TAREA PROGRAMADA # 2 (Avance proyecto) .

Fecha entrega: **SÁBADO 20 DE SETIEMBRE**

I. Descripción de la tarea:

Para el proyecto posterior del curso: "Simulación de un procesador de dos núcleos MIPS para enteros (con ejecución de hilos de un mismo proceso compartiendo memoria centralizada)" se pide el siguiente avance como tarea programada 2:

Diseñar y programar **uno de los núcleos MIPS**, en el cual debe implementarse el **paralelismo en el nivel de instrucciones** (**pipeline de 5 etapas visto en clase**) para instrucciones de enteros. Los saltos se resolverán en **ID** (o sea, sea se implementará el "pipeline" MIPS "mejorado")

II. Detalles importantes:

1. **Cada etapa del pipeline debe ser implementada con un hilo de ejecución.**
2. De momento este núcleo ejecutará un único programa MIPS cada vez que se corra la simulación.
3. Aunque para el proyecto sí se trabajará con cachés, para esta tarea sólo se van a simular **dos memorias**, una para guardar las **instrucciones** y otra para **almacenar y leer datos**. Se sugiere para ello utilizar vectores.
4. Los programas que ejecutaría este núcleo, ya vendrán codificados en "lenguaje de máquina" de acuerdo con los formatos MIPS, en un archivo de texto con una instrucción por línea, pero NO en código binario, sino decimal. Las instrucciones de ese programa deberán copiarse en el vector que simula la **memoria para instrucciones, y es de ahí de donde se irá haciendo el "fetch" de cada instrucción.** La idea es que el PC debe apuntar a una posición de esta estructura, cada vez que se debe leer una instrucción. Como notará más abajo, cada instrucción está compuesta por 4 números en decimal, por lo que será muy simple simular las direcciones de memoria múltiplo de 4 para las instrucciones ya que ***cada inicio*** de instrucción estaría en un subíndice del vector que es "múltiplo de 4".
5. En el proyecto se implementarán las siguientes instrucciones MIPS, pero **para esta tarea, no se manejarán el LL ni el SC** (sombreadas con gris) **:**

INSTRUCCIÓN			CODIFICACIÓN				
Operación	Operandos	Acción	0-5	6-10	11-15	16-20	21-31
			Cód. Operación (DECIMAL)	Rf1	Rf2-Rd	Rd	inmediato
DADDI	RX, RY, #n	Rx ← (Ry) + n	001000 - 8	Y	X	n	
DADD	RX, RY, RZ	Rx ← (Ry) + (Rz)	100000 - 32	Y	Z	X	
DSUB	RX, RY, RZ	Rx ← (Ry) - (Rz)	100010 - 34	Y	Z	X	
DMUL	RX, RY, RZ	Rx ← (Ry) * (Rz)	011000 - 12	Y	Z	X	
DDIV	RX, RY, RZ	Rx ← (Ry) / (Rz)	011010 - 14	Y	Z	X	
LW	RX, n(RY)	Rx ← M(n + (Ry))	100011 - 35	Y	X	n	
SW	RX, n(RY)	M(n + (Ry)) ← Rx	101011 - 43	Y	X	n	
BEQZ	RX, ETIQ	Si Rx = 0 SALTA	000100 - 4	X	0	n	
BNEZ	RX, ETIQ	Si Rx ≠ 0 SALTA	000101 - 5	X	0	n	
LL	RX, n(RY)	Rx ← M(n + (Ry)) y establece el link register	0	Y	X	n	
SC	RX, n(RY)	If link register = n + Ry Then M(n + (Ry)) ← Rx y pone un 1 en Rx Else pone un 0 en Rx y no escribe en memoria	1	Y	X	n	
JAL	n	R31←PC, PC← PC+n	000011 - 3	n			
JR	RX	PC ← (Rx)	001000 - 2	X	0	0	
"FIN"		Detiene el programa	111111 - 63	0	0	0	

Ejemplo de un "programita" (solo ejemplo para codificación, no resuelve nada):
(cada instrucción va en una línea del archivo de texto)

INSTRUCCIONES EN MIPS			SU CODIFICACIÓN PARA EL PROYECTO
	DADDI	R1, R0, #4	**8 0 1 4**
	SW	R1, 4(R0)	**43 0 1 4**
	LW	R4, 0(R1)	**35 1 4 0**
ET-2	DADD	R6, R5, R8	**32 5 8 6**
	DSUB	R1, R2, R3	**34 2 3 1**
	BEQZ	R6, ET-1	**4 6 0 1**
	LW	R30, 8(R0)	**35 0 30 8**
ET-1	DSUB	R26, R4, R3	**34 4 3 26**
	DADDI	R4, R4, # -1	**8 4 4 -1**
	BNEZ	R4, ET-2	**5 4 0 -7**
	DSUB	R26, R4, R3	**34 4 3 26**
	DADDI	R9, R0, #-500	**8 0 9 -500**
	FIN		**63 0 0 0**

6. En cuanto al vector para memoria de datos, la idea es que el subíndice **i** de dicho vector, representaría la posición de memoria **4*i**.
7. Toda la aritmética a realizar en este procesador se hará en **aritmética decimal**.
8. En cuanto al tamaño de los vectores para la memoria de instrucciones y la de datos:
   - Para instrucciones se sugiere que el vector tenga espacio para almacenar al menos 100 instrucciones, es decir **400 entradas**.
   - Para datos se sugiere que tenga espacio para 200 enteros mínimo (**200 entradas**)
9. Los conflictos de datos se resolverán deteniendo a la instrucción en ID hasta que todos sus operandos estén escritos en el registro correspondiente. Es decir, **no hay "forwarding".**
10. Los conflictos de control (saltos) se detendrá el ingreso de instrucciones hasta que no se resuelva el salto (condicional o no). O sea, **no hay predicción para branches.**
11. Debe simularse **el reloj del procesador** de manera que cada "TIC" de reloj ocurrirá cuando TODAS las etapas del pipeline hayan finalizado lo que debían hacer en ese ciclo de reloj. (Cuando se implemente el otro núcleo para el proyecto, igual deberá trabajarse con este mismo reloj)
12. **Mientras corre la simulación debe verse en pantalla** el **valor del reloj**
13. **Al finalizar la simulación** debe desplegarse en pantalla
    - **El contenido de la memoria** (las 200 direcciones y su contenido)
    - el **contenido de los 32 registros** del procesador
    - la **cantidad de ciclos** que tardó su ejecución

III. Forma de realizar y entregar la tarea:

- Debe **trabajarse ya con el grupo para realizar el proyecto**, el cual puede tener **2 ó 3** estudiantes como máximo.
- El programa de simulación se puede realizar en el **lenguaje de alto nivel que el grupo de trabajo escoja**, pero este lenguaje debe permitir manejar muy bien el paralelismo en el nivel de hilos.
- El programa debe utilizar **arquitecturas paralelas**, es decir, es **indispensable** que se diseñe utilizando hilos que realmente puedan correr en paralelo y logren no solo aprovechar el paralelismo que permiten los procesadores de al menos doble núcleo sino que también permitan hacer una simulación más "real".
- Debe hacerse la **correcta sincronización** entre hilos
- El día de entrega se envía por correo electrónico **el código de su simulación** y un "**screenshot**" de la pantalla con los resultados después de correr el hilo de prueba que se le indicará más adelante. (Nota importante: si los resultados para este hilo son los esperados, no significa que automáticamente su nota es 100. Razones: este hilos no necesariamente prueba todo, debe revisarse en la documentación y en el código la lógica utilizada para la solución de los problemas, las estructuras de datos, la eficiencia de la programación, etc.