Segundo Examen Parcial

1. Neon es una extension SIMD de la arquitectura ARM, disponible en los procesadores de las series ARM Cortex-A y ARM Cortex-R.

a. **Investigue tres similitudes entre NEON y AVX.**

1. Ambas son extensiones SIMD, ambas permiten ejecutar una misma operación, sobre múltiples datos a la vez, como sumar 4 enteros o floats a la vez.
2. Ambas aguantan operaciones vectoriales sobre números enteros, y de punto flotante, llámese sumas, restas, multiplicaciones, y soportan enteros de 8, 16, 32 bits y floats de 32 bits.
3. Ambos usan registros vectoriales dedicados, AVX usa registros **YMM/ZMM** (256–512 bits) y NEON usa registros **Q y D** (128 bits) y ambos tienen un set de registros aparte para manejar vectores.

b. **Investigue tres diferencia entre NEON y AVX**

1. Arquitectura del Procesador: NEON es la extensión SIMD nativa para la arquitectura ARM , mientras que AVX es la extensión SIMD nativa para la arquitectura x86 (Intel/AMD).
2. NEON está integrado de forma más simple en el pipeline ARM, mientras que AVX requiere más control del sistema (como manejo de estados extendidos y ahorro/restauración más costoso).
3. Modelo de Instrucción: NEON sigue la filosofía RISC con instrucciones de longitud fija y un formato más regular. AVX, al ser x86, tiene instrucciones de longitud variable y permite operaciones más complejas en memoria directamente en algunos casos, lo cual es típico de CISC.

2.Suponga que un procesador RISC con pipelining (por supuesto) ejecuta el siguiente código

ld R1,[R0] ; Se pone en R1 el primer dato
ld R2,[R0,#4] ; Se pone en R2 el segundo dato
ld R3,[R0,#8] ; Se pone en R3 el tercer dato
mul R4,R1,R2 ; Multiplicación de primeros dos datos
add R5,R4,R2 ; Suma del resultado del producto con otro dato

Suponga que la etapa EX de MUL dura 8 ciclos.

a. Haga un diagrama de tiempo que muestre el avance de la ejecución etapa por etapa de este grupo de instrucciones.

- **IF:** Instruction Fetch

- **ID:** Instruction Decode
- **EX:** Execute
- **MEM:** Memory Access
- **WB:** Write Back
- **IC:** Interrupt Check

Ciclo	ld R1, [R0]	ld R2, [R0,#4]	ld R3, [R0,#8]	mul R4, R1, R2	add R5, R4, R2
1	IF				
2	ID	IF			
3	EX	ID	IF		
4	MEM	EX	ID	IF	
5	WB	MEM	EX		
6	IC	WB	MEM		
7		IC	WB	ID	IF
8			IC	EX	
9				EX	
10				EX	
11				EX	
12				EX	
13				EX	
14				EX	
15				EX	
16				MEM	
17				WB	ID
18				IC	EX
19					MEM
20					WB
21					IC

b. Calcule el tiempo promedio de instrucción ideal, y el tiempo promedio de instrucción real en esta ejecución.

Tiempo promedio ideal:

- En un pipeline perfecto, cada instrucción toma **1 ciclo**.
- **CPI ideal = 1**.

Tiempo promedio real:

- **Total de ciclos**: 22 (para 5 instrucciones).
- **CPI real = Total de ciclos / Número de instrucciones = 21 / 5 = 4,2**.

3.¿Cómo se implementan en el procesador SPARC los saltos condicionales?

En la arquitectura SPARC, cuando el procesador necesita realizar un salto condicional, primero consulta los **condition codes** (banderas de estado) que se almacenan en el registro PSR. Estas banderas - *Negative, Zero, Overflow, Carry* - se actualizan automáticamente cada vez que se ejecutan operaciones aritméticas o de comparación.

Las instrucciones de salto como ***bn*** (branch if equal) o ***bne*** (branch if not equal) verifican estas banderas para decidir si el programa debe continuar linealmente o saltar a otra ubicación en memoria. Lo particular de SPARC es que implementa un ***delay slot*** **obligatorio**, que significa que la instrucción inmediatamente posterior al salto **siempre se ejecuta**, sin importar si finalmente se toma o no el salto. Esto no es un error, es característica del diseño que permite mantener el pipeline ocupado y evitar pérdidas de ciclos, aprovecha el tiempo que tarda el procesador en calcular la dirección de destino del salto.