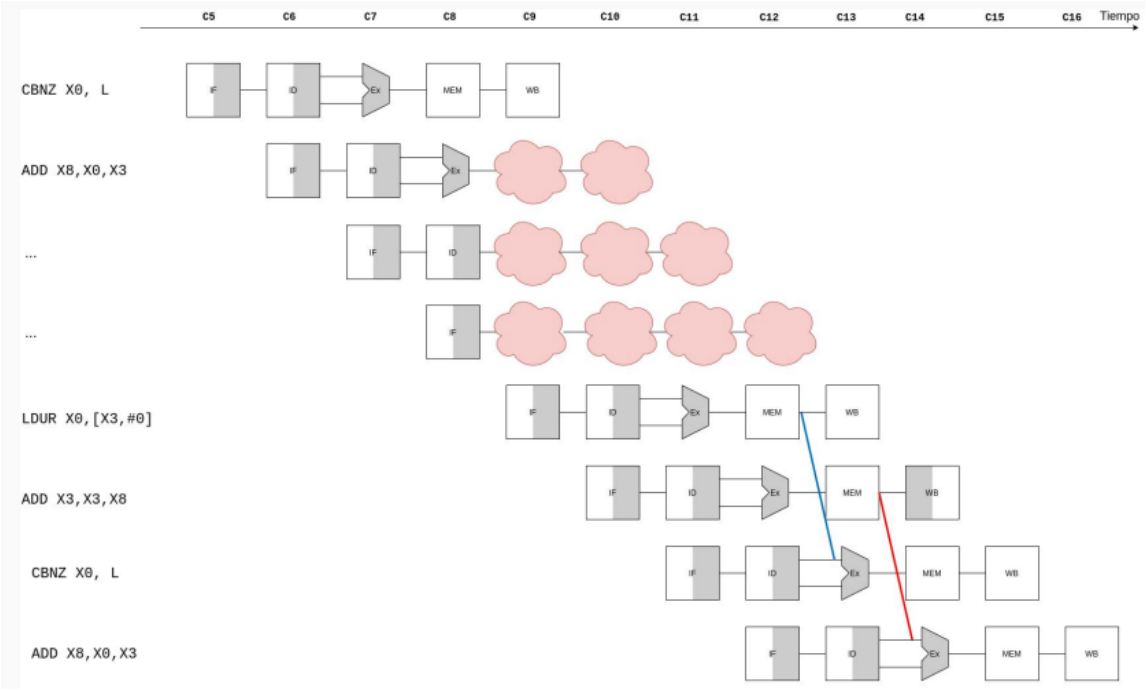
Dependencia Tipo	Instr. 1, Instr. 2, reg	Hazard? Solución?
Datos	1, 3, X3	Si, forwarding
Datos	1, 4, X3	No
Datos (condicional)	4, 6, X3	Si, forwarding
Datos	2, 4, X8	Si, forwarding
Datos (condicional)	3, 6, X0	No
Control	5	Solo si salta, no
Datos	3, 5, X0	Si, forwarding

b. El salto no se toma :



c. El salto si se toma :





- El “Add X8, X0, X3” se convierte en un flush (dura 3 ciclos de clock), ya que nuestro procesador siempre asume que el salto no se realizará, por ende, tiene esa penalidad de 3 ciclos hasta darse cuenta que debería haber hecho el salto (se da cuenta en la etapa Mem).

Ejercicio 8:

En la siguiente secuencia de código los registros X6 y X7 han sido inicializados con los valores 0 y 8N respectivamente.

```
loop:
    1> LDUR X2, [X6, #40]
    2> LDUR X3, [X6, #48]
    3> ADD X2, X2, X3
    4> STUR X2, [X6, #40]
    5> ADDI  X6, X6, #8
    6> CMP X6, X7
    7> B.NE loop
     …
```

a) Determinar qué técnica de predicción de salto (no dinámica) generará la menor cantidad de demoras por flush instructions si N > 1.

b) Analizar la ejecución del código considerando que el resultado y la dirección del salto se determinan en la etapa de decodificación (ID) y se aplican en la etapa de ejecución (EX) y que no hay hazards de datos (Figura 4.61 de "Computer organization and design - ARM edition" Patterson & Hennessy).

a. loop:
LDUR X2, [X6, #40] → X2 = 0
LDUR X3, [X6, #48] → X3 = 0
ADD X2, X2, X3 → X2 = 0
STUR X2, [X6, #40] → &X6[40] = 0
ADDI X6, X6, #8 → X6 = 8
CMP X6, X7 → X6 == X7
B.NE loop

La técnica que generará la menor cantidad de demoras por flush es "taken", es decir, asumir que hay que saltar. Ya que en el "B.NE", la bandera de salto se levanta cuando X6 != X7 por lo que en el único caso que no habría que saltar es cuando ambos sean iguales.

Práctico 4 ADC - Procesador con Pipeline

Ejercicio 9:

Asumiendo que los saltos condicionales son perfectamente predichos (elimina el riesgo de *hazard* de control), si se tiene una única unidad de memoria para instrucciones y datos, existe un riesgo estructural cada vez que se necesita leer una instrucción en el mismo ciclo en que otra instrucción accede a un dato.

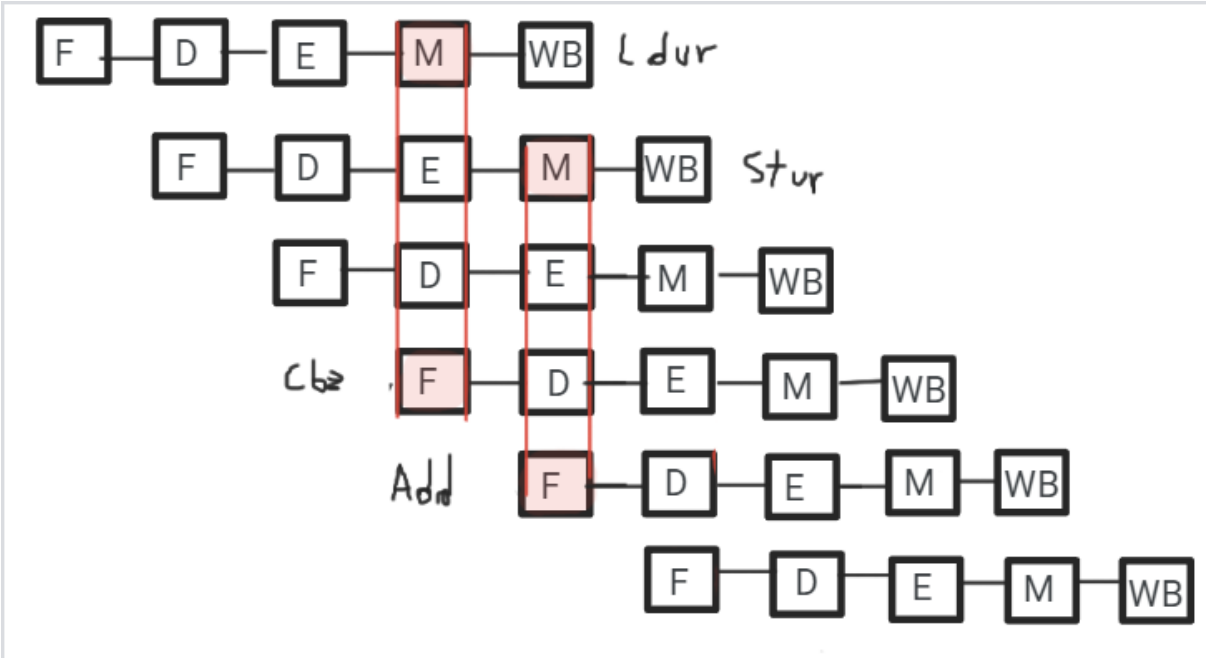
Dados el siguiente fragmento de código de instrucciones LEGv8:

```
                1> STUR X16, [X6, #12]
                2> LDUR X16, [X6, #8]
                3> SUB X7, X5, X4
                4> CBZ X7, Label
                5> ADD X5, X1, X4
                6> SUB X5, X15, X4
Label:
```

a) Asumiendo *forwarding stall* (el cual solo elimina los hazards de datos), mostrar gráficamente el orden de ejecución del programa en el pipeline. Identificar dónde se generan los *Hazards estructurales*.

b) Anteriormente, se vio que los riesgos de datos se pueden eliminar agregando instrucciones **nop** en el código. ¿Se puede hacer lo mismo con este hazard? ¿Por qué?

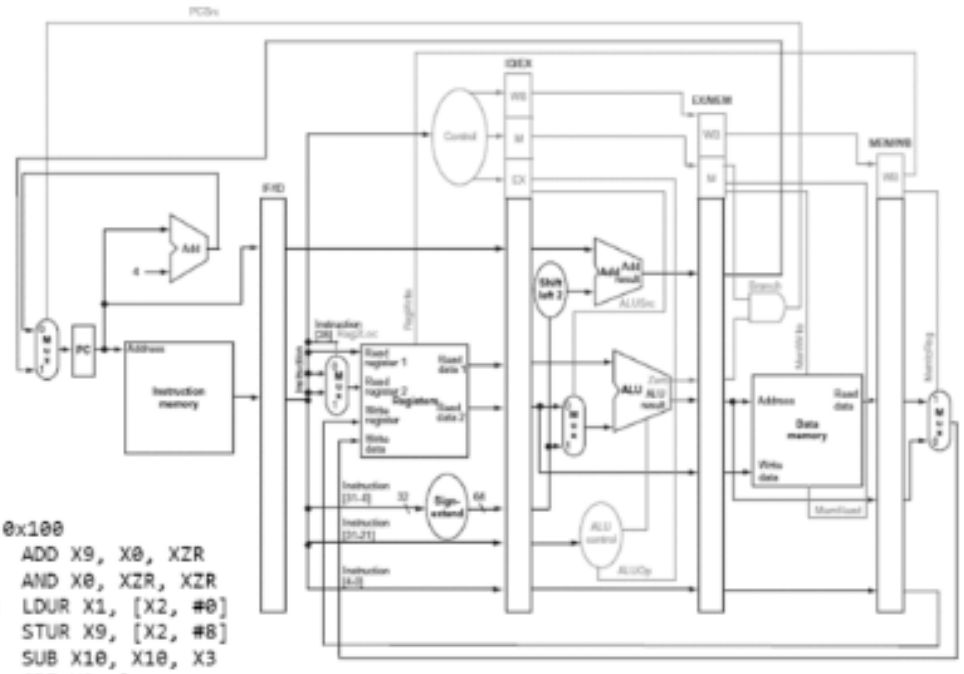
a. Los hazards estructurales se encuentran en los clocks resaltados :



b. No se puede hacer un nop para solucionar, ya que estamos en el mismo bloque de memoria.

Ejercicio 10

El segmento de código dado se ejecuta en el procesador LEGv8 con pipeline de la figura (sin *forwarding stall*). Analizando el estado del procesador en el ciclo de clock número 5 (considerando que el fetch de la primera instrucción se realiza en el ciclo número 1), responder:



```
.org 0x100
       ADD X9, X0, XZR
       AND X0, XZR, XZR
loop:  LDUR X1, [X2, #0]
       STUR X9, [X2, #8]
       SUB X10, X10, X3
       CBZ X3, loop
done:  …
```

a) ¿Qué valor hay a la salida del PC? 0x110

b) ¿Qué valor hay en la entrada "Write register" del bloque Registers? 0x9

c) ¿Qué operación realiza la ALU? X2 + 0

d) Completar el siguiente cuadro con los valores que tienen las señales de control:

Reg2Loc	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch
1	1	0	1	0	0	0