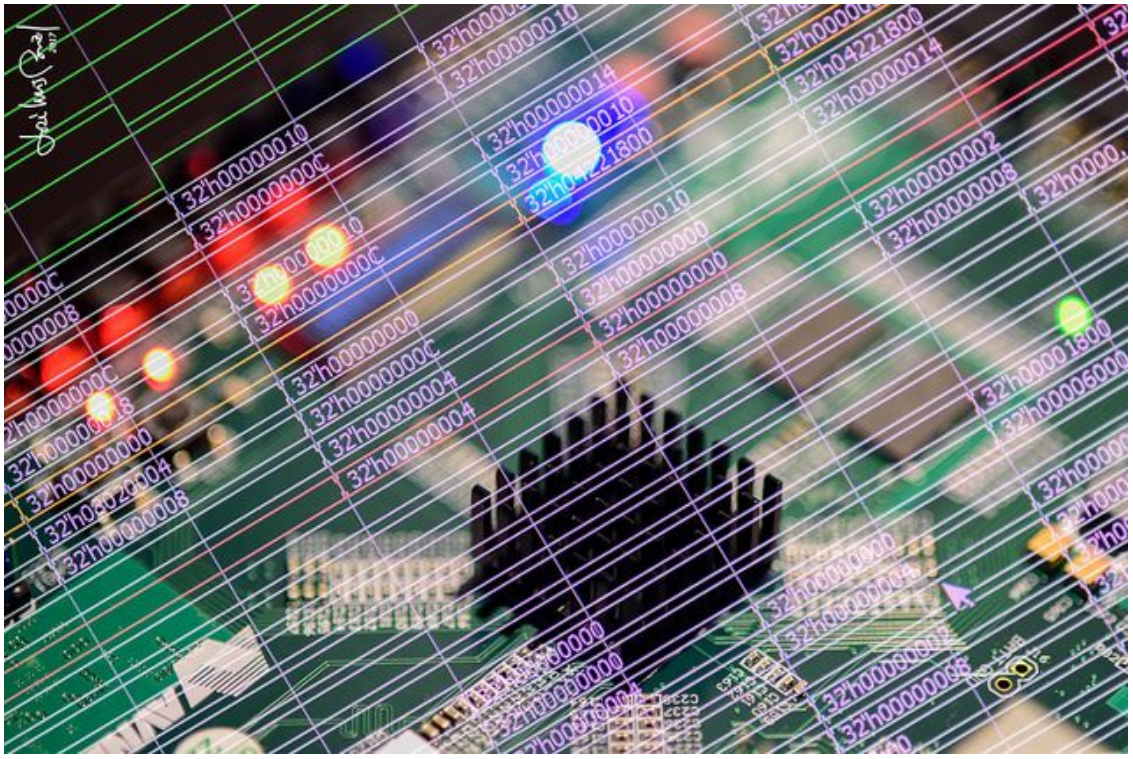
Segundo Proyecto AOC II

Jerarquía de memoria de datos



Autores:
Rubén Rodríguez 737215
Andrew Mackay 737069

Índice de contenidos

1. Introducción

El objetivo de este proyecto ha consistido en incorporar una memoria caché de datos a la memoria principal del procesador MIPS diseñado en el proyecto anterior. Para poder realizar dicha tarea se ha tenido que diseñar el controlador de la memoria caché, empleado para gestionar las peticiones del procesador realizando las transferencias que sean necesarias.

Paralelamente se ha incluido en el procesador soporte para que éste pueda detenerse cuando la memoria caché no pueda realizar la operación solicitada en un ciclo de reloj.

2. Trabajo Inicial

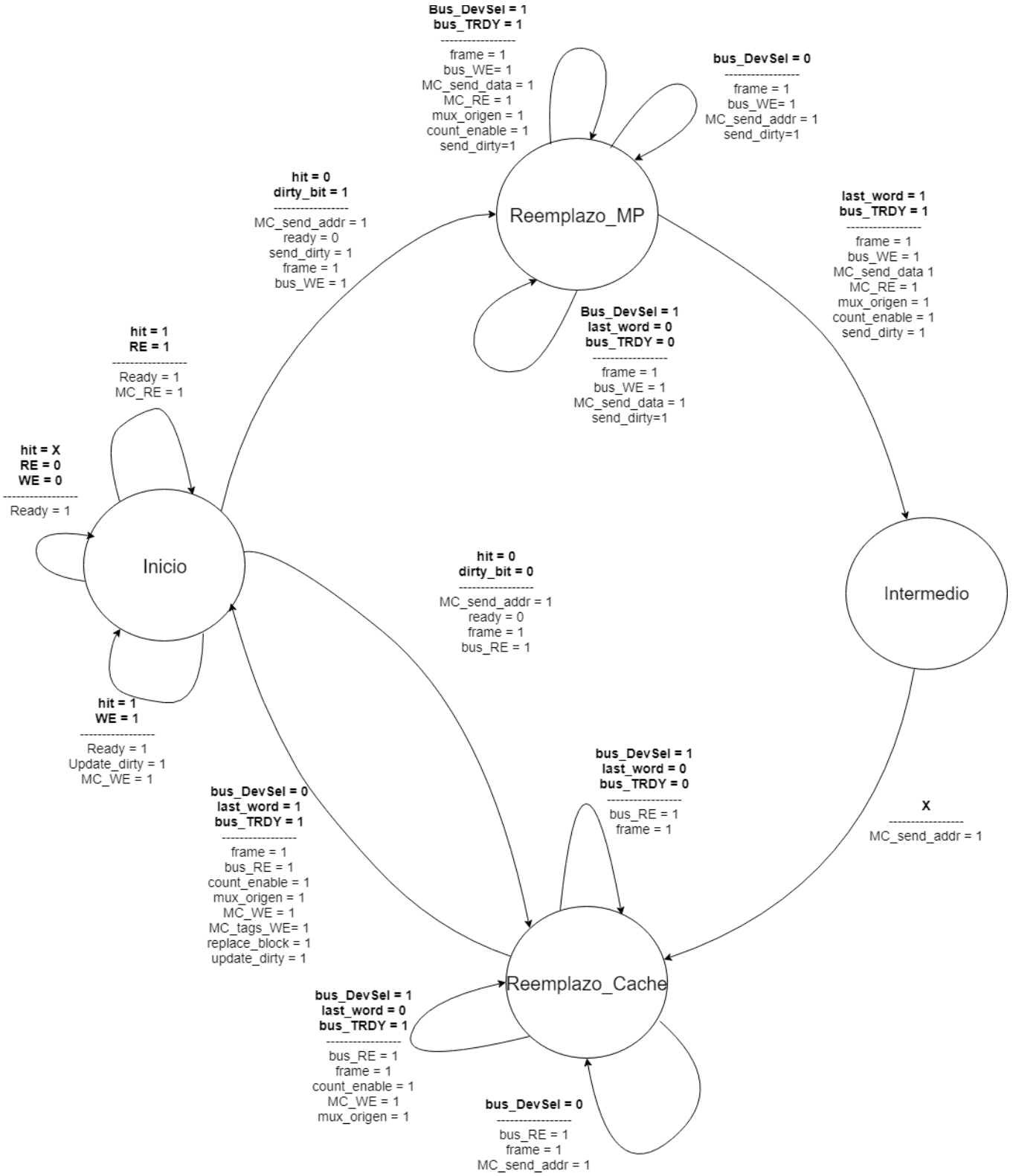
Para realizar este proyecto se ha tomado como punto de partida la implementación del procesador MIPS diseñada en el proyecto anterior, al que se le han tenido que realizar ciertas modificaciones para poder obtener el objetivo explicado en el apartado anterior. Así las cosas, el interfaz con el MIPS ha sido prácticamente idéntica a la del anterior proyecto salvo por el uso de una señal nueva, denominada *Mem_ready*. Dicha señal tiene como valor 1 cuando la memoria caché puede atender a la instrucción solicitada en el ciclo de reloj actual, en caso contrario esta señal tiene como valor 0.

3. Autómata de control

Para poder cumplir los objetivos explicados anteriormente, se ha implementado un autómata de control, también denominada máquina de estados, de tipo Mealy para poder diseñar la lógica de funcionamiento de la memoria caché con la finalidad de diseñar un mecanismo de sincronización entre está y la memoria principal del procesador MIPS. Seguidamente se muestran los estados que componen el autómata de control.

Nombre del estado	**Acción efectuada**
Inicio	Estado de inicio
Reemplazo_Cache	Estado de reemplazo de la memoria caché en caso de que no se produzca acierto
Reemplazo_MP	Estado en el que se reemplaza el bloque en la memoria principal en caso de que sea necesario reemplazar un bloque con el bit sucio activado en la caché
Intercambio	Estado intermedio entre las dos anteriores

3.1 Diagrama de estados del autómata Mealy



Se empieza por el estado llamado *Inicio*, donde se comprueba si se ha producido un acierto o no. Esto se consigue mediante la señal *hit* que se activará cuando la etiqueta y el conjunto de la dirección introducida coincida con la del banco de etiquetas. Si se produce un acierto y la instrucción es de lectura, leerá de la caché la palabra indicada por la dirección de entrada. Si es de escritura, escribirá el dato de entrada en la palabra indicada por la dirección.

Por otro lado, si no se produce un acierto se pueden distinguir dos casos distintos. Si el dirty_bit no está activado se procederá a reemplazar el bloque indicado por la dirección introducida con el bloque correspondiente de la memoria principal. Esto se realiza independientemente de si la instrucción es de escritura o lectura. Antes de llegar al estado de reemplazo, se manda la dirección al bus, se indica que la memoria caché está ocupada y que aún no ha terminado la operación.. Una vez en el estado *Reemplazo_Cache*, espera a que la memoria principal indique que ha reconocido la dirección. Una vez lo haya indicado, entra en un bucle en la que se leen datos procedentes del bus cuando la memoria principal avisa de que manda una palabra. Se leen y escriben los datos procedentes de la memoria principal cuando se activa la señal *Bus_TRDY.* Este bucle se repite hasta que se hayan leído y escrito las 4 palabras que forman el bloque. Una vez acabado el bucle se escribe la etiqueta en el banco de etiquetas, se marca al bloque como limpio y se vuelve al estado *Inicio*, de forma que hará hit y realizará la acción correspondiente.

Por último, si *dirty_bit* está activado, se realizará un reemplazo en la memoria principal, ya que el bloque indicado ha sido modificado en la cache pero no está modificada en la memoria principal. Se manda la dirección a la memoria principal y se indica que la caché está ocupada, que aún no ha terminado su operación y que se va a mandar un bloque sucio al bus. Después, se pasa al estado *Reemplazo_MP*, donde se realiza un bucle parecido al del estado *Reemplazo_Cache* descrito anteriormente, solo que en vez de leer del bus y escribir en la caché, se escribe en el bus las palabras leídas de caché para escribirlas en la memoria principal. Una vez terminado el bucle, se pasa a un estado llamado *Intermedio* para poner la señal *frame* a 0, ya que entre la escritura y la lectura de la memoria principal debe haber un ciclo intermedio con *frame* a 0, y se le manda la dirección a la memoria principal. De este estado se va otra vez al estado *Reemplazo_Cache* para reemplazar el bloque en la memoria caché.

4. Modificaciones de la Unidad de Detención

Una vez confeccionado el autómata de control que regula la secuencia de acciones de la memoria caché, se han tenido que llevar a cabo una secuencia de cambios en la unidad de detención del MIPS, diseñada en el proyecto anterior, con el objetivo de poder adaptarla a las nuevas necesidades que debe cumplir.

Inicialmente la UD del procesador MIPS contaba con cuatro señales para referenciar todos los posibles casos de parada. Dichas señales eran *Parar_EX* para controlar la instrucción del ADDFP hasta que ésta finalizara, y *Parar_ID*, para poder parar la ejecución cuando se

detectase una instrucción en la etapa EX y el registro en el que quiere escribir es uno de los que necesita la instrucción que viene justo después.

Además de estas dos señales se ha tenido que crear una nueva señal denominada Not_ready para detener la ejecución del MIPS en aquellas ocasiones en las que la memoria caché no pueda atender la instrucción solicitada en ese ciclo. Cuando la señal Not_ready esté activada, el resto de las señales de la UD están a 0. Cuando Not_ready se activa se paran los registros *pc, IF_ID, ID_EX* y *EX_MEM.*

5. Implementación de los contadores

Una vez finalizado el autómata de control y comprobado su correcto funcionamiento se ha procedido a implementar un conjunto de contadores para poder evaluar y en caso de no ser óptimo, mejorar, el rendimiento del nuevo procesador diseñado.

Para poder comprobar de forma exhaustiva que el procesador es eficiente al máximo se han empleado unos contadores que contabilizan las paradas que sufre el sistema implementado a la hora de ejecutar un programa concreto. Se han creado cuatro contadores destinados a contabilizar paradas de control, paradas de riesgos de datos, paradas de suma en coma flotante y paradas de memoria respectivamente.

5.1 Contador de paradas de control

En primer lugar se ha diseñado el contador paradas de control, surgidas como consecuencia de los riesgos de control. Se debe recordar que los riesgos de control se producían cuando se procede a ejecutar una instrucción de salto, es decir, una instrucción BEQ y no se conocía la siguiente instrucción a ejecutarse en caso de efectuar el salto hasta la etapa MEM.

Este problema fue solucionado utilizando la señal *Z*, que compara las salidas del banco de registros, la cual, se activa a 1 cuando son iguales, y otra señal llamada *Branch*, que indica que se está ejecutando actualmente una instrucción de salto. Cuando las dos señales anteriores están activadas, se pone a 1 *Kill_IF*. De esta forma, una parada de control debe efectuarse si y sólo si cuando se ejecuta una instrucción BEQ y debe tomarse el salto condicional, dado que deben desecharse todas las instrucciones que estén en ejecución de forma errónea insertando ceros en los valores de control. En definitiva, el contador de paradas de control debe contabilizar una parada si y sólo si la señal *Kill_IF* toma el valor 1. Se ha considerado que el contador sólo depende de *Kill_IF* ya que si la instrucción anterior modifica el contenido de algún registro de BEQ se interpreta como riesgo de dato y no de control.

5.2 Contador de paradas de datos

Paralelamente se ha implementar el contador de paradas de datos, en otras palabras, el contador encargado de contabilizar todas las paradas producidas por riesgos de datos. Es importante recordar que los riesgos de gestión de datos se producían en aquellos casos en los que la instrucción que se está ejecutando depende de instrucciones previas que aún están en proceso de tratamiento.

Tal problema fue resuelto implementando una unidad de anticipación de datos, permitiendo así poder usar los datos para ejecutar futuras instrucciones antes de que éstos sean almacenados en el banco de registros.

Para poder controlar la anticipación de los datos se usaba la señal *Parar_ID*, cuyo valor era 1 en aquellos casos en los que se es necesario parar hasta que el dato en cuestión ya está disponible, y valor 0 cuando no es necesario efectuar ninguna parada. Por consiguiente el contador de paradas de datos solamente debe recibir la orden de incremento cuando la señal *Parar_ID* toma el valor 1.

5.3 Contador de paradas por coma flotante

Seguidamente se ha procedido a diseñar el contador que lleva la cuenta de las paradas producidas por la operación suma en coma flotante. Es necesario enfatizar que el MIPS inicialmente no tenía implementada esta operación, factor al que se debe añadir que tampoco contaba con el hardware necesario para poder soportarla.

Por ello, para poder solventar estos riesgos estructurales, se tuvieron que realizar modificaciones en la Unidad de Control del MIPS, y al mismo tiempo programar la lógica de funcionamiento de la señal *FP_add_EX*, una señal de control del multiplexor encargado de seleccionar entre la salida de la unidad de suma en coma flotante y de la salida de la ALU de enteros, guardando el resultado en la salida *ALU_out_EX*. Además, dado que el ADDFP tarda varios ciclos en ejecutarse a diferencia del resto del repertorio de instrucciones, se programaron las señales *FP_done*, para marcar el fin de la ejecución de la instrucción, y la señal *Parar_EX* para que el procesador se pare hasta que la salida *FP_done* tome el valor 1, es decir, el final de la operación.

En conclusión, cuando la señal *Parar_EX* toma el valor 1, se incrementa el contador de paradas en coma flotante ya que el procesador está detenido esperando que la señal *FP_done* tome valor 1.

5.4 Contador de paradas de memoria

Por último se ha implementado el contador de paradas en memoria. Este contador tiene como labor contabilizar todas aquellas paradas que se producen cuando el procesador MIPS debe detener su ejecución como consecuencia de que la memoria caché sigue procesando una operación anterior, es decir, que la memoria caché no está preparada para poder atender la instrucción solicitada por el MIPS, y por tanto, eśte debe pararse hasta que la caché haya finalizado.

Para controlar el comienzo y el final de tarea de la memoria caché se ha programado el comportamiento lógico de una señal auxiliar denominada Not_ready para indicar que la caché todavía no está para continuar recibiendo instrucciones del MIPS. Así, si la señal Not_ready toma el valor 0 significa que la caché ha finalizado la instrucción y que está lista para poder continuar, mientras que si toma el valor 1, la caché no está lista para seguir porque todavía no ha terminado de ejecutar la instrucción recibida en el ciclo de reloj anterior, teniéndose que parar el MIPS hasta que termine. De esta manera cada vez que el MIPS tenga que pararse a esperar a la caché, se contabilizará una parada por el contador de paradas de memoria.

6. Pruebas de comprobación

Una vez concluído todo el proceso de implementación y verificación de errores se ha procedido a realizar un conjunto de pruebas exhaustivas para verificar el comportamiento adecuado y correcto tanto del MIPS como de la caché incorporada, comprobando en todo momento los valores de los contadores de paradas, controlando los fallos y aciertos tanto en lectura como escritura, así como los correspondientes reemplazos. Es importante enfatizar en el hecho de que todos los registros del banco inicialmente almacenan el valor 0.

6.1 Primera prueba

En esta primera prueba se ha implementado un programa que pone a prueba el funcionamiento de la memoria caché a la hora de gestionar los aciertos y fallos tanto de lectura como de escritura, así como la gestión de la política de reemplazo. Además también se encarga de confirmar que los contadores de paradas diseñados funcionan correctamente, contabilizando única y exclusivamente en aquellos casos en los que es necesario.

Para esta primera prueba en la dirección @0 está 0x3E4CCCCD, es decir, 0.2 en coma flotante y en la @1 está guardado 0x00000004

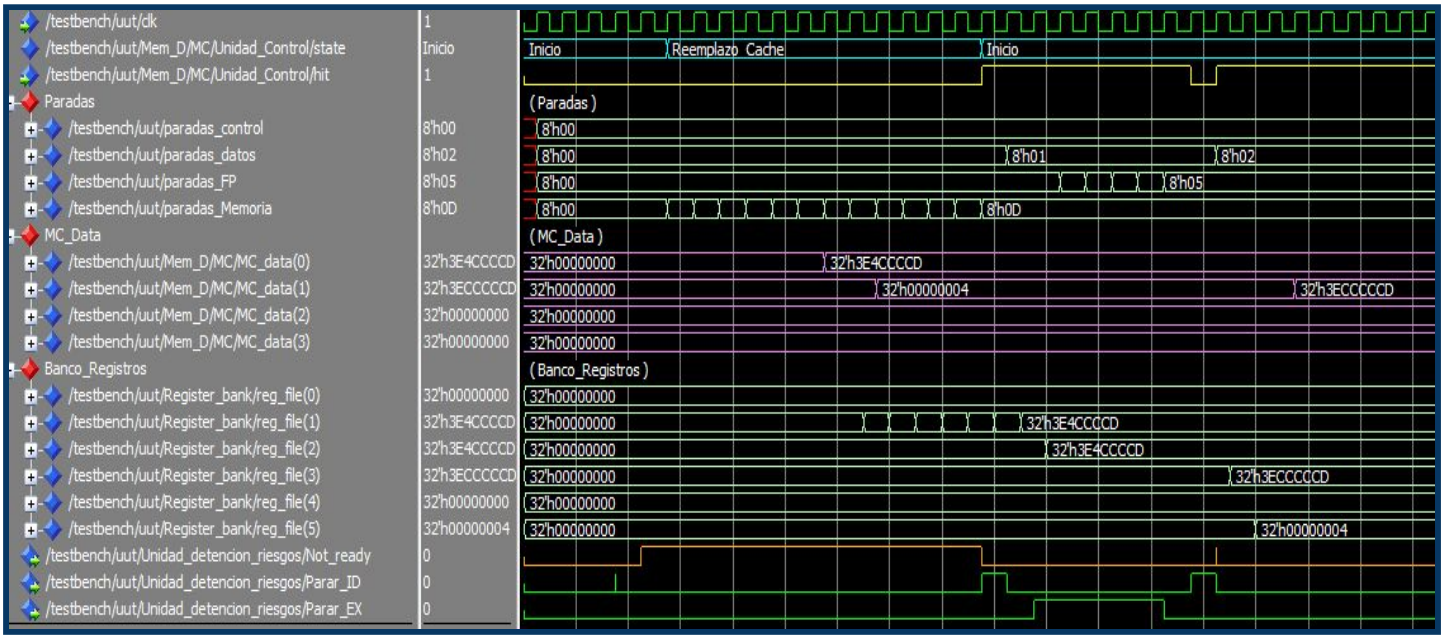
Instrucción	Comportamiento de la instrucción
LW r1(r0)	Carga en r1 el dato almacenado en memoria en la dirección apuntada por r0
LW r2(r0)	Carga en r2 el dato almacenado en memoria en la dirección apuntada por r0
ADDFP r3,r2,r1	Guarda en r3 la suma en coma flotante de r2 y r1
LW r5,4(r0)	Carga en r5 el dato apuntado por r0 en desplazado cuatro unidades
ST r3, (r5)	Guarda el registro r3 en la dirección de memoria apuntada por r5

Codificación de las instrucciones de gestión de memoria

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r1 , (r0)	000010	00000	00001	0000000000000000	08010000
LW r2 , (r0)	000010	00001	00010	0000000000000000	08020000
LW r5, (r0)4	000010	00000	00101	0000000000000100	08050004
SW r3, (r5)	000011	00011	00101	0000000000000000	0CA30000

Codificación de las instrucciones aritmético-lógicas

Instrucción	Op	Rs	Rt	Rd	Shamt	Funct	Hexadecimal
ADDFP r3, r2 , r1	100000	00010	00001	00011	00000	000000	80411800



En la captura anterior se puede observar como varían las señales durante la ejecución del programa. El programa carga en los registros r1 y r2 los datos de la dirección apuntada por r0, de tal forma que almacenan el valor 0x3E4CCCCD, es decir, 0.2 en coma flotante. Al ejecutarse la suma instrucción ADDFP se obtiene el resultado 0X3ECCCCD, aproximadamente 0.4 concluyendo que la instrucción ADDFP realiza bien los cálculos.

También puede observarse cómo evolucionan las señales *Not_ready*, *Parar_ID* y *Parar_EX*. Concretamente, la señal *Parar_EX* se activa en el mismo instante en el que comienza la ejecución ADDFP. De esta forma se puede evidenciar que el ADDFP tarda en ejecutarse exactamente 5 ciclos de reloj.

La señal *Not_ready* se activa a 1 debido a que la memoria caché no ha terminado de ejecutar la instrucción actual, y por ello el MIPS debe detenerse. Así, el contador de paradas de memoria contabiliza los ciclos que el MIPS permanece parado hasta que acaba la caché, concretamente 13 ciclos de reloj desde que Not_ready se pone a 1 hasta que adquiere valor 0.

La señal *Parar_ID* también evoluciona durante el programa como consecuencia de los riesgos de datos que se detectan, concretamente hay dependencias entre el LW anterior al ADDFP y el propio ADDFP, el último LW y el SW y el ADDFP con el SW debido a que se están utilizando registros cuyos valores todavía no se han actualizado, por ello, el contador de paradas de datos registra 15 paradas de datos, en definitiva, los mismos ciclos de reloj que la señal *Parar_ID* está activada a 1.

Se puede observar también que se escribe el bloque correctamente en la memoria cache y que al final del programa se escribe el resultado de la suma.

6.2 Segunda prueba

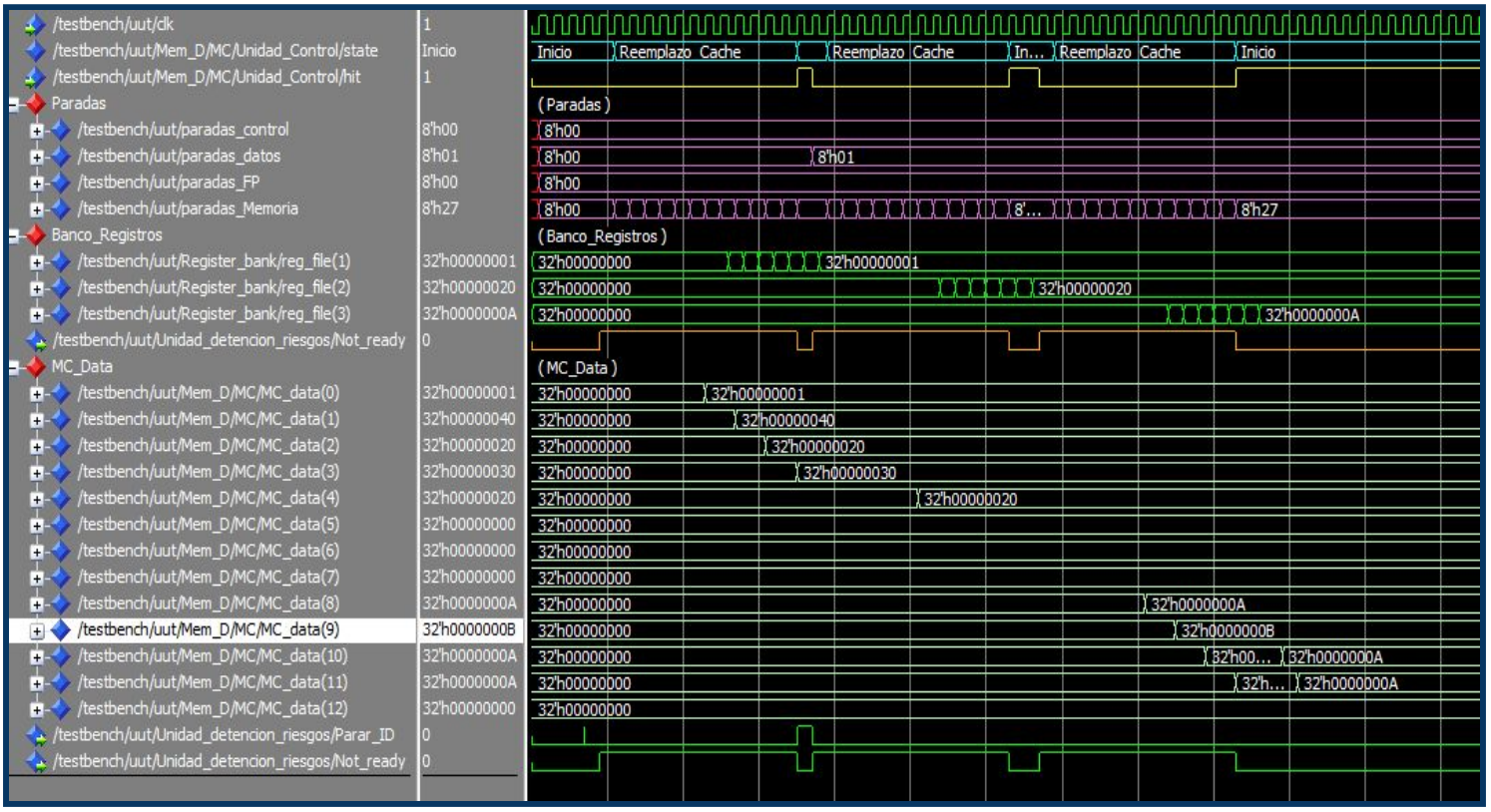
En esta segunda prueba se ha implementado un programa que pone a prueba el funcionamiento de la memoria caché a la hora de gestionar los aciertos y fallos tanto de lectura como de escritura y las políticas de reemplazo. En dicho programa solamente se incluyen operaciones que trabajan con memoria, es decir, loads y stores, analizando paralelamente los contadores de de riesgos de datos y de memoria.

Para esta primera prueba en la dirección los datos son este orden 0x00000001, 0x00000040, 0x00000020, 0x00000030 en la etiqueta 0 del conjunto 0, en la etiqueta 0 del conjunto 1 se guarda 0x00000020 y en la etiqueta 0 del conjunto 3 se almacena 0x0000000a, 0x0000000b, 0x0000000c, 0x0000000d.

Instrucción	Descripción de la instrucción
LW r1,(r0)	Carga en r1 el dato almacenado en memoria en la dirección apuntada por r0
LW r2,(r0)16	Carga en r2 el dato apuntado por r0 en desplazado dieciséis unidades
LW r3,(r2)	Carga en r3 el dato almacenado en memoria en la dirección apuntada por r2
SW r2,(r0)16	Guarda el registro r2 en la dirección r0 desplazada dieciséis unidades
SW r3,(r2)8	Guarda el registro r3 en la dirección r2 desplazada ocho unidades
SW r3,(r2)12	Guarda el registro r3 en la dirección r2 desplazada doce unidades

Codificación de las instrucciones de gestión de memoria

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r1,(r0)	000010	00000	00001	0000000000000000	08010000
LW r2,(r0)16	000010	00000	00010	0000000000010000	08020010
LW r3,(r2)	000010	00010	00011	0000000000000000	084300000
SW r2,(r0)16	000011	00010	00000	0000000000010000	0C020010
SW r3,(r2)8	000011	00011	00010	0000000000001000	0C430008
SW r3,(r2)12	000011	00011	00010	0000000000001100	0C43000C



Tal y como se reflejó anteriormente, en esta prueba solamente se están ejecutando instrucciones de gestión de memoria, es decir, loads y stores. En esta captura se puede reflejar el comportamiento de la caché con los aciertos y fallos que efectúa así como los reemplazos. Concretamente se puede observar que el primer load del programa falla ya que la dirección corresponde a la etiqueta 0 del conjunto 0, y como no está cargada, se efectúa un reemplazo rellenando las primeras 4 direcciones. De manera análoga ocurre con el segundo load ya que ocurre en la etiqueta 0 del conjunto 1, volviendo a fallar y reemplazando las 4 direcciones en caché, y exactamente lo mismo en el tercer load solo que en la etiqueta 0 del conjunto 2.

A continuación se vuelve a escribir en el tag 0 y set 1, y como ya está cargado, se escribe la nueva información y se activa la señal hit indicando que se ha producido un acierto, y así de forma análoga con los dos últimos stores, sobreescribiendo en el set 0 y tag 2 la palabra 3, y en el set 2 y tag 0 la palabra 4, respectivamente y haciendo hit porque ya existían en caché.

Paralelamente se han representado los valores de los contadores de paradas en memoria y de datos. De igual forma que en los métodos anteriores se puede observar que el contador de memoria se haya contando tantos ciclos como los que permanece activada la señal Not_ready ya que la memoria caché no ha terminado y el MIPS de pararse. Exactamente se registran un total de 27 ciclos de paradas por memoria.

En cuanto al contador de paradas por riesgos de datos, se detecta 1 parada debido a las dependencias del código. Se detectan todos esos ciclos debido a que la señal Not_ready está a 1 y además existen situaciones adicionales de dependencias como es el caso de la segunda y tercera instrucción.

6.3 Tercera prueba

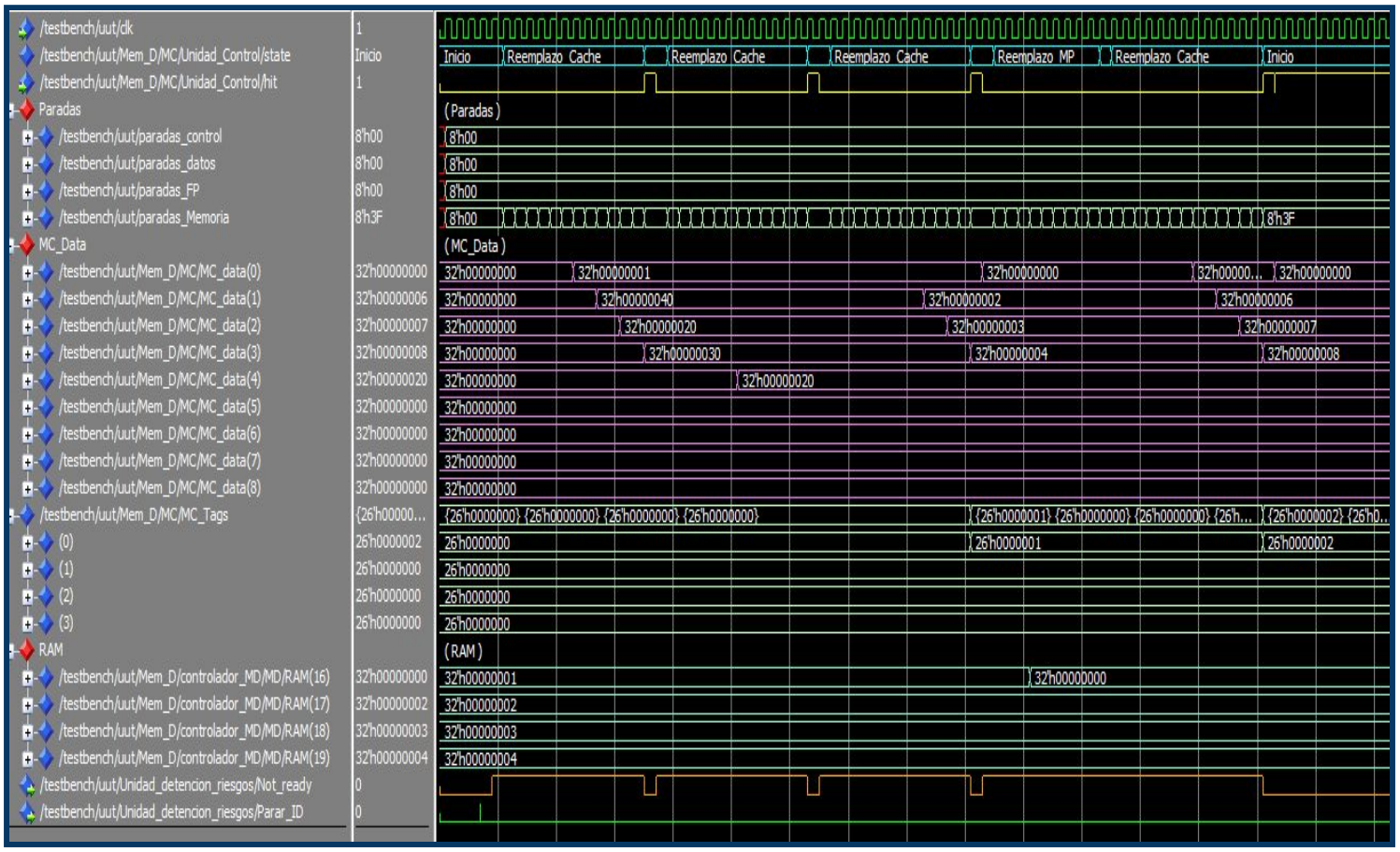
En esta tercera prueba se ha realizado un programa exactamente idéntico al anterior, con instrucciones de gestión de memoria para comprobar el correcto funcionamiento de la cache con etiquetas distintas para el mismo conjunto.

Para esta primera prueba en la dirección los datos son este orden 0x00000001, 0x00000040, 0x00000020, 0x00000030 en la etiqueta 0 del conjunto 0, en la etiqueta 0 del conjunto 1 se guarda 0x00000020, en la etiqueta 0 del conjunto 3 se almacena 0x0000000a, 0x0000000b, 0x0000000c, 0x0000000d, en la etiqueta 1 del conjunto 0 se guardan 0x00000001, 0x00000002, 0x00000003, 0x00000004 y en la etiqueta 2 del conjunto 0 se almacena 0x00000005, 0x00000006, 0x00000007 y 0x00000008.

Instrucción	Descripción de la instrucción
LW r1,(r0)	Carga en r1 el dato de la memoria apuntado por r0
LW r2,(r3)16	Carga en r2 el dato de la dirección apuntada por r3 desplazado 16 unidades
SW r4, (r5) 64	Guarda r4 en la dirección apuntada por r5 desplazado 64 unidades
SW r6,(r7)128	Guarda r6 en la dirección apuntada por r7 desplazado 128 unidades

Codificación de las instrucciones de gestión de memoria:

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r1,(r0)	000010	00000	00001	0000000000000000	08010000
LW r2,(r3)16	000010	00011	00010	0000000000010000	08620010
SW r4,(r5)64	000011	00100	00101	0000000001000000	0CA40040
SW r6,(r7)128	000011	00110	00111	0000000010000000	0CE60080



Esta prueba es exactamente idéntica a la anterior ya que trabaja con sólo operaciones de lectura y escritura en memoria. De acuerdo a la siguiente captura se pueden observar perfectamente los aciertos y fallos en caché a la hora de efectuar las lecturas y escrituras de datos. Se puede observar que la primera instrucción load falla porque en caché no hay nada almacenado, reemplazando primero las 4 primeras direcciones de la etiqueta 0 del conjunto 0. El segundo load ejecutado también falla ya que se trabaja en la etiqueta 0 del conjunto 1, por lo que se reemplaza el bloque en la memoria caché.

El primer store falla ya que escribe en el bloque con etiqueta 1 en el conjunto 0. Una vez reemplazado el bloque, escribe en el bloque, por lo que el bloque tendrá el bit sucio activado. También se modifica el tag guardado en el banco de etiquetas, como se puede observar en el grupo *MC_Tags*. El siguiente store lo hace sobre el la etiqueta 2 del conjunto 0. Como en el conjunto 0 está el bloque marcado con la etiqueta 1 y el bit sucio está activado, reemplaza en memoria principal el bloque modificado y luego reemplaza el bloque en la caché. La modificación de la memoria principal se puede observar con el grupo de señales *RAM*.

En cuanto a los contadores de parada se puede observar que el único contador que está activo es el contador de memoria, ya que en este programa no hay riesgos de control, ni operaciones ADDFP, al igual que tampoco hay riesgos de datos. Por esa razón esos tres

contadores valen 0, mientras que el de memoria ha contabilizado 3F ciclos, es decir, 58 ciclos de parada.

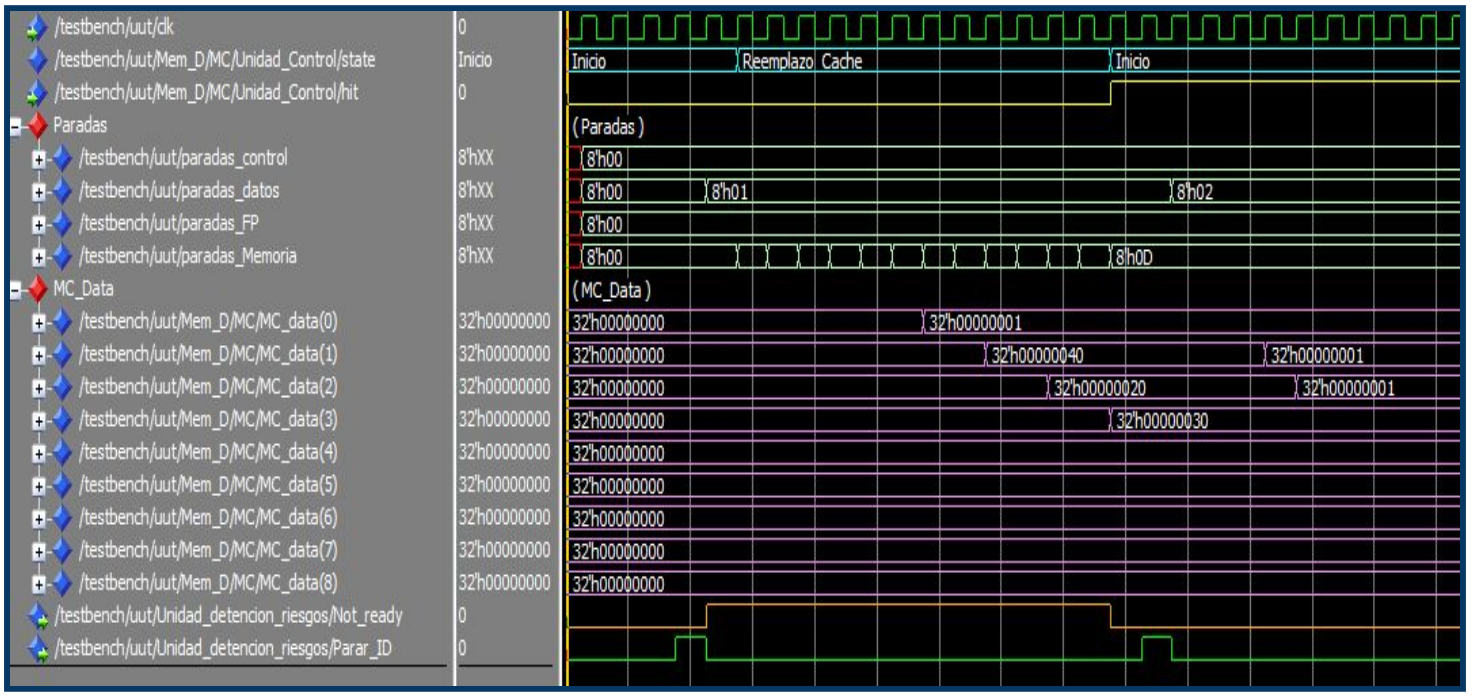
6.4 Cuarta prueba

En esta tercera prueba se ha realizado un programa exactamente idéntico al anterior, con instrucciones de gestión de memoria para comprobar el correcto funcionamiento de la caché cuando utilizan los mismos registros y de los contadores. La memoria principal es idéntica a la de la tercera prueba.

Instrucción	Descripción de la instrucción
LW r1,(r0)	Carga en r1 el dato de la dirección apuntada por r0
LW r1,(r0)	Carga en r1 el dato de la dirección apuntada por r0
SW r1,(r0)4	Guarda r1 en la dirección apuntada por r0 desplazado 4 unidades
SW r1,(r0)8	Guarda r1 en la dirección apuntada por r0 desplazado 8 unidades

Codificación de las instrucciones :

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r1,(r0)	000010	00000	00001	0000000000000000	08010000
LW r1, (r0)	000010	00011	00010	0000000000000000	08010000
SW r1,(r0)4	000011	00001	00000	0000000000000100	0C010004
SW r1,(r0)8	000011	00001	00000	000000010001000	0C010008



Tal y como se muestra en la captura anterior se puede observar cómo evolucionan los aciertos, fallos y reemplazos en caché. De acuerdo con el código del programa y la imagen anterior se puede observar que la primera instrucción del programa falla porque en la caché no hay ningún dato almacenado. Sin embargo, se puede ver que el segundo load sí que se produce un acierto ya que ya ese bloque ya existía en caché tras la ejecución de la instrucción anterior. Esto se consigue gracias a las detenciones de la unidad de control, lo que hace esperar al segundo load un ciclo más para que pueda realizar la anticipación correcta.
No obstante, se puede observar como en las dos últimas instrucciones se produce un acierto ya que se está utilizando mismo set y tag que en los anteriores, activándose por esa razón la señal hit a valor 1.

En lo referido a los contadores se puede apreciar que el contador de memoria contabiliza 13 ciclos de reloj, y el contador de datos registra 2 paradas de un ciclo como consecuencia de las dependencias entre el los dos primeros loads y el segundo load con el store. Los restantes contadores permanecen a 0 dado que no se ejecutan ADDFPs ni tampoco hay riesgos de control.

6.5 Quinta prueba

En esta última prueba se ha creado un programa que combina todos los aspectos importantes del nuevo diseño del procesador. Al igual que en los casos anteriores se utilizan instrucciones de gestión de memoria, empleando también instrucciones aritmético-lógicas con situaciones de riesgos de datos y saltos condicionales. La memoria principal es idéntica a la de la tercera prueba.

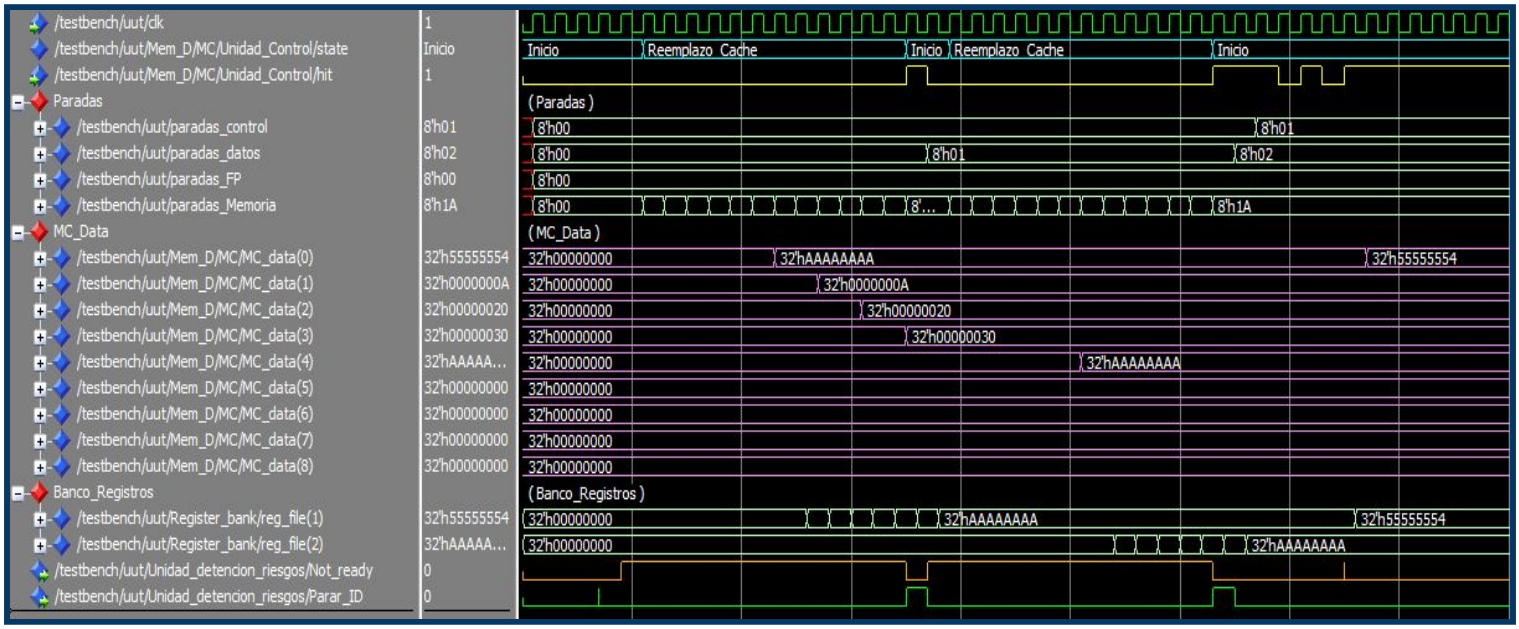
Instrucción	Descripción de la instrucción
LW r1,(r0)	Carga en r1 el dato de la dirección apuntada por r0
LW r2,(r3)16	Carga en r1 el dato de la dirección apuntada por r0
BEQ r1,r2,#1	Compara los registros r1 y r2 y salta 1 instrucción si son iguales
BEQ r1,r1,#-4	Compara los registros r1 y r1 y salta 4 instrucciones atrás si son iguales
ADD r1, r1, r1	Guarda en r1 el resultado de sumar r1 consigo mismo
SW r1,(r0)	Guarda r1 en la dirección apuntada por r0

Codificación de las instrucciones e¡de gestión de memoria

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r1,(r0)	000010	00000	00001	0000000000000000	08010000
LW r2, (r3)16	000010	00011	00010	0000000000010000	08620010
BEQ r1, r2,#1	000100	00001	00010	0000000000000001	10220001
BEQ r1, r1, #-4	000100	00001	00001	1111111111111011	1021FFFB
SW r1,(r0)	000011	00001	00000	0000000000000000	0C010000

Codificación de las instrucciones aritmético-lógicas

Instrucción	Op	Rs	Rt	Rd	Shamt	Funct	Hexadecimal
ADD r1, r1 , r1	000000	00001	00001	00001	00000	000000	04210800



Se puede observar que los datos que se están sumando son 0xAAAAAAAA cuya suma es el valor obtenido en el registro r1, y que ha sido almacenado en memoria en la dirección apuntada por r0. Se puede observar el correcto funcionamiento de las instrucciones BEQ dado que el programa no entra en bucle. El programa finaliza en la primera iteración dado que los registros r1 y r2 son iguales.

De acuerdo al programa anterior, se puede observar que en las primeras instrucciones se producen dos fallos que se pueden observar en la caché cuando se efectúan los reemplazos. También se puede ver la escritura del resultado de la suma

Se puede apreciar que la señal *Parar_ID* está activada como consecuencia de que se produce detención en la ejecución del segundo load con el primer beq dado que r2 todavía no ha sido cargado y el add con el store ya que se debe actualizar el registro de r1 antes de guardarse en memoria, correspondiendose con las dos paradas de *Parar_ID*.

La señal de *Not_ready* también se activa debido a que la caché todavía no ha terminado la ejecución de la instrucción actual y por tanto, el MIPS debe pararse hasta que la caché finaliza. El contador de control ha realizado 1 ciclo, mientras que el contador de memoria ha contado 26 ciclos (1a) y que el contador de riesgos ha registrado 2 paradas.

7. Corrección

El primer fallo que se ha solucionado es el no-crítico: se ha detectado que se debe detener el load del registro *Mem/WB* ya que si se detiene el MIPS por el *Mem_ready*, se pierden los valores, por lo que no puede anticipar de forma correcta una instrucción si la instrucción siguiente es un acceso a memoria que falla.

Por otro lado, se ha arreglado el autómata de la memoria cache para que se comporte de forma correcta cuando se realiza un reemplazo en la memoria principal. Se ha añadido que el autómata debe esperar en el estado *Reemplazo_MP* mientras *Bus_DevSel* esté desactivado. Además, se ha añadido que cuando se activa la señal *Replace_block* se activa también la señal *Update_Dirty* para poner el bit sucio a 0. Se han hecho los cambios correspondientes al autómata de la página 4.

7.1 Prueba de corrección

A continuación se muestra una prueba de corrección para poder asegurar que los fallos comentados en el apartado anterior han sido solventados correctamente. Esta prueba combina instrucciones de gestión de memoria, saltos condicionales y operaciones aritmético-lógicas. De igual forma permite comprobar que la caché se comporta correctamente. De igual forma que en las pruebas anteriores todos los registros valen 0.
Para esta primera prueba en la dirección los datos son este orden 0x00000001, 0x00000002, 0x0000000c, 0x0000000d

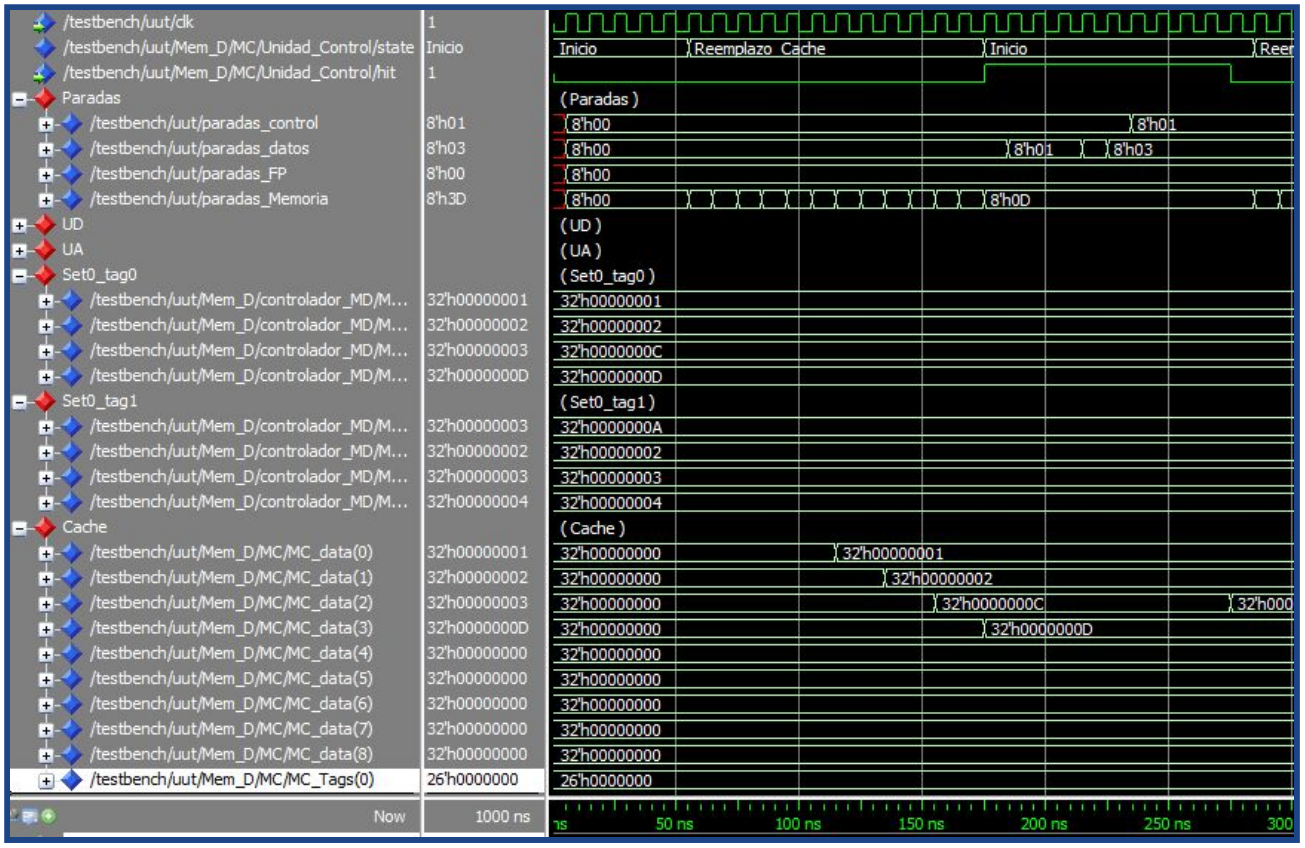
Instrucción	Descripción de la instrucción
LW r1,(r0)	Carga en r1 la @ apuntada por r0
LW r2,(r3),#4	Carga en r2 la @ apuntada por r3 desplazada 4 unidades
ADD r3,r2,r1	Guarda en r3 la suma de r2 y r1
ADD r4,r2,r1	Guarda en r4 la suma de r2 y r1
BEQ r4,r3,#1	Compara r4 y r3 y si son iguales salta una instrucción
BEQr4,r4,#-1	Compara r4 y r4 y si son iguales salta una instrucción hacia atrás
STR r4,(r0)#8	Almacena r4 en la @ de r0 desplazada 8 unidades
STR r3,(r0)#64	Almacena r3 en la @ de r0 desplazada 64 unidades
LW r6,(r0)#64	Carga en r6 lla @ de r0 desplazada 64 unidades
LW r1,(r0)	Carga en r1 la @ apuntada por r0

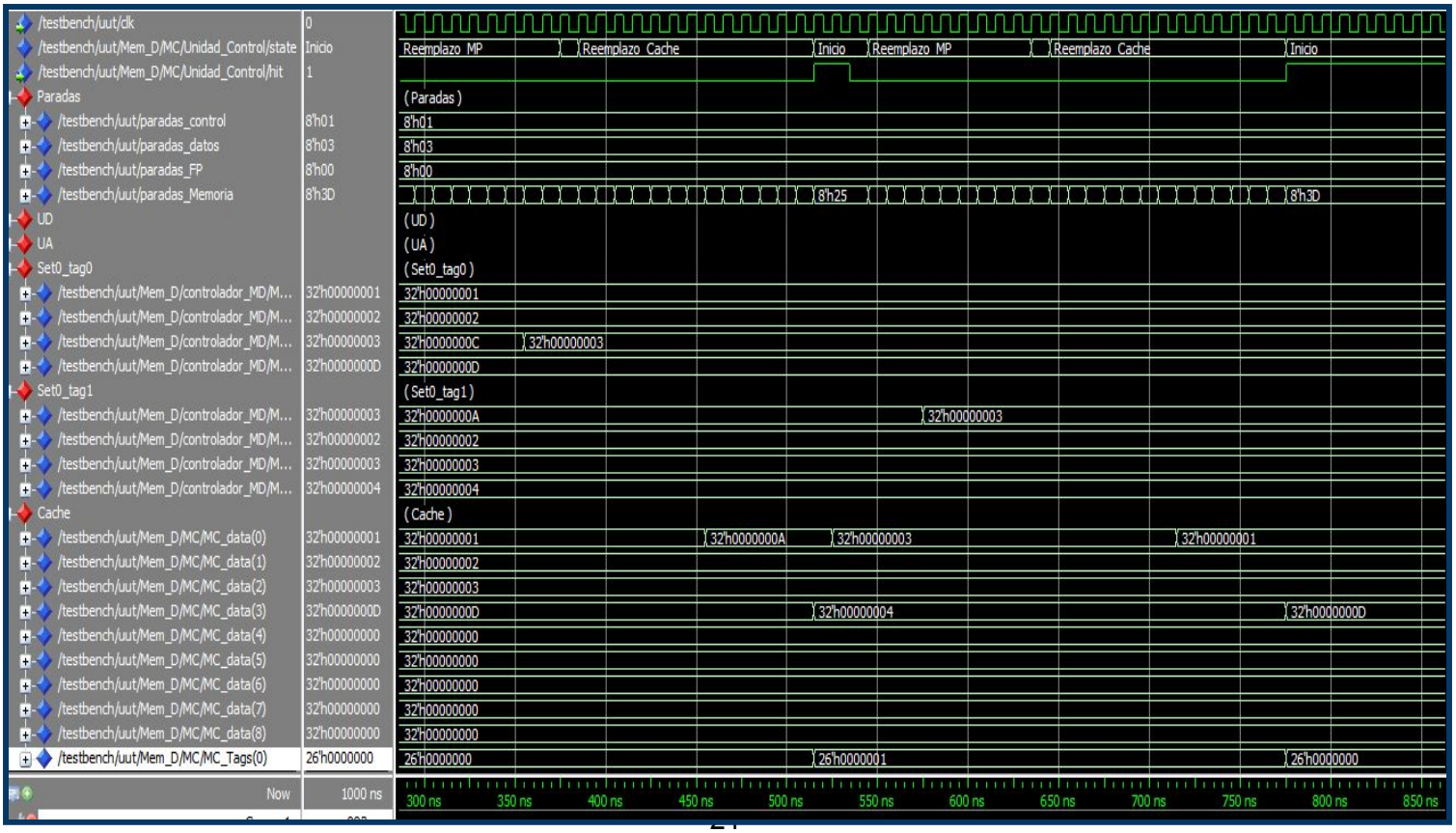
Codificación de las instrucciones de gestión de memoria

Instrucción	Op	Rs	Rt	K	Hexadecimal
LW r1,(r0)	000010	00000	00001	0000000000000000	08010000
LW r2, (r3)#4	000010	00011	00010	0000000000010000	08620004
BEQ r4, r3,#1	000100	00100	00011	0000000000000001	10830001
BEQ r4, r4, #-1	000100	00100	00100	1111111111111111	1084FFFF
SW r4,(r0),#8	000011	00100	00000	0000000000001000	0C040008
STR r3,(r0)#64	00011	00011	00000	0000000001000000	0C030040
LW r6,(r0)#64	00010	00110	00000	0000000001000000	08060040
LW r1,(r0)	00010	00000	00001	0000000000000000	08010000

Codificación de las instrucciones aritmético-lógicas

Instrucción	Op	Rs	Rt	Rd	Shamt	Funct	Hexadecimal
ADD r3, r2 , r1	000001	00001	00001	00001	00000	000000	04411800
ADD r4,r2,r1	00001	00010	00001	00110	00000	000000	04412000


/testbench/uut/clk
/testbench/uut/Mem_D/MC/Unidad_Control/state
/testbench/uut/Mem_D/MC/Unidad_Control/hit
Paradas
/testbench/uut/paradas_control
/testbench/uut/paradas_datos
/testbench/uut/paradas_FP
/testbench/uut/paradas_Memoria
UD
UA
Set0_tag0
/testbench/uut/Mem_D/controlador_MD/M...
Set0_tag1
Cache
/testbench/uut/Mem_D/MC/MC_data(0)
/testbench/uut/Mem_D/MC/MC_data(1)
/testbench/uut/Mem_D/MC/MC_data(2)
/testbench/uut/Mem_D/MC/MC_data(3)
/testbench/uut/Mem_D/MC/MC_data(4)
/testbench/uut/Mem_D/MC/MC_data(5)
/testbench/uut/Mem_D/MC/MC_data(6)
/testbench/uut/Mem_D/MC/MC_data(7)
/testbench/uut/Mem_D/MC/MC_data(8)
/testbench/uut/Mem_D/MC/MC_Tags(0)
Now
1000 ns
Inicio
Reemplazo_Cache
50 ns
100 ns
150 ns
200 ns
250 ns


/testbench/uut/clk
/testbench/uut/Mem_D/MC/Unidad_Control/state
/testbench/uut/Mem_D/MC/Unidad_Control/hit
Paradas
/testbench/uut/paradas_control
/testbench/uut/paradas_datos
/testbench/uut/paradas_FP
/testbench/uut/paradas_Memoria
UD
UA
Set0_tag0
/testbench/uut/Mem_D/controlador_MD/M...
Set0_tag1
Cache
/testbench/uut/Mem_D/MC/MC_data(0)
/testbench/uut/Mem_D/MC/MC_data(1)
/testbench/uut/Mem_D/MC/MC_data(2)
/testbench/uut/Mem_D/MC/MC_data(3)
/testbench/uut/Mem_D/MC/MC_data(4)
/testbench/uut/Mem_D/MC/MC_data(5)
/testbench/uut/Mem_D/MC/MC_data(6)
/testbench/uut/Mem_D/MC/MC_data(7)
/testbench/uut/Mem_D/MC/MC_data(8)
/testbench/uut/Mem_D/MC/MC_Tags(0)
Now
1000 ns
Reemplazo_MP
Reemplazo_Cache
Inicio
300 ns
350 ns
400 ns
450 ns
500 ns
550 ns
600 ns
650 ns
700 ns
750 ns
800 ns
850 ns

Tal y como se muestra en las captura la ejecución de la primera instrucción del programa producen un fallo en caché y por ello, se debe efectuar el reemplazo en el set 0 y tag 0 tal y como se puede ver en la primera imagen. La segunda instrucción produce un acierto reemplazando en caché la tercera palabra del set 0 y tag 0. En la segunda imagen mostrada se puede observar como en el primer store se produce un acierto y como en el siguiente store se produce un fallo y un reemplazo, ya que el bloque actual en el conjunto 0 ha sido modificado y se trae de la memoria principal el bloque correspondiente a la etiqueta 1 de ese conjunto. Luego se produce un acierto en lectura de cache y en la última instrucción se produce un fallo donde hay que reemplazar la memoria principal, ya que el conjunto indicado ha sido modificado.

En estas capturas se puede mostrar el correcto funcionamiento de las instrucciones BEQ ya que se producen los saltos correctamente y el programa no entra en bucle infinito

8. Conclusión del proyecto

Durante la realización de este proyecto, ambos participantes hemos aprendido gestionar el funcionamiento de una caché, concretamente los aciertos, los fallos y la política de reemplazos. También hemos aprendido a manejar un lenguaje de descripción de Hardware y un entorno de simulación como es el programa ModelSim.

9. Planificación del trabajo

En este último apartado se muestra una planificación de las tareas realizadas y del tiempo aproximadamente en horas que ha contado hacer cada una de ellas

Tarea	Duración
Comprensión de simulador ModelSim:	cuatro horas
Diseño del autómata	tres horas
Modificación de la UD	media hora
Implementación de contadores	una hora
Debuggear y depuración de fallos	veinte horas
Realización de la memoria	cinco horas

10. Autoevaluación

10.1 Rubén Rodríguez

La asignatura me ha parecido muy interesante desde el principio de curso, además me parece que las clases de problemas, las prácticas y los proyectos son muy beneficiosos a la hora de facilitar la comprensión del temario de la asignatura. Personalmente creo que he cumplido con todos los objetivos de la asignatura ya que entiendo perfectamente los contenidos y he realizado bien todos los trabajos mandados. Personalmente creo que me merezco un 8 porque he trabajado diariamente y he comprendido el mensaje que la asignatura pretende transmitir.

10.2 Andrew Mackay

Me ha parecido interesante la asignatura. La idea de la evaluación continua, entre hacer las prácticas, los problemas y los proyectos, es un buen criterio de evaluación ya que te motiva a trabajar mucho y aprenderte el temario. Desde mi punto de vista, creo que me merezco un 8 porque he sido un alumno constante y aplicado durante todo el cuatrimestre.