Proyecto evaluación continua 2: Jerarquía de memoria de datos

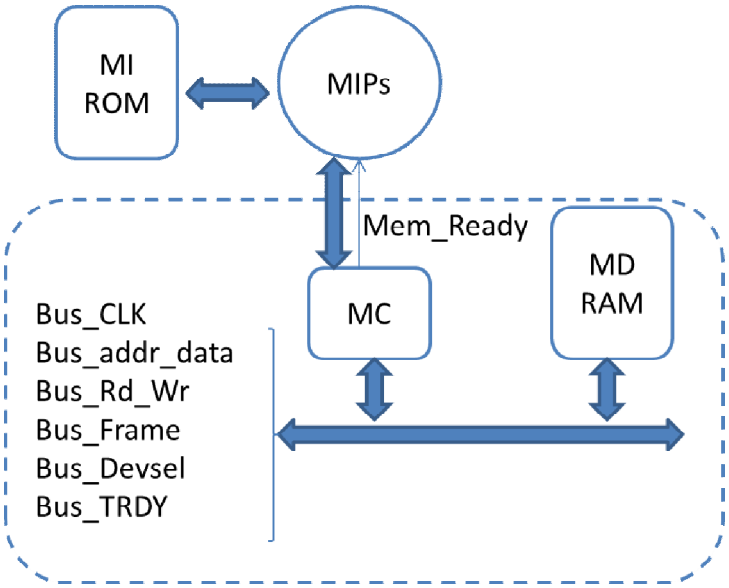
Fecha de entrega: 3 de Junio

Resumen

En este proyecto vamos a introducir una memoria cache de datos conectada a través de un bus semi-síncrono a la memoria principal. El objetivo es diseñar el controlador de la memoria cache que se ocupará de gestionar las peticiones del procesador realizando las transferencias que sean necesarias. Además, debéis incluir en vuestro procesador soporte para que se detenga cuando la memoria cache no pueda realizar la operación solicitada en un ciclo de reloj. Finalmente deberéis incluir tres contadores en vuestro diseño que se activen respectivamente cuando haya una parada debido a un riesgo de datos, a un riesgo de control, o a un fallo en memoria cache. De esta forma al ejecutar un código podremos evaluar qué factores están limitando nuestro rendimiento.

Detalles del sistema a diseñar:

En este proyecto vamos a sustituir la memoria de datos del MIPS por el componente MD_mas_MC. Este componente incluye la memoria cache, la memoria de datos y un bus síncrono que las conecta. El interfaz con el MIPS es idéntico salvo por una señal nueva: *Mem_Ready*. *Mem_Ready* vale 1 cuando la MC puede realizar la operación solicitada en el ciclo actual, en caso contrario vale 0. El esquema con las señales del bus se puede ver en la siguiente figura:



El bus de memoria: Es un bus semi-síncrono basado en el bus PCI explicado en clase, en el que MC es siempre el máster y MD siempre el Slave. Las **líneas de datos y direcciones están multiplexadas**. El bus soporta ráfagas de tamaño variable.

- **Sincronización**
  - Al comenzar una transferencia, el máster activará en primer lugar la señal *Bus_Frame* y enviará la dirección (líneas *Bus_addr_data*) y el tipo de operación (*Bus_Rd_Wr = '0' para lecturas y = '1' para escrituras)*. El máster mantendrá la dirección hasta que el esclavo active la señal Bus_Devsel. A partir del ciclo siguiente el máster retirará la dirección y el esclavo activará la señal *Bus_TRDY* (target_ready) y recibirá o enviará un dato por ciclo a través de *Bus_addr_data* hasta que el máster desactive la señal *Bus_Frame.* Si en

un ciclo dado el esclavo no es capaz de mandar/enviar el dato que le toca desactivará la señal *Bus_TRDY* (target_ready) para indicar al máster que debe esperar. Los datos que se envían /reciben por el bus serán siempre consecutivos. Por ejemplo si el máster pide la palabra 4, el esclavo le dará la 4, la 8, la 12....hasta que el máster baje la señal *Bus_Frame*. Cuando esto ocurra la transferencia habrá finalizado. **Nota**: *Bus_Frame* debe estar al menos un ciclo a 0 entre dos transferencias porque de otro modo el esclavo no tiene modo de diferenciar una transferencia de la otra.

- **Arbitraje**: Como sólo hay un máster no es necesario arbitraje. MC puede usar el bus cuando quiera.

MC:

La memoria cache está compuesta de 4 bloques de 4 palabras de 4 bytes con: **emplazamiento directo**, **escritura con actualización inmediata (write-through),** y la política ***write_around*** en fallo de escritura.

Controlador MC:

Recibe las solicitudes del MIPS y si puede las responde con los datos almacenados en MC. En caso contrario realiza las transferencias necesarias.
Importante: aparece **un nuevo riesgo estructural** que hay que gestionar. Si el MIPS ejecuta la etapa MEM de un lw o sw y el controlador desactiva *Mem_Ready*, el MIPS tendrá que detener la ejecución (pensad qué etapas deben detenerse) hasta que la operación de la etapa MEM se pueda realizar.

Controlador Memoria de datos:

Siempre actúa como esclavo, trasladando las solicitudes del bus a la Memoria de datos (MD) si están dentro de su rango (X"00000000"-X"000001FF"). Las direcciones fuera de su rango se ignoran.
Temporización: La MD que vamos a utilizar es la misma que teníamos previamente pero con retardos adicionales. Cuando no pueda enviar un dato en el ciclo actual desactivará la señal *Bus_TRDY* para que el controlador sepa que debe esperar.
Modo ráfaga: cuando *Bus_Frame* esté activado la memoria de datos realizará la operación solicitada con la dirección inicial, y después la incrementará internamente.

Contadores de rendimiento: Queremos ayudar a los programadores a entender qué factores afectan al rendimiento de sus códigos. Para ello vamos a utilizar varios contadores, tanto en el MIPS como en la MC, y debéis activar sus señales de cuenta cuando corresponda:

1. Ciclos totales: cuenta siempre y contabilizará todos los ciclos de ejecución. Salida: *ciclos*.
2. Contadores de paradas: contabilizarán las paradas del procesador distinguiendo entre las causadas por riesgos de datos, control, y las causadas por el acceso a la memoria de datos. Salidas: *ciclos*, *paradas_control*, *paradas_datos*, y *paradas_memoria*.
3. Contador de referencias a memoria. Indican el número de ld y sw ejecutados. Salidas: *mem_reads, mem_writes*.
4. Contadores de eventos en la memoria cache. Indicarán los fallos de lectura y los fallos y aciertos de escritura. Salidas: rm, wh y wm.

Resultados y memoria de la práctica

En primer lugar, **debéis comprobar que el diseño funciona correctamente para todos los casos posibles** (fallos y aciertos de lectura y escritura, utilizando los cuatro conjuntos y haciendo algún reemplazo), y que los contadores de paradas cuentan cuando les corresponde. Para ello debéis definir esos casos en un banco de pruebas. Os damos ejemplos en los que se prueban algunos casos, pero debéis añadir vosotros mismo el resto y **describir vuestro banco de pruebas en la memoria** explicando qué casos cubre.

En la memoria debéis explicar también brevemente vuestro diseño. Hay que incluir **el diagrama de estados** de la unidad de control, explicando qué se hace en cada estado y qué señales se activan.

También debéis realizar una **descripción algorítmica** de vuestra MC siguiendo el modelo explicado en clase, indicando qué eventos gestiona y los ciclos que necesita para hacerlo. Podéis obtener los retardos de la memoria de datos a partir de una simulación. Los retardos deben de consignarse como en los problemas de clase. Por ejemplo: en caso de fallo de lectura la penalización será CrB(MP) +1, dónde CrB(MP) = N ciclos (para la primera palabra) +3* M ciclos (para las palabras restantes).

A partir de este algoritmo indicad la fórmula de los **ciclos efectivos** (ciclos medios por acceso a memoria).

Finalmente debéis incluir las **conclusiones** y **tiempo dedicado** por cada componente del equipo y una **autoevaluación** en la que debéis contestar **de forma individual** a dos preguntas**:** ¿Crees que has cumplido los objetivos de la asignatura? ¿Qué nota te pondrías si te tuvieses que calificar a ti mismo?

Criterios de corrección:

Durante el desarrollo de vuestro proyecto debéis de respetar los nombres de los componentes y señales principales proporcionados en el código o indicados en la memoria. El cambio de nombre de componentes y señales nos retrasa la evaluación del proyecto, y en consecuencia la penalizaremos a no ser que esté justificado y explicado en la memoria.

Como en la práctica anterior el diseño debe funcionar correctamente para aprobar. Y además debéis incluir en la memoria un **diagrama de estados claro** que podamos seguir. La omisión del diagrama de estados, o un diagrama de estados confuso o mal concebido supone la invalidación del proyecto.

Valoración de vuestro controlador: **El principal factor** al evaluarlo será que el diagrama de estados **sea claro, esté bien explicado y presentéis un banco de pruebas que cubra los distintos eventos**. También se evaluará que el controlador sea eficiente y genere el mínimo número de ciclos de parada. Y que vuestra descripción y fórmula de los ciclos efectivos coincida con la realidad. Os recomendamos partir de un esquema sencillo que os resulte fácil entender y después si os da tiempo tratar de mejorar su rendimiento. Algunas técnicas que podéis usar son:

- Eliminar estados que generan retardos y pueden fundirse con otros.
- Máquina Mealy en lugar de Moore para reducir un ciclo el tiempo de respuesta
- **Opcional**: Incluir un **buffer para las escrituras en memoria**. Cuando haya que escribir en memoria se almacena el dato y la dirección en dos registros y la MC continúa gestionando aciertos mientras la Unidad de Control gestiona la escritura a través del bus. Si hay que hacer una segunda transferencia en el bus y no se ha terminado de gestionar la anterior habrá que parar. Razonar en la memoria cuándo puede suponer una mejora en rendimiento.
- **Opcional**: **Adelantar el envío** de la palabra solicitada. Cuando el controlador gestione un fallo de lectura lo más sencillo es traer el bloque entero y después darle la palabra solicitada al procesador. Sin embargo, se puede acelerar dando al procesador la palabra solicitada tan pronto como llegue. Así el procesador puede continuar mientras el controlador termina de traer el resto de palabras del bloque. Razonar en la memoria cuándo puede suponer una mejora en rendimiento.

Otro factor importante para la evaluación es la calidad de la memoria. Debe estar correctamente organizada, con un índice que permita localizar los apartados a incluir, mencionados anteriormente.

Fuentes y memoria (pdf) deben de entregarse en una carpeta (sin anidación) comprimida (.zip) Las cuestiones durante la defensa presencial del proyecto pueden abarcar tanto aspectos prácticos (sobre el código etc) como conceptos teóricos relacionados (memorias cache, segmentación, buses).

Anexo 1: ¿Cómo incluyo los nuevos fuentes en mi MIPS?

Debéis sustituir el componente del MIPS memoriaRAM_D por el nuevo componente MD_mas_MC. El interfaz es el mismo pero añadiendo la señal *Mem_ready* y el *reset.* Hay una explicación detallada junto a los fuentes *(Cambio en el MIPS.txt)*. Después incluid todos los fuentes nuevos en el proyecto.

Anexo 2: ¿Cómo interactúo desde el MIPS con el controlador de la MC?

El controlador responderá a las instrucciones de lectura y escritura en memoria ocupándose de que se realice todo correctamente. El MIPS debe estar atento a la señal *Mem_ready*. Si *Mem_ready* vale 1 todo se debe ejecutar como en el MIPS anterior, si vale cero el MIPS debe detenerse y esperar.

Anexo 3: Estrategia de depuración

Para acelerar la depuración y entender mejor vuestro diseño es importante depurar en cada nivel en el que trabajéis. Es difícil ver los errores de vuestra unidad de control depurando sobre el procesador completo. En lugar de eso es recomendable hacer un banco de pruebas para cada nivel. Por ejemplo primero comprobad que la máquina de estados funciona como pensáis y genera las salidas correctas. Después comprobad que al unirla al resto de la MC sigue funcionando correctamente. En la siguiente prueba comprobad distintas transferencias en el componente MD_mas_MC. Y sólo cuando hayamos comprobado que cada parte funciona lo integraremos en el MIPS. Parece más trabajo, pero ir poco a poco es la clave para depurar eficientemente.

En los fuentes incluimos dos bancos de prueba como ejemplo. Uno para ver cómo funciona la nueva MD (y poder observar sus retardos) y otro para simular la MC con el bus y la MD sin incluir el MIPs. Para la comprobación final del sistema completo se usa el mismo test que en el proyecto anterior. La forma de generar fallos y aciertos será a través de instrucciones lw y sw.

Anexo 4: Estimación del tiempo dedicado

Esta es nuestra estimación:

- Estudio de los fuentes: 2 horas
- Diseño inicial de la unidad de control: 2 horas
- Depuración y ajustes: 16 horas
- Memoria: 3 horas

Cuando nos deis los datos de tiempo dedicado por favor comparaos con esta estimación y analizar las divergencias. La utilidad de vuestro análisis dependerá de que registréis bien los datos. En otras palabras tratad de ser profesionales medid los tiempos con precisión.