AOC 2

Memoria proyecto 2

2018/2019

Jesús Aguas Acín -- 736935@unizar.es
Daniel Barceló Labuena -- 737070@unizar.es

Índice

1. Introducción

Para el desarrollo de este proyecto se utilizará el entorno de simulación ModelSim y el lenguaje de descripción de hardware VHDL.

Como punto de partida, se utilizarán los fuentes en VHDL del MIPS segmentado utilizado en el proyecto anterior, junto con los fuentes de la memoria de datos y la memoria caché, que se encuentran en Moodle.

- El primer paso será estudiar los ficheros proporcionados, para entender todas las señales que se utilizan y qué papel desempeñan en el intercambio de información entre la caché y la memoria de datos.
- El siguiente paso será completar los ficheros para que funcionen correctamente y cumplan con los objetivos (**3. Objetivos**), el diseño utilizado para lograr los objetivos se describe en el apartado **4. Diseño**.

2. Objetivos

En el **Proyecto** se establecen como objetivos:

- Diseñar el controlador de la memoria cache que se ocupará de gestionar las peticiones del procesador realizando las transferencias que sean necesarias.
- Detener el procesador cuando la memoria cache no pueda realizar la operación solicitada en un ciclo de reloj.
- Incluir tres contadores 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 caché.

A continuación se detalla el desarrollo, la metodología y el diseño del proyecto para cumplir estos objetivos, en el apartado **4. Diseño**.

3. Diseño

3.1 Diseño del controlador

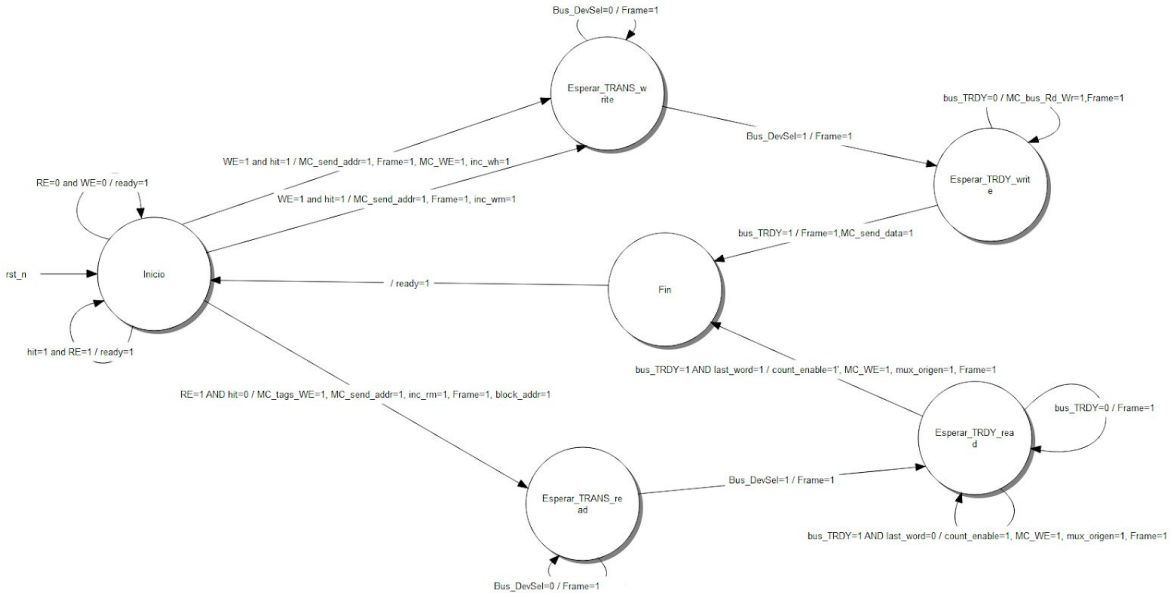


Diagrama de estados de la MC

Para desarrollar el controlador de la memoria caché, se ha empezado diseñando un autómata Mealy (para reducir un ciclo el tiempo de respuesta, ya que podemos sacar las salidas en las transiciones en vez de en los estados). El autómata se compone de seis estados, entre los que tenemos uno de inicio y otro de fin.

El autómata comienza en el estado “Inicio”, que corresponde al estado en el que la caché está lista para gestionar accesos a memoria. Cuando haya un acceso a memoria existirán 4 situaciones posibles:

- **Read hit** (rh): Ocurrirá cuando el procesador quiera hacer una lectura en memoria, y la memoria caché ya tenga disponible el contenido de esa dirección, en cuyo caso devolverá el dato y no se parará el procesador.
- **Read miss** (rm): Ocurrirá cuando el procesador quiera hacer una lectura en memoria, y la memoria caché no tenga disponible el contenido de esa dirección, en cuyo caso lo primero será detener el procesador (*ready=0*), enviar la dirección a la memoria principal (*MC_send_addr=1*), actualizar los tags (*MC_tags_WE=1*), e indicarle a la memoria de datos que queremos

realizar una transferencia (*Frame=1*). A partir de ahí tendremos que esperar a que la Memoria de Datos esté lista para transferencias (*bus_TRDY=1*) será entonces cuando la caché almacenará la palabra que le llegue (MC_WE=1) por el Bus_AD desde la memoria de datos. Tendrá que esperar a recibir 3 palabras más, será entonces cuando pasará al estado final (Fin) desde donde se le indicará al procesador que puede seguir con la ejecución normal(*ready=1*).

- **Write hit** (wh): Ocurrirá cuando el procesador quiera escribir en la memoria, y la caché ya tenga el tag de la dirección, en cuyo caso lo primero será detener el procesador (*ready=0*), enviar la dirección a la memoria principal (*MC_send_addr=1*), e indicarle a la memoria de datos que queremos realizar una transferencia (*Frame=1*). A partir de ahí tendremos que esperar a que la Memoria de Datos esté lista para transferencias (*bus_TRDY=1*), será entonces cuando la caché le pasará el dato a la memoria de datos, para que esta lo escriba.

- **Write miss** (wm): Ocurrirá cuando el procesador quiera escribir en la memoria, y la caché no tenga el tag de la dirección, en ese caso se el proceso será el mismo que en el Write hit(wh), excepto que el dato no se escribirá en la caché.

Es muy importante que cuando se estén realizando procesos de gestión entre la caché y la memoria de datos, el controlador de la caché active en todo momento el *Frame=1*, esto permite que la memoria de datos mantenga la transferencia, hasta que se indique lo contrario, en el estado Fin, con *Frame=0*.

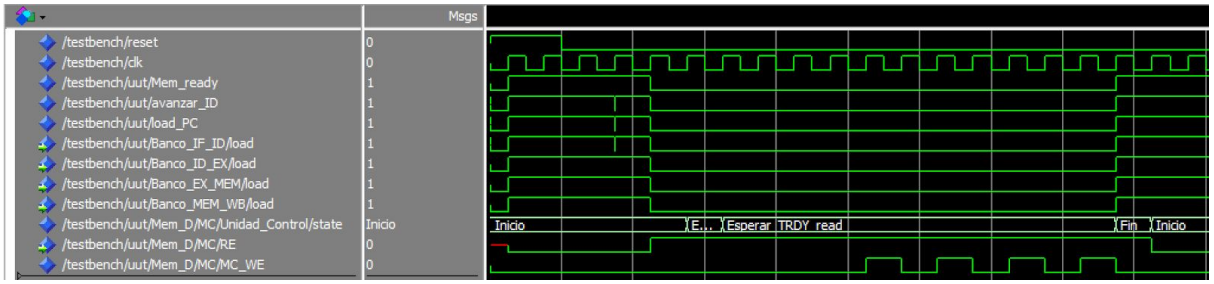
3.2 Detención del procesador

Este apartado describe los procedimientos llevados a cabo para lograr la detención del procesador MIPS cuando la memoria caché no puede realizar la operación solicitada en un ciclo de reloj.

La señal *Mem_ready* indica cuando la memoria está disponible para ser utilizada, de modo que cuando *Mem_ready=0* se debería detener el MIPS para evitar que pueda acceder a la memoria.

Para detener el MIPS será necesario detener todos los bloques secuenciales para que no actualicen su valor con el flanco de subida del reloj, para ello deberemos hacer que todas las señales de *load* de los registros dependan de que Mem_ready sea igual a 1.

Por lo que se deberán cambiar todas las señales de load de los bancos de registros de cada etapa (que antes eran constantes a '1') por la señal *Mem_ready*. Además habrá que detener la señal de load del PC, a la que actualmente le da valor la variable *avanzar_ID*. Esta variable se utilizó en el primer proyecto, e indica cuando el PC debe dejar de avanzar (cuando se producía un riesgo de datos), en este caso (*Mem_ready*=0), no debería avanzar, por lo que habrá que añadir la condición de que *avanzar_ID* será 0 si *Mem_ready*=0.



En esta imagen podemos observar que cuando ocurre un Read Miss(rm), *Mem_ready* pasa a valer 0, y por lo tanto todas las señales que actualizan los bloques secuenciales del MIPS también valen 0. Hasta que la memoria termina la transferencia y el MIPS continúa su ejecución de manera normal.

3.3 Implementación de los contadores

Contador de ciclos

Para contar el número de ciclos es suficiente con instanciar un componente contador (counter) inicializado a 0, y que incremente cada vez que hay un flanco de reloj de subida (clk=1).

Contador de paradas por riesgos de control

Para implementar el contador de paradas por riesgos de control se utiliza un contador que incrementa cada vez que hay un error de predicción.

Contador de paradas por riesgos de datos

Para implementar el contador de paradas de riesgos de datos se utiliza un contador que incrementa cada ciclo en el que hay un riesgo de beq consumidor o cuando hay un riesgo de load-uso.

Contador de paradas por acceso a memoria

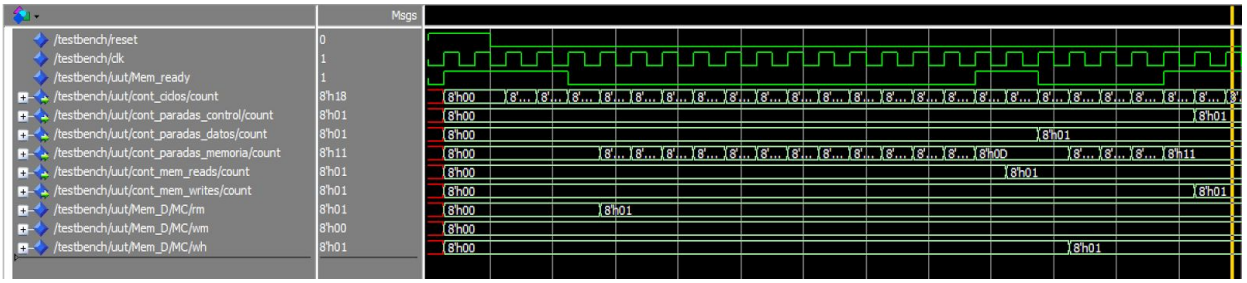
Para contar el número de paradas por accesos a memoria se utiliza un contador que incrementa cada ciclo en el que *Mem_ready=0*, ya que esto quiere decir que la memoria no está disponible para ser utilizada por el MIPS, que tendrá que ser detenido.

Contador de referencias a memoria

Para contar de referencias a memoria se utilizan dos contadores, uno para contar el número de *ld* y otro para contar el número de *sw* ejecutados, que incrementan su valor cada vez que *MemRead_MEM=1* y *MemWrite_MEM=1*, respectivamente.

Contador de eventos en la memoria caché

En este caso se utilizarán 3 contadores distintos, uno para contar el número de rm (Read Miss), otro para el número de wh (Write Hit) y otro para el número de wm (Write Miss). Todos ellos se incrementarán desde el autómata de la Memoria Caché, cuando este los detecte.



Esta imagen muestra todos los contadores citados, tras ejecutar un Read Miss(rm) - un caso load uso - un Write Miss (wm) - y un salto no predicho. En la segunda columna podemos ver los valores de los contadores y confirmar su correcto funcionamiento.

4. Pruebas

Todas las pruebas realizadas y descritas hasta el momento se han ido realizando a medida que se iba modificando el MIPS y avanzando en el funcionamiento de la memoria caché en cada fase del proyecto.

A continuación, y con la versión final del proyecto, se pasará una prueba para comprobar que satisface todas las funcionalidades especificadas en el guión del proyecto.

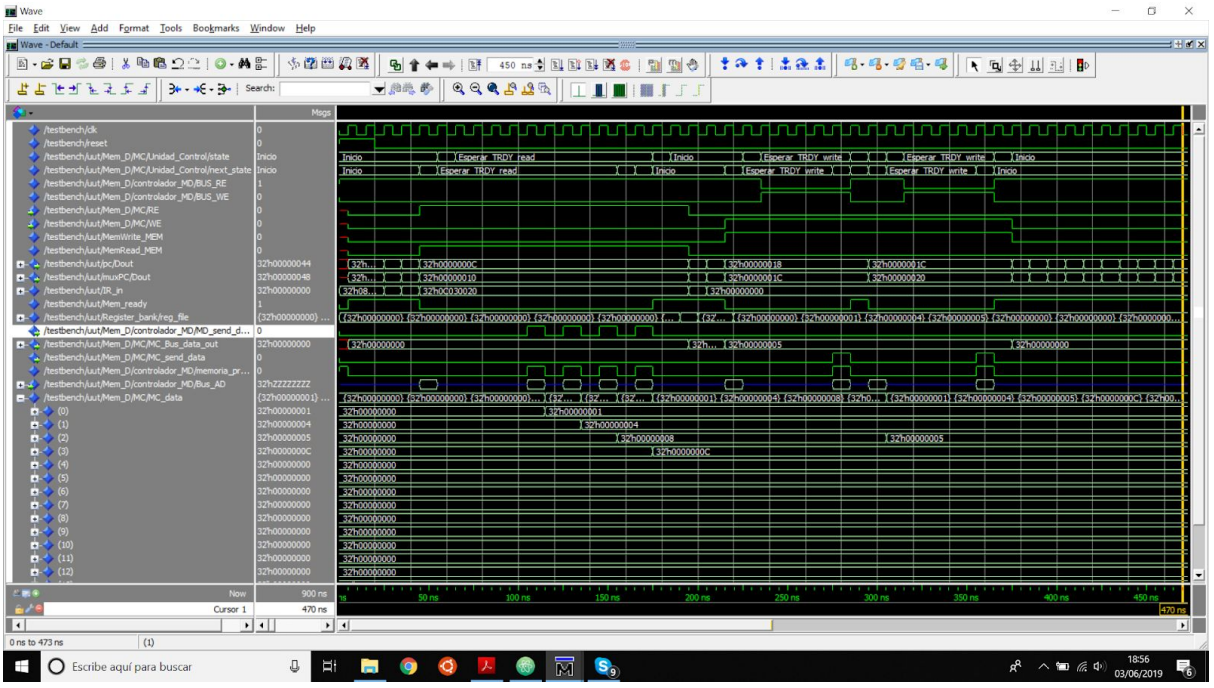
```
CODIGO PRUEBA
ini:
    LW R1, 0(R0)                //R1=@0 (read miss)
    LW R2, 4(R0)                //R2=@4 (read hit + load uso)
    ADD R3, R2, R1              //R3= @0+@4 (load uso)

    SW R3, 32(R0)               //@32 = @0 + @4 (write miss)
    SW R3, 8(R0)                //@8 = @0 + @4 (write hit)

EN Binario:
ini:
000010  00000   00001   0000000000000000        LW  R1, 0(R0)  @0
000010  00000   00010   0000000000000100        LW  R2, 4(R0)  @4
000001  00001   00010  00011 00000000000        ADD R3, R2, R1

000011  00000  00011  0000000000100000          SW R3, 32(R0)
000011  00000  00011  0000000000001000          SW R3, 8(R0)


En hexadecimal:
ini:
    08010000        LW  R1, 0(R0)
    08020004        LW  R2, 4(R0)
    04221800        ADD R3, R2, R1

    0C030020        SW R3, 32(R0)
    0C030008        SW R3, 8(R0)
```

Código de prueba para demostrar el funcionamiento

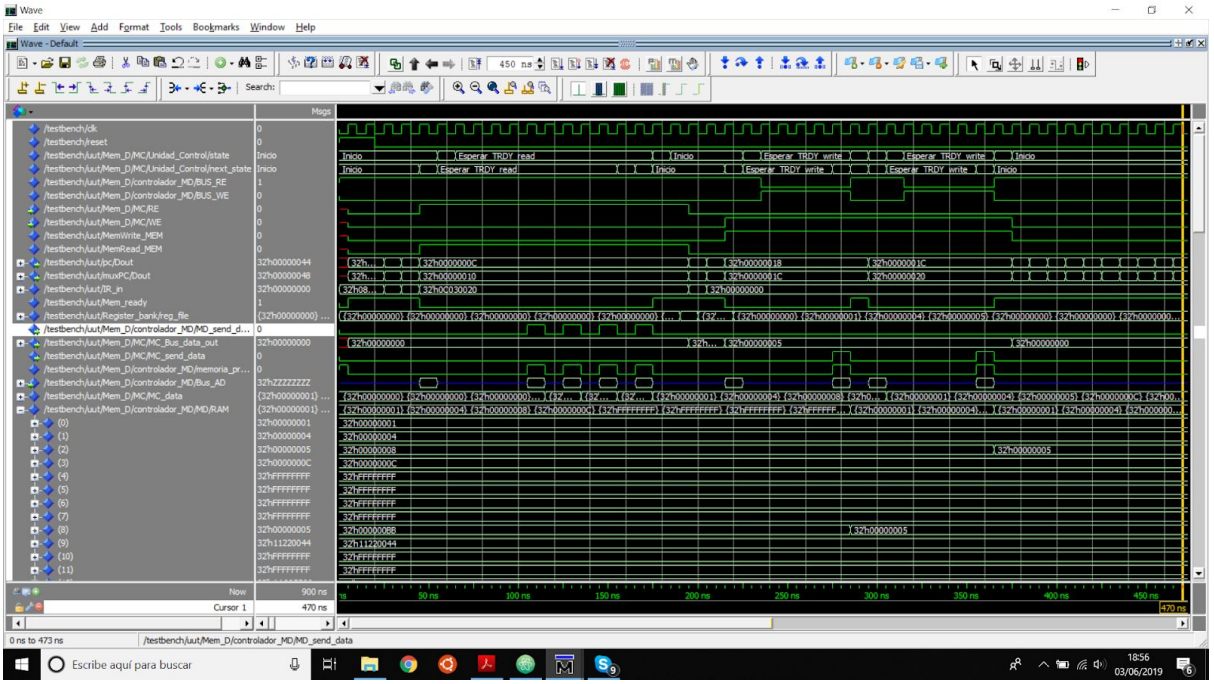
En el test realizado, se han probado todos los casos en los que se utiliza la memoria caché, siguiendo el siguiente orden: read miss (rm), read hit (rh), write miss (wm) y write hit (wh). Se trata de un código muy sencillo pero a la vez muy efectivo.

Primero, se intenta acceder a la dirección @0 de la memoria, pero no está almacenada en la memoria caché, por lo que no encontramos un read miss (aunque los tags coincidan, no va a suceder ningún problema gracias al bit de validez, que evita que se dé un falso hit).

Tras esto, se solicita a la memoria principal el bloque de 4 palabras a partir de la dirección @0 **(imagen "Captura mostrando MC")**. Después se solicita a la memoria la palabra escrita en la dirección @4. Como hemos traído el bloque a la memoria caché desde la memoria principal en la instrucción anterior, esto nos genera un read hit. A continuación se procede a sumar los contenidos de las dos instrucciones leídas (M(@0) + M(@4)) y lo guardamos en un registro para posteriormente almacenar el resultado en la memoria. Primero se intenta guardar el dato en una dirección de memoria que no está en la memoria caché (@32), por lo que aparece un write miss y escribe el resultado directamente en la memoria principal, ya que la política de la memoria caché en este apartado es Write Through**(imagen "Captura mostrando MD")**. A continuación se intenta almacenar el mismo resultado en la dirección @8, que sí se encuentra en la memoria caché, así que se escribe correctamente tanto en la memoria caché como en la memoria principal**(imágenes "Captura mostrando MC" y "Captura mostrando MD")**.



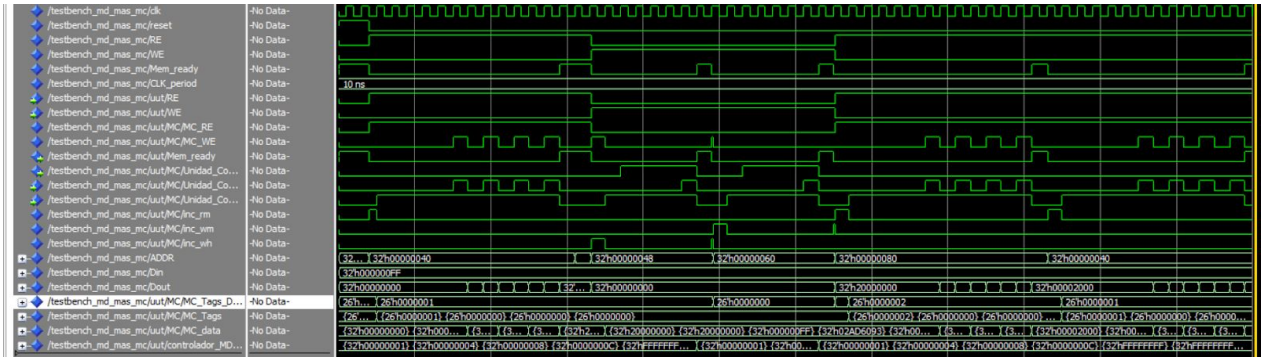
"Captura mostrando MC"



“Captura mostrando MD”

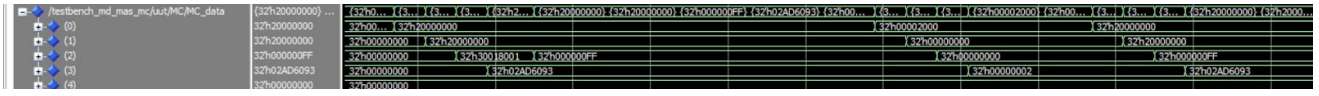
Prueba de las memorias MD mas MC

Para comprobar que las memorias MD y MC funcionaban correctamente, se ha utilizado el banco de pruebas que se ha proporcionado junto con los fuentes. Las pruebas que se han llevado a cabo son las descritas en el fichero testbench_MD_mas_MC.vhd. En el fichero, se declaran las acciones que se van a realizar que son, en orden: read miss (rm), read hit (rh), write hit (wh), read miss (rm) + reemplazo y readmiss (rm) + reemplazo.



“Captura wave MD mas MC”

Como se puede ver en la captura, las señales de incrementar cada una de las peticiones funcionan correctamente. Al ver esto y el contenido final de la MC, podemos confirmar que la memoria caché funciona correctamente .



5. Resultados

Descripción algorítmica de la MC

```
look-up(@x);
switch (proc_r/w)
        case proc_r: {      if hit(@x) { ret x'; }
                            else { Mp(rB,@x); waitfor Mp; Mc+X; MRU(@x); ret x'; }
                     }
        case proc_w: {      if hit(@x) { Mc+x'; Mp(wW,X‘); waitfor Mp; ret; }
                            else { Mp(wW,X‘); waitfor Mp; ret; }
                     }
```

En caso de acierto en lectura no habrá ciclos de penalización, por lo tanto los ciclos totales en este acceso a memoria será de 1.

En caso de acierto o fallo en escritura tardará CwW(MP), es decir, lo que tarde en escribir la palabra en la Memoria de Datos, en este caso, CwW(MP)= 4 ciclos.

En caso de fallo en lectura tardará CrB(MP) + 1, es decir, lo que tarde en leer el bloque de la Memoria de Datos, dónde CrB(MP) = N ciclos (para la primera palabra) + 3* M ciclos (para las palabras restantes), en este caso N=7 y M=2, por lo tanto CrB(MP) + 1 =
7 + 3*2 + 1 = **14 ciclos**

La fórmula para calcular los ciclos efectivos sería la siguiente:

$$\text{Ceff} = 1 + \frac{\Sigma wh * CwW}{\Sigma refs} + \frac{\Sigma wm * CwW}{\Sigma refs} + \frac{\Sigma rm * (CrB+1)}{\Sigma refs}$$

La primera mejora que se ha añadido ha sido la red de anticipación de operandos, que permite que el procesador funcione más rápido ya que no tiene que detener el pipeline (o introducir **nops**) por problemas de productor/consumidor RAW(Read after Write).
Se podrá ver mejor el impacto que tendrá en el número de instrucciones con un ejemplo:

Supongamos que tenemos una instrucción **add** que utiliza el registro **r1** para almacenar su resultado, y justo a continuación, otro **add** que utiliza también el registro **r1** para hacer una operación. Con la red de anticipación, se podría realizar una anticipación de operandos del registro **r1**, ahorrándonos la espera de 2 ciclos (EX Y MEM, no sería necesario esperar al ciclo WB ya que el procesador escribe en el BR en el flanco de bajada). Sin embargo, si no tuviésemos la red de anticipación y quisiéramos evitar los riesgos de datos tendríamos que introducir en el código, con planeación estática, 2 **nops** entre los dos **add.** Por lo que el impacto en el rendimiento con esta mejora sería evitar 2 instrucciones (y por lo tanto ciclos) menos por cada riesgo de datos. El CPI sería el mismo.
También se ha añadido la anticipación de operandos para el direccionamiento de la memoria RAM, lo que también nos ahorra dos ciclos de espera (No hace falta esperar a que la instrucción anterior escriba en el banco de registros de las etapas EX Y MEM).

La segunda mejora, también relativa al riesgo de datos, consiste en parar el pipeline el número de ciclos necesario (1 o 2) en el caso de productor-**beq** consumidor, el problema radica en que la instrucción de salto no puede beneficiarse de la red de anticipación. Por lo que, en cuanto al impacto en el rendimiento, al igual que en la anterior mejora, no sería necesario introducir 1 o 2 nops para evitar el riesgo, sin embargo el CPI aumentaría para este tipo de riesgos de datos, debido a los ciclos de detención del pipeline.

La tercera mejora es la gestión de un módulo de predicción dinámica de saltos, esto es una clara mejora en el rendimiento, ya que en el mejor de los casos(predicción correcta), no habría ningún ciclo de penalización en el salto. Antes de esta mejora, siempre se ejecutaba la siguiente instrucción al beq, por lo que era siempre necesario poner una nop, o en su defecto, detener el pipeline. Con esta mejora, en el peor de los casos, solo se detendrá el pipeline un ciclo si la predicción ha sido errónea. Por lo que el CPI será menor que antes de introducir esta mejora.

Como resultado de todas estas mejoras, serán necesarias 0 **nops** extra para evitar riesgos, por lo que el número de instrucciones totales para realizar una tarea será menor, y por lo tanto, los ciclos totales en realizar esa tarea. Además, con el módulo de predicción dinámica de saltos, los saltos que se predigan bien, durarán 1 ciclo menos que en el MIPS sin mejoras.

6. Conclusiones

Finalmente, tras realizar las pruebas y comprobar su correcto funcionamiento, se puede concluir que el proyecto ha sido completado, el MIPS funciona perfectamente con la cache y la memoria de datos añadida, y el diseño del autómata de la memoria cache es Meally, y con el número mínimo de estados para garantizar su óptimo funcionamiento.

Con los datos obtenidos en el anterior apartado **5.Resultados** se puede concluir que los acesos a memoria que mas penalizan son los rew o rm.

Tiempo dedicado

El tiempo dedicado es el mismo para los dos miembros del grupo, ya que ambos trabajamos a la vez por videollamada, dividiendo tareas en algunos casos pero generalmente diseñando el proyecto juntos.

El tiempo aproximado dedicado para cada tarea es el siguiente:

- Estudio de los ficheros fuente: 6 horas
- Diseño inicial de la unidad de control: 2 horas
- Depuración y ajustes: 8 horas
- Memoria: 7 horas

En comparación con el tiempo estimado en el guión del proyecto se han dedicado más horas a estudiar los ficheros, pero menos en depuración y ajustes. Esto es debido a que nuestra metodología de trabajo consistía en entender a la perfección el problema antes de enfrentarnos a él, de modo que hicimos un estudio exhaustivo de todas las señales de la memoria de datos y cache, además de recordar las del MIPS, estudiadas en el proyecto anterior, por esta razón fue mucho más fácil diseñar el proyecto y la unidad de control con menos errores, y en el caso de que hubiese un error, al depurar era más fácil entender dónde se producía el error.

Autoevaluación - Jesús Aguas (736935)

¿Crees que has cumplido los objetivos de la asignatura?

Sí, todo el trabajo que hay que hacer en la evaluación continua hace que te esfuerces por llevar el día la asignatura y realizar todos los trabajos a tiempo. El trabajo práctico en esta asignatura en esencial para asimilar todos los conocimientos, y por mi desarrollo en las prácticas y trabajos, puedo confirmar que he cumplido satisfactoriamente los objetivos de la asignatura.

¿Qué nota te pondrías si te tuvieses que calificar a ti mismo?

Un 9, ya que he completado todos los problemas, prácticas y trabajos en los límites establecidos y con la máxima calidad que he podido, revisando bien que no haya errores, y

cuidando mucho la presentación. Ni he faltado ningún día en las clases de problemas, ni en las sesiones de prácticas. No me pongo un 10 porque creo que eso solo puede decidir alguien que tenga el conocimiento total del ámbito de la asignatura para evaluar si mis conocimientos actuales sobre la materia son los óptimos o si me ha faltado profundizar en algún detalle del temario.

Autoevaluación - Daniel Barceló (737070)

¿Crees que has cumplido los objetivos de la asignatura?
En mi opinión, he cumplido con todos los objetivos de la asignatura, llevándola siempre al día y estudiando lo que se da. He entregado todos los proyectos, prácticas y problemas a tiempo intentando hacerlas lo mejor que me ha sido posible, siempre tratando mejorar algunos detalles hasta el día de la entrega definitiva. Debido a esto, he asimilado y aprendido todos los conocimientos necesarios para superar la asignatura, por lo que pienso que sí he cumplido con los objetivos.

¿Qué nota te pondrías si te tuvieses que calificar a ti mismo?

Me pondría un 9,5, ya que como he mencionado anteriormente, he cumplido todos los requisitos de la asignatura intentando sacar lo mejor de mí para realizar el mejor trabajo posible. Pero aun así un 10 implicaría la perfección en absolutamente todo, pero pienso que todo el mundo siempre puede esforzarse un poco más y seguir estirando el límite de sus capacidades.