Proyecto evaluación continua 2: Jerarquía de memoria de datos

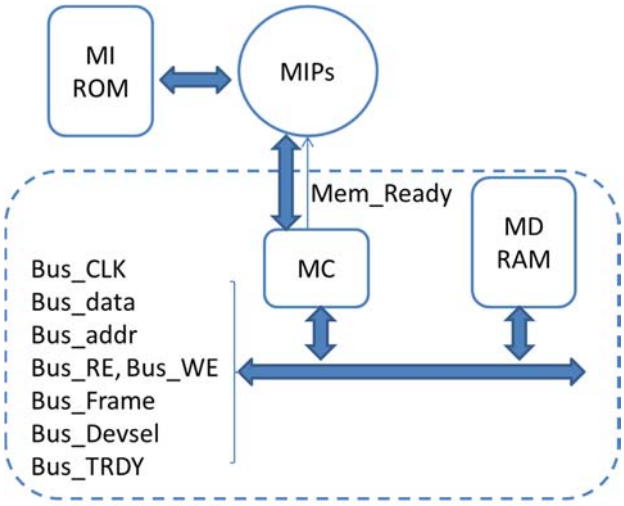
Fecha de entrega: 1 de Junio

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 detenerse 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 PCI en el que MC es siempre el máster y MD siempre el Slave. El bus soporta ráfagas de tamaño variable.

- **Sincronización**
  - Al comenzar una transferencia en primer lugar el máster activará la señal *Bus_Frame* y enviará la dirección (líneas *Bus_addr*) y el tipo de operación (*Bus_RE se activa en las lecturas y Bus_WE en las escrituras)*. El máster mantendrá la dirección hasta que el esclavo active la señal Bus_Devsel. A partir del ciclo siguiente el esclavo activará la señal *Bus_TRDY* (target_ready) y recibirá o enviará un dato por ciclo a través de *Bus_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 dado que sino el esclavo pensaría que son la misma.

- **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 datos con: **emplazamiento directo**, **escritura retardada (copy-back),** y la política convencional en fallo de escritura (*fetch on write miss*).

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 debéis incluir cuatro contadores (componente *counter*) en el sistema que contabilicen las paradas del procesador distinguiendo entre las causadas por riesgos de datos, control, FP, y las causadas por el acceso a la memoria de datos. Los contadores deben estar en el MIPS. Para facilitar la corrección llamar a la salida de los contadores: *paradas_control*, *paradas_datos*, paradas_FP, *paradas_memoria*.

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.

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:

Como en la práctica anterior el diseño debe funcionar correctamente para aprobar. Y además debéis presentar en la memoria un **diagrama de estados claro** que podamos seguir. En caso contrario no lo corregiremos.

Valoración de vuestro controlador: **El principal factor** al evaluarlo será que el diagrama de estados **sea claro y esté bien explicado**. También se evaluará que el controlador sea eficiente y genere el mínimo número de ciclos de parada. 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
- **Avanzado**: Incluir un **buffer para los bloques sucios**. Cuando se deba reemplazar un bloque sucio el esquema más sencillo es primero escribirlo en MD y después traer el bloque nuevo a MC. Un esquema alternativo, similar al que usamos en una clase de problemas, es almacenar el bloque sucio en un buffer (su dirección y el contenido del bloque a expulsar), y a continuación traer el bloque que ha pedido el procesador y enviarle la palabra solicitada. Después se enviaría el bloque sucio a memoria, y la MC funcionaría de forma normal a no ser que hubiese que expulsar otro bloque sucio antes de terminar con el primero.
- **Avanzado**: Incluir un **buffer para fallos de escritura.** Si fallamos en escritura, en lugar de parar la ejecución hasta que se traiga el bloque y se escriba, podemos guardar la dirección y el dato a escribir en un buffer y decirle al procesador que puede continuar. La MC traerá el bloque y lo actualizará mientras el procesador continúa con su ejecución normal. Para esta opción hay que pensar: ¿qué ocurre si hay un segundo fallo de escritura mientras la MC está gestionando el primero?, ¿qué pasa si el procesador pide el dato que acaba de mandar escribir y la MC todavía no está actualizada?.
- **Avanzado**: **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.

Nota: las mejoras son optativas, si alguien las hace debe explicar en qué casos mejoran el rendimiento.

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*. 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 hay que hacer un banco de pruebas para cada nivel. 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 un banco de ejemplo en el que se trabaja con MC, MD y el bus sin incluir el MIPs.

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 y no os inventáis un número al acabar la práctica.