ECCI-UCR **II SEMESTRE, 2015, Nov**
CI-1323 ARQUITECTURA DE COMPUTADORAS **Prof. Ileana Alpízar A.**
Para entregar K 17 de Nov antes de media noche

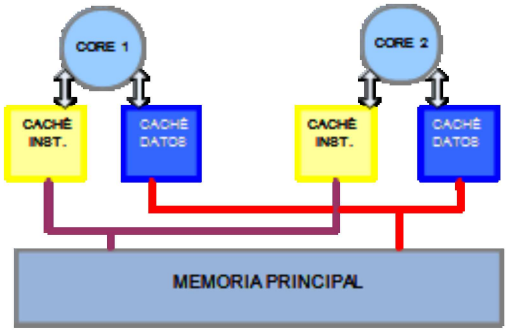
PROYECTO PROGRAMADO PARTE FINAL (valor 12%)

Diseño y programación de la simulación de un procesador MIPS para enteros doble núcleo, con la capacidad de ejecutar hilos de un mismo proceso utilizando memoria compartida centralizada

1. La última parte de este proyecta incluye:

- La solución de los problemas de la primera parte.
- Programar la parte de la simulación diseñada en la II parte, es decir, todo lo relacionado con los accesos a memoria de datos y la ejecución de instrucciones para hacer sincronización: ejecución de las instrucciones LW, SW, LL Y SC, cachés de datos, memoria de datos, coherencia de caché con el protocolo snooping e invalidación para escrituras.

Se completará el procesador doble núcleo, con las estructuras de memoria de datos y el bus correspondiente, así como el manejo de las instrucciones de acceso a memoria:



1.1. **Los bloques de memoria** y de cachés son de **4 palabras**, cada una de **4 bytes**. Tanto las cachés de instrucciones como las cachés de datos tienen capacidad para **8 bloques** y son de **mapeo directo.** Cada **instrucción MIPS codificada es de una palabra de longitud.**

1.2. Las cachés de datos tienen la estrategia **"Write Back"** para hit de escritura (escribe solo en caché, y la actualización a memoria se hace **o cuando** se vaya a **reemplazar el bloque**, **o cuando se debe copiar a otra caché** para resolver un fallo de caché para ese bloque) y la estrategia **"Write allocate"** para fallos de escritura, es decir se sube el bloque a caché (Se trae de otra caché si está modificado ese bloque en esa otra caché, o de memoria principal en cualquier otro caso). Tanto en la caché de instrucciones como en la de datos, y en memoria principal, los **bloques son de 4 palabras**.

1.3. La **memoria principal** contiene **128 bloques** (128 *4 = **512 palabras, o sea 2048 bytes**). Para simplificar la labor del "sistema operativo, el cual en la simulación no será otro más que parte de su mismo programa" se va a suponer que en esta memoria **del bloque 0 al 39** solo se podrán almacenar instrucciones (es decir los hilos que correrán en su simulación) y **a partir del bloque 40** (**dirección 640**) se tendrá el **área de datos compartida para ambos núcleos. No se necesita simular la parte de memoria que no será parte de la memoria compartida.**

1.4. **La lectura o escritura de una palabra en memoria tarda m ciclos (latencia de memoria)**. Para efectos de la simulacinó, el valor de **m** será dado como un dato inicial del usuario.

1.5. Tanto el **bus de datos como el de instrucciones** son de una palabra de ancho y tardan "**b**" ciclos para transferir una palabra de memoria a caché o viceversa. El valor de "b" se le debe solicitar al usuario al inicio.

1.6. Cada núcleo tiene **32 registros de propósito general: R0, R1, … R31**. El registro R0 siempre tendrá asignado el valor cero y no puede utilizarse como registro destino para operación alguna. Y el registro **RL** para enlace para las instrucciónes LL y SC. El "contexto" entonces para un hilo estará compuesto por R0 ...R31, RL y el PC.

1.7. **La sincronización** entre hililos **se realiza solo mediante semáforos binarios** (mutex o locks) los cuales permiten como máximo a un hilo el ingreso a una sección crítica (exclusión mutua). Estos mutex se implementarán tan solo usando dos instrucciones que **agregaremos** al MIPS: **Load linked (LL) y el Store Conditional (SC),** instrucciones que serán **incluidas como parte del código de los hilos** que correrán en su procesador, para lograr la sincronización. **Muy diferente** a como usualmente ocurre en la realidad, ya que cuando en un programa se ha incluido una herramienta de sincronización, cuando ésta debe ejecutarse, lo que sucede es que se da una interrupción y se hace un llamado a una rutina del sistema operativo, la cual es la que realiza la sincronización, utilizando instrucciones como el LL y SC.

Se define que un valor **0 en el "lock o mutex" indica que está disponible, y un 1 que está ocupado**

2. Características que debe tener la simulación para esta parte final:

2.1. Por claridad y para evitar errores, debe **inicializarse las cachés con ceros** en cada entrada y con cada "bloque" inválido". La **memoria principal también debe inicializarse, PERO con UNOS.** Esto para inicializar en "cerrados" los candados que así se necesiten.

2.2. "**Recursos críticos"** (En todos estos casos debe esperarse a que se libere el recurso, y debe contabilizarse este tiempo):

2.2.1. No es posible usar un **bus para ir a memoria** si el otro núcleo lo está utilizando. En todos esos casos debe esperarse a que se libere el recurso.

2.2.2. En el caso particular de tener que **invalidar un bloque en una caché porque se va a modificar en la otra caché,** debe hacerse hasta que finalice el ciclo de reloj actual y antes de que inicie el siguiente para no invalidar algo que se está utilizando en ese momento. Es decir, una invalidación de bloque "**rige" hasta para el siguiente ciclo de reloj**.

Note que el momento de invalidación de un bloque es uno de los dos momentos en los que el **registro RL** pueda necesitar ser puesto en **-1** (el otro momento, y acá sí es de fijo, es cuando finaliza el quantum de un hilo)

2.3. Cuando un hilo está "gastando" su **quantum** de tiempo utilizando el procesador, pueden darse 3 situaciones que afectan el control: una, que se produzca una interrupción para realizar una operación de E/S; otra, que el hilo termine por completo su ejecución; y la última situación posible es que acabe su quantum pero que aún necesite más tiempo de procesador, en este caso el hilo pasará al final de la cola que maneja el sistema operativo a esperar por su próximo turno de tiempo de procesador. **Por simplicidad, se trabajará sin tomar en cuenta la primera situación, es decir, obviando las interrupciones para E/S -** que en todo caso no tenemos.

2.4. Características del conjunto de instrucciones del procesador a simular. El procesador MIPS ejecutará apenas un subconjunto de todas las aprox 90 instrucciones MIPS. Además se le agregan instrucciones LL y SC. Se debe respetar las **reglas de alineación**. Las instrucciones serán codificadas en formatos de longitud fija de **4 enteros en decimal**.

Subconjunto de instrucciones MIPS que se deben implementar			**CODIFICACIÓN**			
			1	**2**	**3**	**4**
Operación	**Operandos**	**Acción**	**Cód. Op.**	**Rf1**	**Rf2 ó Rd**	**Rd ó inmediato**
DADDI	RX, RY, #n	Rx <-- (Ry) + n	8	Y	X	n
DADD	RX, RY, RZ	Rx <-- (Ry) + (Rz)	32	Y	Z	X
DSUB	RX, RY, RZ	Rx <-- (Ry) - (Rz)	34	Y	Z	X
DMUL	RX, RY, RZ	Rx <-- (Ry) * (Rz)	12	Y	Z	X
DDIV	RX, RY, RZ	Rx <-- (Ry) / (Rz)	14	Y	Z	X
LW	RX, n(RY)	Rx <-- M(n + (Ry))	35	Y	X	n
SW	RX, n(RY)	M(n + (Ry)) <-- Rx	43	Y	X	n
BEQZ	RX, ETIQ	Si Rx = 0 SALTA	4	X	0	n
BNEZ	RX, ETIQ	Si Rx <> 0 SALTA	5	X	0	n
JAL	n	R31<--PC, PC<-- PC+n	3	0	0	n
JR	RX	PC <-- (Rx)	**2**	X	0	0
LL	RX, n(RY)	Rx <-- M(n + (Ry)) RL <-- n + (Ry)	**50**	Y	X	n
SC	RX, n(RY)	If RL = n + (Ry) then M(n + (Ry)) <-- Rx else Rx <-- 0	**51**	Y	X	n
FIN		Detiene el programa	**63**	0	0	0

NOTA: Tanto la aritmética como el tratamiento que se le de a cada valor en la instrucción codificada se hará utilizando instrucciones de alto nivel, no se pide simular aritmética en base 2 ó 16. (**Se usará, por simplificación, aritmética decimal**)

2.5. Descripción de detalles del funcionamiento de la simulación:

2.5.1. MUY IMPORTANTE: es necesario ofrecer **dos modalidades de ejecución**: una lenta y una rápida. Con la lenta se avanza un ciclo de reloj al oprimirse una tecla (o se introduce un delay). Con la rápida simplemente se detiene hasta que se termina de ejecutar los programas, y en ese momento se puede ver memoria, cachés y registros con sus valores finales.

2.5.2. Mientras corre la **simulación debe verse en pantalla:**

-el **valor del reloj** del procesador
- la **identificación del hilo** que está corriendo en cada núcleo.
-el contenido de **las 2 cachés de datos**.

2.5.3. Al finalizar la simulación debe desplegarse en pantalla

- El contenido de la memoria a partir de la dirección **640** (las 352 direcciones y su contenido)
- Para cada hilo que corrió:
    - el **contenido de los 32 registros y del contenido del RL** del procesador
    - la **cantidad de ciclos** que tardó su ejecución
    - el **contenido de la caché de datos final para ese hilo**

3. *Forma de realizar y de entregar el proyecto:*

3.1. Grupos de trabajo de **2 ó 3 estudiantes** como máximo quienes deben designar a un líder de proyecto (responsable de organizar el trabajo, de reportar problemas en el grupo, y de reportar el porcentaje de nota que le debe corresponder a cada integrante del grupo, dependiendo de en cuánto contribuyó al desarrollo del proyecto)

3.2. En esta última parte se debe incluir una fotocopia de la revisión de la parte 2 del proyecto (el diseño) y ese diseño ya corregido

3.3. La **documentación interna** debe dejar muy claros los algoritmos utilizados para resolver cada problema.

3.4. Se envía código fuente, ejecutable (con cambio de extensión para que no de error en gmail), un **manual muy claro de instalación** (es decir, debe explicarse los detalles de ambiente que necesita el programa para correr) y una lista de **problemas no resueltos** al tiempo de entrega e ideas sobre su solución, y una **nota** para todos y cada uno de los miembros del grupo de trabajo puesta por el líder de acuerdo con el desempeño del integrante en el desarrollo del proyecto (0 a 100). La nota del líder debe ser aprobada por el resto del grupo. Con ese valor se calculará la nota del proyecto, parte 3. correspondiente a cada uno de los integrantes del grupo como:

nota obtenida en el proyecto por el grupo * porcentaje asignado a ese integrante.