Universidad Católica de Córdoba
Facultad de Ingeniería
ARQUITECTURA DE COMPUTADORAS II

Fecha: 25/10/2022
Nombre:

Parcial 2

Ejercicio 1 (20%):

Dada la siguiente secuencia de código en assembler LEGv8:

```
ADDI X9, X6, #8
ADD  X10, X6, XZR
STUR X10, [X9, #0]
LDUR X9, [X9, #0]
ADD  X0, X9, X10
```

n = 5 instr.

Y asumiendo que las etapas individuales ~~del pipeline~~ del procesador LEGv8 tienen las siguientes latencias:

Instruction Fetch	Instruction Decode	Execute	Memory	Write Back
100 ns	60 ns	80 ns	120 ns	70 ns

a) Completar la tabla:

	Sin pipeline	**Con** pipeline
Frecuencia del reloj mínima	$\frac{1}{430} = 2{,}3$ MHz	$\frac{1}{120} = 8{,}3$ MHz
Latencia de una instrucción	430 ns	600 ns
Tiempo que tarda en ejecutar la secuencia dada(*)	5 · 430 ns = 2150	600 + 120 · 4 = 1080
Mejora en el tiempo de ejecución de la secuencia	Ganancia = 1.9907	

(*) asumir que el procesador con pipeline tiene implementado forwarding-stall

b) Asumiendo que es posible rediseñar una de las etapas del procesador dado y disminuir su latencia a la mitad, responder:

i) ¿Sobre qué etapa se debería trabajar para mejorar el tiempo de ejecución del procesador? Memoria

ii) ¿Cuál sería la frecuencia de reloj del procesador **con** pipeline modificado? $\frac{1}{100} = 10$ MHz

Ejercicio 2 (45%):

La siguiente secuencia de código se ejecuta en un procesador LEGv8 con pipeline. Considerando que antes de la ejecución del código, el contenido de la posición de memoria apuntada por X5 es: 0b1101.

```
1> E1:  ADDI X1, XZR, #1      // X1 = 1 = 0001
2>      LDUR X2, [X5, #0]     // X2 = 1101
3>      AND  X0, X1, X2       // X0 = 0001
4>      CBZ  X0, E2           // NO SALTA (0001 ≠ 0000)
5>      SUBI X2, X2, #1
6>      CBZ  XZR, E3          // SI SALTA
7> E2:  SUBI X3, X3, #1
8> E3:  ADD  X5, X5, #8
9>      STUR X3, [X5,#0]
```

Universidad Católica de Córdoba
Facultad de Ingeniería
ARQUITECTURA DE COMPUTADORAS II

Fecha: 25/10/2022
Nombre:

a) Analizar en el código las dependencias de datos y de control, y luego completar la siguiente tabla:

Tipo de dependencia	Registro involucrado (si aplica)	N° instrucciones involucradas	Genera hazard sin forwarding stall?
...	...	...	...

b) Mostrar el orden de ejecución de las instrucciones (hasta que se termine de ejecutar la instrucción n° 9), considerando un procesador con forwarding-stall. Dejar correctamente indicados los caminos de forwarding.

Ejercicio 3 (35%):

Dado un procesador de arquitectura LEGv8 2-issue, que predice los saltos perfectamente, de modo que los hazard de control son manejados por hardware, con una modificación que permite que en cada *issue packet* una instrucción pueda ser cualquier tipo y la otra deba ser una instrucción aritmética/lógica. Para el siguiente fragmento de código LEGv8:

```
        1>  ADDI X0, XZR, #0x100
        2>  ADD X1, XZR, XZR
        3>  ORRI X2, X0, #0x200
LOOP:   4>  LDUR X3, [X0, #0]
        5>  CBZ X3, END
        6>  SUBI X3, X3, #1
        7>  ADDI X0, X0, #8
        8>  ADD X4, X1, X3
        9>  LDUR X4, [X3, #0]
        10> ADDI X2, X2, #8
        11> STUR X4, [X2, #0]
        12> CBZ XZR, LOOP
END:    13> …
```

a) Sin alterar el orden de las instrucciones, mostrar en la siguiente tabla cómo organizaría los issue packets para ejecutar el programa en la menor cantidad posible de ciclos de clock (cada instrucción sólo puede agruparse con la inmediata anterior, la inmediata posterior o una nop). El compilador asume toda la responsabilidad de insertar instrucciones nop para que el código se ejecute sin necesidad de generación de stalls.

Instrucción de cualquier tipo	Instrucción aritmética/lógica
...	...

b) Mostrar el orden de ejecución del código del punto "a" en el procesador 2-issue (sólo hasta completar una iteración del bucle "LOOP", suponer que la posición de memoria 0x100 contiene un valor mayor que cero). Indicar los caminos de forwarding utilizados.