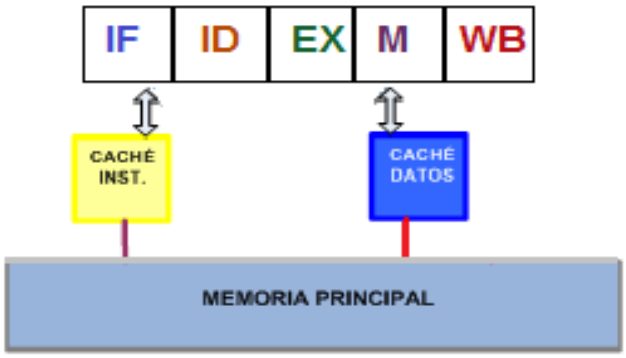
ECCI-UCR **II SEMESTRE, 2012, L 8 de Octubre**

CI-1323 ARQUITECTURA DE COMPUTADORAS **Prof. Ileana Alpízar A.**

TAREA PROGRAMADA GRANDE (valor 13%, Fecha entrega: M 31 de OCTUBRE)
Diseño y programación de la simulación de un procesador MIPS para enteros de 5 etapas y su jerarquía de memoria



1. Objetivos de esta tarea

1.1. Conocer más a fondo la **arquitectura MIPS**.

1.2. Conocer el manejo de la **jerarquía de memoria**.

1.3. Ganar experticia con la programación paralela utilizando hilos para realizar la simulación. No es importante acá la comparación de tiempo entre ejecución secuencial y paralela, sino que importa **simular un comportamiento real el cual es paralelo**.

2. Descripción del sistema que se va a simular:

2.1. El procesador es el **MIPS** de **5 etapas** visto en clase, el cual ejecutará solo un subconjunto de las instrucciones para enteros el cual se describe más abajo. Se simulará su capacidad de ejecución con solamente un programa cada vez, el cual **puede estar compuesto por varios hilos.**

2.2. El procesador tiene una **caché de datos y una caché de instrucciones.** Hay dos puertos para memoria principal uno para el bus para la caché de instrucciones y otro para el bus para la caché de datos.

2.3. Los bloques de memoria y de cachés son de **2 palabras**, cada una de **4 bytes**. Tanto la caché de instrucciones como la caché de datos tienen capacidad para **4 bloques** y son de **mapeo directo.** Cada **instrucción MIPS codificada es de una palabra de longitud.**

2.4. La caché de datos tiene la estrategia **"Write Through"** para hit de escritura (escribe en caché y en memoria, y la estrategia **"NO Write allocate"** para fallos de escritura, es decir se envía el dato a escribir a memoria.

2.5. La **memoria principal** contiene **256 bloques** (256 * 2 = **512 palabras**). Para simplificar suponga que en esta memoria **del bloque 0 al 79** solo se podrán almacenar instrucciones (es decir los hilos

que correrán en su simulación) y **a partir del bloque 80** (**dirección 640**) se tendrá el **área de datos.**

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

2.7. Las direcciones de memoria deben cumplir con las **reglas de alineación**, aunque no se simulará el acceso a memoria activando filas.

2.8. 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. No es necesario que se simulen los buses, tan solo se debe tomar en cuenta el tiempo de transferencia de datos e instrucciones.

2.9. El procesador 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 instrucciones LL y SC. El "contexto" entonces para un hilo estará compuesto por R0 ...R31, RL y el PC.

2.10. La **asignación de la CPU** para los hilos del programa se realiza con un algoritmo de planificación por turno rotatorio **(round robin**) con un **quantum de X** ciclos de reloj para cada "hilo". Para efectos de la simulación, el valor del quantum será dada como un dato inicial del usuario. **Cada vez que se va a realizar la lectura de una instrucción (fetch) se verifica si este es el último ciclo del quantum**, en cuyo caso se debe leer esa instrucción, pero se detiene la lectura de las instrucciones posteriores de ese hilo. La instrucción recién leída, junto con las que ya están en el pipeline se deben dejar finalizar (aunque se tengan fallos de caché a partir de ese momento). Si se da **un fallo de caché o si se detiene una instrucción en espera de "forwarding" para resolver un conflicto de datos antes del último ciclo del quantum, aunque los ciclos del reloj deben seguir avanzando, el quantum no se debe afectar.** No tiene sentido sacar de ejecución un hilo que está en medio de una solución de un fallo de caché. Si se lee la **instrucción correspondiente al fin del hilo**, entonces, independientemente del quantum, esa será la última instrucción leída, y una vez que haga WB, se finaliza la ejecución de ese hilo.

2.11. **La sincronización** entre los hilos que correrán en su simulador **se realiza solo mediante semáforos binarios** (mutex o locks) los cuales permiten o no 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)** dos "primitivas" que permiten hacer **espera activa**[1] para entrar a sección crítica y trabajan juntas de manera atómica. Se define que un valor **0 en el "lock o mutex" indica que está disponible, y un 1 que está ocupado**

3. Características que debe tener la simulación:

3.1. **El programa debe hacerse utilizando hilos,** los cuales deben sincronizarse correctamente con las herramientas que brinde el lenguaje de programación elegido. **Cada etapa del pipeline debe implementarse con un hilo.**

[1] Esta es una sobre-simplificación de la forma normal en la que se manejan los semáforos por el sistema operativo, el cual crea una cola para cada semáforo con los hilos que lo esperan.

3.2. Las estructuras de datos deben simular las indicadas, pero no se almacenarán los datos ni las instrucciones en código binario, se usarán valores decimales. Debido a que las instrucciones MIPS son de 4 bytes, pero dado que para la simulación los programas de prueba contienen instrucciones MIPS codificadas como un arreglo de 4 enteros cada una, ud puede usar **dos estructuras de datos para simular la memoria principal, una para instrucciones** que abarcaría del bloque 0 al 79 y en donde cada entrada de una palabra puede almacenar los 4 enteros que representan una instrucción MIPS en lenguaje de máquina **y otra estructura de datos para los datos**, que abarcaría del bloque 80 al 255 - dirección de memoria inicial: 640- en donde cada entrada almacenaría un dato (un entero) pero cuya dirección será un múltiplo de 4 para respetar no solo las reglas de alineación para las direcciones de memoria, sino también el tamaño de los datos. Como se dijo anteriormente, no se simulará el acceso a memoria activando filas, ud. puede definir por ejemplo, un vector de enteros para "las memorias". Por simplicidad se va a trabajar **suponiendo que las direcciones virtuales que da la CPU para pedir instrucciones o datos corresponden a su dirección física.**

3.3. Al inicio

- **al usuario se le solicita el número de hilos a correr en el procesador y su ubicación.** Cada hilo a ejecutar estará escrito en un archivo de texto en donde cada línea es una instrucción MIPS ya codificada en lenguaje de máquina de acuerdo a como se indicará más adelante. (Su programa debe **ofrecer la facilidad de localizar el directorio** en el que se encuentran dichos archivo)
- su programa debe **leer los hilos que el usuario indicó que desea correr** (se asume que todos pertenecen a un mismo proceso), y los debe colocar en la memoria simulada para instrucciones, guardando **el PC** para cada uno de ellos.
- **Debe solicitar al usuario los valores de "m"** (número de ciclos que se tardan leyendo o escribiendo una palabra en memoria principal), de "**b**" (número de ciclos de reloj que tarda cada bus transfiriendo una palabra: sea ésta un dato, una dirección de memoria o una instrucción) y de "**X"** valor del quantum. Esos valores regirán para esa corrida.
- Se comienza asignando el hilo1 a la CPU, el cual hace las veces de **hilo principal** del programa que correrá (esto para efectos de cerrar o activar candados).

3.4. Para resolver conflictos de datos se debe utilizar "forwarding" y para resolver conflictos de control debe detenerse el ingreso de instrucciones

3.5. Todas las etapas deben iniciar con el inicio de ciclo de reloj. Note que no es posible definir una longitud de ciclo de reloj. En realidad para poder avanzar de ciclo, deben haber concluido todas las etapas de hacer lo que les correspondía realizar en ese ciclo de reloj. Si en la etapa se da un fallo de caché, o se detiene por conflicto de datos o branches, igual esta etapa debe "avisar" que por "ella" ya se puede avanzar al siguiente ciclo de reloj.

3.6. 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 con ceros** por claridad.

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

3.8. Características del conjunto de instrucciones del procesador a simular. El procesador MIPS ejecutará apenas un subconjunto de todas las instrucciones MIPS. Además se le agregan instrucciones que no

son propias del MIPS (LL y SC, las cuales se explican más adelante). Se debe respetar las reglas **de alineación**. Las instrucciones serán codificadas en formatos de longitud fija de 4 enteros en decimal.

Descripción del subconjunto de instrucciones MIPS que se deben implementar y su codificación.

INSTRUCCIÓN			**CODIFICACIÓN**			
			1	**2**	**3**	**4**
Operación	**Operandos**	**Acción**	**Cód. Operación**	**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)	**11**	Y	X	n
SC	RX, n(RY)	If RL = n + (Ry) then M(n + (Ry)) <-- Rx else Rx <-- 0	**22**	Y	X	n
FIN		Detiene el programa	**63**	0	0	0

Ejemplo de un programa muy muy simple y su codificación: (no lo use de prueba)

INSTRUCCIONES EN MIPS			**CODIFICACIÓN**
	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**
	JR	R6	**2 6 0 0**
	DSUB	R1, R2, R3	**34 2 3 1**
	BEQZ	R5, ET-1	**4 5 0 1**
	DMUL	R3, R6,R7	**12 6 7 3**
ET-1	DDIV	R27, R28, R5	**14 28 5 27**
	BNEZ	R4, ET-2	**5 4 0 -7**
	DSUB	R26, R4, R3	**34 4 3 26**
	JAL	8	**3 8 0 0**
	DADDI	R9, R0, #--500	**8 0 9 -500**
	FIN		**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**)

3.9. Descripción de detalles del funcionamiento de la simulación:

3.9.1. 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. Con la rápida simplemente se detiene hasta que se termina de ejecutar los programas, y en ese momento se puede ver memoria, acumulador y registros con sus valores finales.

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

-el **valor del reloj** del procesador
- la **identificación del hilo** que está corriendo
-el contenido de **las 2 cachés**

3.9.3. **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

4. Forma de realizar y de entregar la Tarea Programada Grande:

4.1. Grupos de trabajo de **2 ó 3 estudiantes** como máximo quienes deben designar a un líder (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 de la Tarea Programada Grande)

4.2. 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 que se indicó.

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

4.4. La **documentación externa** debe-como mínimo- estar compuesta por:

4.4.1. Descripción de la **lógica de la simulación** (use un diagrama de bloques-o de actividades) Se describe la simulación completa a grandes rasgos, y luego una descripción de la lógica implementada en el programa para cada una de estos operaciones:

-cuando se va a hacer una lectura (un LW) o sea cómo es la lógica **para resolver todos los casos que se pueden dar** y los trabajos que se hacen en el programa para implementar una buena ejecución de un LW

-cuando se va a hacer una escritura (un SW) idem a los solicitado para el LW

-Cómo se maneja el paso de los ciclos cuando hay ciclos de retraso (fallos de caché, invalidación de bloques, revisión de directorios, etc)

4.4.2. **Descripción de los hilos** que se usarán para la simulación, describiendo la lógica global de cada uno y los métodos de sincronización utilizados

4.4.3. Un "**screenshot**" de la pantalla con los resultados después de correr los hilos de prueba que se le indicarán más adelante. (**Nota importante**: si los resultados para estos hilos son los esperados, no significa que automáticamente su nota es 100. Razones: estos hilos no necesariamente prueban 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.

4.5. El día de entrega o antes, se envía por correo electrónico a arqui.gx@gmail.com (con x =1 ó a 2) lo siguiente:(uno por grupo)

4.5.1. El **fuente** y el **ejecutable** del programa (**debe hacerse el ejecutable. Cambiarle la extensión .exe por .ajo**) El código debe venir con una completa y clara documentación interna.

4.5.2. La **documentación externa**

4.5.3. Un **manual muy claro de instalación** (es decir, debe explicarse los detalles de ambiente que necesita el programa para correr)

4.5.4. Lista de **problemas no resueltos** al tiempo de entrega e ideas sobre su solución

4.5.5. 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 de la tarea (0 a 100). La nota del líder debe ser aprobada por el resto del grupo. Con ese valor se calculará la nota correspondiente a cada uno de los integrantes del grupo como:

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

5. CALIFICACIÓN: La nota final de la tarea se calcula así:

Puntaje:	Documentación	10%	
	Diseño y lógica	**50%**	**(a revisar en doc, en código, y en corrida)**
	Programación correcta	40%	