


Proyecto 2

Arquitectura de computación

Trabajo Realizado por:

Gregorio Largo Mayor
Alonso Muñoz García

NIA:746621
NIA:745016



ÍNDICE

| | |
|--|----|
| 1) Explicación breve del diseño realizado | 3 |
| 2) Hardware añadido y que función desempeña | 3 |
| A) Diagrama de estados (Autómata) | 3 |
| AUTÓMATA..... | 4 |
| B) Modificaciones en el fichero UC_MC..... | 9 |
| C) Modificaciones realizadas en el MIPS..... | 9 |
| D) Modificaciones realizadas en la Unidad de detención..... | 9 |
| E) Modificaciones realizadas en la cache..... | 9 |
| 3) Impacto en rendimiento de cada una de las modificaciones con respecto al procesador inicial | 10 |
| 4) Pruebas realizadas para verificar que funciona correctamente | 10 |
| 4.1 Prueba 1 | 10 |
| 4.2 Prueba 2 | 11 |
| 4.3 Prueba 3 | 12 |
| 4.4 Prueba 4 | 14 |
| 4.5 Prueba 5 | 15 |
| 4.6 Prueba 6 | 15 |
| 5) Conclusiones y cuantificación de horas dedicadas por cada miembro. | 17 |

1) Explicación breve del diseño realizado

Para este diseño hemos realizado la implementación en vhd de una memoria cache, que posee una comunicación por medio de un bus síncrono con la memoria principal. Hemos tenido que implementar para ello un autómata de estados que realizara tanto la comunicación con la memoria principal, como con el procesador MIPS que diseñamos en el proyecto 1. Para ser capaces de controlar todo este entorno hemos tenido que realizar modificaciones, tanto en el MIPS, deteniendo su procesamiento en determinados casos debido a que la cache tarda en obtener los datos en algunos casos un periodo de tiempo, como en la cache dada por los profesores en la cual hemos tenido que añadir el autómata mencionado.

2) Hardware añadido y que función desempeña

Antes de mencionar en los ficheros los cambios realizados y el porqué de estos cambios vamos a mostrar el autómata que hemos realizado para gestionar la unidad de control de la memoria cache y vamos a explicarlo.

A) Diagrama de estados (Autómata)

Las entradas que se tienen en cuenta en el autómata son las siguientes y siguen el siguiente orden:

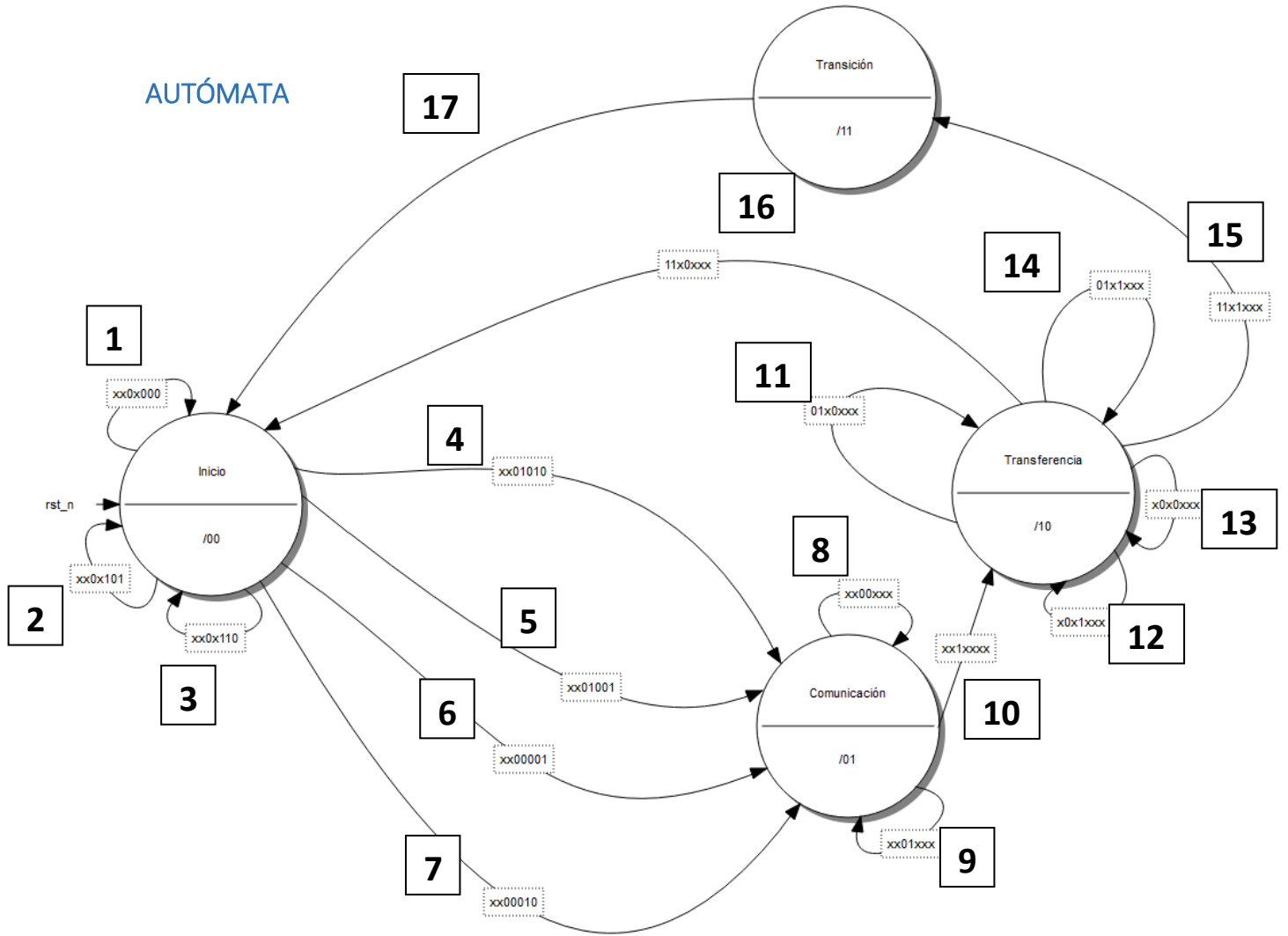
- **Last_word:** Si nos encontramos en la última palabra del bloque, es decir que el contador tenga un valor de 4 iteraciones ('11') entonces esta entrada valdrá '1'.
- **Bus_TRDY:** Entrada procedente del slave, en este caso nuestra memoria principal, esta lista para recibir o para enviar datos en caso de que valga '1', si no está entrada valdrá '0'.
- **Bus_Devsel:** Entrada procedente del slave es utilizada para confirmar que la dirección dada por el MIPS está dentro del rango de nuestra memoria principal si es así valdrá '1' sino valdrá '0'.
- **Dirty_bit:** Esta entrada vale '1' cuando el conjunto, al que queremos acceder está sucio es decir está escrito, si no está escrito o lo que es lo mismo si lo que está escrito nos da igual valdrá '0'.
- **Hit:** Esta entrada valdrá '1' cuando la etiqueta que hay en el tag apuntado por el conjunto y la etiqueta de la dirección son iguales, en cualquier otro caso valdrá '0'.
- **RE:** Valdrá '1' cuando la operación que queramos realizar sea de lectura, sino '0'.
- **WE:** Valdrá '1' cuando la operación que queramos realizar sea de escritura, sino '0'.

Las salidas del autómata están establecidas siempre con el valor '0' y la del estado siguiente con el valor inicio, en los diferentes estados varía dependiendo de la transición que se realice lo detallaremos más adelante por qué cambiamos el valor de esa salida y a que se debe en cada momento.

replace_block, MC_send_data, Update_dirty, MC_tags_WE, mux_origen, count_enable, send_dirty, bus_WE, bus_RE, reset_word, Frame, MC_send_addr, MC_WE, MC_RE, ready

NOTA: Para entender las transiciones del autómata las hemos numerado si no se ven claras tu pinchas sobre el número y te lleva a la explicación, y para volver al autómata simplemente tienes que pinchar en (Autómata) y te lleva de nuevo a la imagen así es más sencillo desplazarse.

AUTÓMATA



Explicación de lo que se mira en cada transición al estar poco claras

1. Transición número 1 me encuentro en el estado inicio (Autómata)

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| x | x | x | x | x | 0 | 0 |

Si no se detecta ninguna operación entonces nos mantenemos en el estado que estamos (`next_state = 0`) y las salidas las mantenemos todas a cero.

2. Transición número 2 me encuentro en el estado inicio (Autómata)

Las entradas para que se cumplen en este estado son:

| | | | | | | |
|-----------|----------|------------|-----------|-----|----|----|
| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
| x | x | x | x | 1 | 0 | 1 |

Si detecto una operación de escritura ($WE = 1$) y la señal hit esta activa significa que el conjunto al que quiero acceder ya lo tengo en la cache por lo que puede responder al MIPS en este mismo ciclo. Las salidas que en este caso cambiaré serán:

| Next_State | Ready | MC_WE |
|------------|-------|-------|
| inicio | 1 | 1 |

La señal ready para avisar al MIPS de que le respondo en este mismo ciclo y MC_WE para que MC_data actualice el dato en memoria cache al que apunta esa dirección.

3. Transición número 3 me encuentro en el estado inicio (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| x | x | x | x | 1 | 1 | 0 |

Si detecto una operación de lectura (RE = 1) y la señal hit esta activa significa que el conjunto al que quiero acceder ya lo tengo en la cache igual que en el caso anterior. Las salidas que en este caso cambiaré serán:

| Next_State | Ready | MC_RE |
|------------|-------|-------|
| inicio | 1 | 1 |

La señal ready para avisar al MIPS de que le respondo en este mismo ciclo y MC_RE para que MC_data lea el dato en memoria cache al que apunta esa dirección.

4. Transición número 4 y 5 me encuentro en el estado inicio (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----------|----|
| x | x | x | 1 | 0 | RE OR WE | |

Si detecto una operación de lectura (RE = 1) o de escritura (WE = 1) y la señal hit esta desactiva significa que el conjunto al que quiero acceder no lo tengo en la cache y si a su vez la señal dirty_bit está activa significa que ya tengo un conjunto en la cache, por lo que no puedo modificarlo aún, debo de enviar primero al slave el bloque sucio para que lo guarde. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | MC_send_addr | Reset_word | bus_WE | Send_dirty |
|--------------|-------|--------------|------------|--------|------------|
| comunicación | 1 | 1 | 1 | 1 | 1 |

La señal frame la activo para avisar al slave de que quiero realizar una comunicación, MC_send_addr la activo para que el slave sepa que le estoy enviando la dirección, Reset_word la activo para que se ponga a cero el contador de palabras recibidas e enviadas, bus_WE la activo para que el slave sepa que operación quiero realizar en este caso de escritura, y la señal send_dirty la activo para que la dirección que se envíe sea la del tag de la cache.

5. Transición número 6 y 7 me encuentro en el estado inicio (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----------|----|
| x | x | x | 0 | 0 | RE OR WE | |

Si detecto una operación de lectura (RE = 1) o de escritura (WE = 1) y la señal hit esta desactiva significa que el conjunto al que quiero acceder no lo tengo en la cache y si a su vez la señal

dirty_bit está desactiva significa que no tengo un conjunto en esa posición de la cache, por lo que puedo modificarlo trayéndome la correspondiente dirección de memoria principal. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | MC_send_addr | Reset_word | bus_RE |
|--------------|-------|--------------|------------|--------|
| Comunicación | 1 | 1 | 1 | 1 |

La señal frame la activo para avisar al slave de que quiero realizar una comunicación, MC_send_addr la activo para que el slave sepa que le estoy enviando la dirección, Reset_word la activo para que se ponga a cero el contador de palabras recibidas e enviadas y bus_RE la activo para que el slave sepa que operación quiero realizar en este caso de lectura.

6. Transición número 8 y 9 me encuentro en el estado comunicación (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| x | x | 0 | 1 OR 0 | x | x | |

En este estado simplemente me mantendré hasta que reciba la confirmación del slave de que la dirección que le he mandado está dentro de su rango, tengo en cuenta el dirty_bit para mantener activa la operación que estoy realizando si es lectura o escritura. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | bus_RE si Dirty_bit = 0 | bus_WE si Dirty_bit = 1 |
|--------------|-------|-------------------------|-------------------------|
| Comunicación | 1 | 1 | 1 |

La señal frame la activo para avisar al slave de que quiero mantener la comunicación, y la operación la mantengo para que no se olvide si es lectura o escritura.

7. Transición número 10 me encuentro en el estado comunicación (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| x | x | 1 | 1 OR 0 | x | x | |

Cuando recibo la confirmación del slave de que la dirección que le he mandado está dentro de su rango, bus_devsel = '1', paso al estado de transferencia, tengo en cuenta el dirty_bit para mantener activa la operación que estoy realizando si es lectura o escritura. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | bus_RE si Dirty_bit = 0 | bus_WE si Dirty_bit = 1 |
|---------------|-------|-------------------------|-------------------------|
| Transferencia | 1 | 1 | 1 |

La señal frame la activo para avisar al slave de que quiero mantener la comunicación, y la operación la mantengo para que no se olvide si es lectura o escritura.

8. Transición número 11 me encuentro en el estado transferencia (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| 0 | 1 | x | 0 | x | x | |

Cuando el slave me activa la señal bus_TRDY significa que está listo para enviar o recibir a datos, en este caso para recibir al estar dirty_bit desactivado. Si no es la última palabra significa que me mantengo en el estado de transferencia. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | Count_enable | bus_RE | Mux_origen | MC_WE |
|---------------|-------|--------------|--------|------------|-------|
| Transferencia | 1 | 1 | 1 | 1 | 1 |

La señal frame la activo para avisar al slave de que quiero mantener la comunicación, y la operación la mantengo para que no se olvide si es lectura o escritura, en este caso al tener el dirty_bit desactivado significa que recibo los datos del slave por eso mux_origen y MC_WE están activos para guardarlos en MC_data según me van llegando.

Count_enable esta activo para contar las palabras que vamos enviando.

9. Transición número 14 me encuentro en el estado transferencia (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| 0 | 1 | x | 1 | x | x | |

Cuando el slave me activa la señal bus_TRDY significa que está listo para enviar o recibir a datos, en este caso para enviar al estar dirty_bit activado. Si no es la última palabra significa que me mantengo en el estado de transferencia. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | Count_enable | bus_RE | Mux_origen | MC_send_data | MC_WE |
|---------------|-------|--------------|--------|------------|--------------|-------|
| Transferencia | 1 | 1 | 1 | 1 | 1 | 1 |

La señal frame la activo para avisar al slave de que quiero mantener la comunicación, y la operación la mantengo para que no se olvide si es lectura o escritura, en este caso al tener el dirty_bit activado significa que envío los datos del slave por eso mux_origen, MC_Send_data y MC_RE están activos para enviarlos desde MC_data al slave según nos vaya diciendo. Count_enable está activo para contar las palabras que vamos enviando.

10. Transición número 12 y 13 me encuentro en el estado transferencia (Autómata)

Las entradas para que se cumpla este estado son:

| Last_Word | Bus_TRDY | Bus_devsel | Dirty_bit | Hit | RE | WE |
|-----------|----------|------------|-----------|-----|----|----|
| x | 0 | x | x | x | x | |

Cuando el slave me desactiva la señal bus_TRDY significa que no está listo para enviar o recibir datos, por lo que espero en el estado actual. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | bus_RE si Dirty_bit = 0 | bus_WE si Dirty_bit = 1 |
|---------------|-------|----------------------------|----------------------------|
| Transferencia | 1 | 1 | 1 |

11. Transición número 15 me encuentro en el estado transferencia (Autómata)

Las entradas para que se cumpla este estado son:

| Last_Word | Bus_TRDY | Bus_devsels | Dirty_bit | Hit | RE | WE |
|-----------|----------|-------------|-----------|-----|----|----|
| 1 | 1 | x | 1 | x | x | |

Cuando se da este caso es que envió la última palabra del conjunto (last_word = '1') y además si está activo el dirty_bit significa que estaba enviando un bloque sucio, por lo que paso al estado de transición dado que después traeré otro bloque antes de activar la señal ready. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | Count_enable | bus_RE | Mux_origen | MC_send_data | MC_WE |
|------------|-------|--------------|--------|------------|--------------|-------|
| Transición | 1 | 1 | 1 | 1 | 1 | 1 |

| Replace_block | Update_dirty |
|---------------|--------------|
| 1 | 1 |

Similar a la transición 14 pero en este caso al ser el último actualizo el dirty_bit de ese conjunto poniéndolo como limpio (Replace_block = '1').

12. Transición número 16 me encuentro en el estado transferencia (Autómata)

Las entradas para que se cumplen en este estado son:

| Last_Word | Bus_TRDY | Bus_devsels | Dirty_bit | Hit | RE | WE |
|-----------|----------|-------------|-----------|-----|----|----|
| 1 | 1 | x | 0 | x | x | |

Cuando se da este caso es que envió la última palabra del conjunto (last_word = '1') y además si está desactivo el dirty_bit significa que estoy trayéndome un bloque a la cache. Las salidas que en este caso cambiaré serán:

| Next_state | Frame | Count_enable | bus_RE | Mux_origen | MC_send_data | MC_WE |
|------------|-------|--------------|--------|------------|--------------|-------|
| inicio | 0 | 1 | 1 | 1 | 1 | 1 |

| MC_tags_WE | Update_dirty |
|------------|--------------|
| 1 | 1 |

Similar a la transición 11 pero en este caso al ser el último actualizo el dirty_bit de ese conjunto poniéndolo como sucio (Replace_block = '0') y cargando la etiqueta en el MC_tags.

13. Transición número 17 me encuentro en el estado transición (Autómata)

Simplemente se utiliza para desactivar la señal frame y que el slave sepa que vamos a cambiar de operación.

B) Modificaciones en el fichero UC_MC

Las modificaciones realizadas en este fichero simplemente fueron el añadir la señal de reset_word que se encarga esta señal de resetear el contador del número de palabras y el implementar el autómata anteriormente explicado.

C) Modificaciones realizadas en el MIPS

- Sustituimos la MD de datos como se nos pide en el proyecto por MD_mas_MC.
- Añadimos a la unidad de detención algunas entradas más, como es si está realizando una operación o no, si es escritura (WE_MD) o si es lectura (RE_MD) y la señal Mem_Ready.
- Añadimos otra modificación que es el parar el la carga de datos en el banco WB debido a que si se produce una anticipación y no lo hacemos se pierde el dato y el resultado es incorrecto, posteriormente en las pruebas explicaremos el porqué.

D) Modificaciones realizadas en la Unidad de detención

- En el Kill_If realizamos ese añadido de comparar con esos registros debido a que en caso de que se realiza una operación delante del beq que modifique el resultado de uno de los registros que tenemos en cuenta para el salto, puede dar lugar a que no se produzca el salto el cual si solo pusiéramos PCSRC=1 puede que se diera. Esto lo hemos concluido por una prueba realizada que explicaremos posteriormente, por qué, en el apartado de las pruebas.
- Hemos añadido dos apartados de parada en el Parar_ID y en el Parar_EX en caso de que este activo Mem_ready y haya una operación activa de lectura o escritura.

E) Modificaciones realizadas en la cache

Al no saber si era necesario que el procesador no fallase si se disponía de una dirección mayor de la que se puede procesar en la memoria principal (0x000001FF) hemos añadido una entrada más a la unidad de control de la memoria cache, out_range que si se activa devuelve como dato el valor del bus_B y no interactúa la dirección con el slave debido a que como comprobamos en una prueba daba errores.

3) Impacto en rendimiento de cada una de las modificaciones con respecto al procesador inicial

Inicialmente tenemos un procesador el cual no poseía una memoria cache, este cambio supone que al hacer un load o un store puedan producirse tres sucesos, que se produzca un acierto que supondría un coste en tiempo igual al anterior procesador, que se produzca un fallo limpio entonces tardara algunos ciclos más al tener que traernos el dato de memoria o que se produzca un fallo sucio que tardara bastante al tener que mandar el bloque actual que tenemos en la cache a memoria y traernos el que queremos modificar. Tanto la unidad de detención como la de anticipación siguen funcionando de la misma manera que en el procesador anterior.

4) Pruebas realizadas para verificar que funciona correctamente

A continuación se presentan todas las pruebas que hemos realizado para verificar el correcto funcionamiento del procesador. En ellas se presentara una tabla con los datos que hemos asignado en las regiones de memoria y las instrucciones utilizadas en el código, que deberían de dar, al final hemos incluido lo que se buscaba verificar con las pruebas realizadas.

Al incluir estas pruebas estamos completamente seguros de que su funcionamiento es el correcto y que sigue las medidas expresadas por los profesores.

4.1 Prueba 1

Tabla de datos

| | | | | | |
|-----|----------|-----|----------|-----|----------|
| @0 | 00000001 | @20 | 00000009 | @50 | 10000000 |
| @4 | 00000002 | @24 | 0000000A | @54 | 20000000 |
| @8 | 00000003 | @28 | 0000000B | @58 | 30000000 |
| @C | 00000004 | @2C | 0000000C | @5C | 40000000 |
| @10 | 00000005 | @30 | 0000000D | @70 | 50000000 |
| @14 | 00000006 | @34 | 0000000E | @74 | 60000000 |
| @18 | 00000007 | @38 | 0000000F | @78 | 70000000 |
| @1C | 00000008 | @3C | 00000010 | @7C | 80000000 |

Código y lo que hace

| | |
|---------------------|--|
| @0x0 LW R1, (R0) | ;fallo limpio lleno el conjunto 00 <-(1,2,3,4) |
| @0x4 LW R1, 16(R0) | ;fallo limpio lleno el conjunto 01 <-(5,6,7,8) |
| @0x8 LW R1, 32(R0) | ;fallo limpio lleno el conjunto 10 <-(9,A,B,C) |
| @0xC LW R1, 48(R0) | ;fallo limpio lleno el conjunto 11 <-(D,E,F,10) |
| @0x10 LW R1, 8(R0) | ;acierto conjunto 00 palabra 8 |
| @0x14 LW R1, 24(R0) | ;acierto conjunto 01 palabra 8 |
| @0x18 LW R1, 44(R0) | ;acierto conjunto 10 palabra c |
| @0x1C LW R1, 52(R0) | ;acierto conjunto 11 palabra 4 |
| @0x20 LW R1, 80(R0) | ;fallo sucio conjunto 01 ->(5,6,7,8) y <-(10000000,20000000,30000000,40000000) |

@0x24 LW R1, 112(R0) ;fallo sucio conjunto 11 ->(D,E,F,10) y <= (50000000,60000000,70000000,80000000)

Que se quiere comprobar con esta prueba:

- Con las 4 primeras instrucciones

Con esta prueba se quiere ver si se llenan bien los cuatro conjuntos en la memoria cache forzando a que se produzcan cuatro fallos limpios al estarla memoria vacía. Comprobamos sobre todo si ponía de forma correcta el bloque como sucio y activaba el valid_bit de forma correcta.

Desde el principio el funcionamiento de este fallo limpio lo realizo de forma correcta.

- Con las instrucción 4 – 8

Se busca si realiza bien los aciertos que comprueba en un ciclo y que nos selecciona la palabra que queremos de forma correcta. (No tuvimos ningún tipo de problema)

- Con las 2 últimas instrucciones

Se busca que el fallo sucio lo realiza de forma correcta que primero envía lo que hay almacenado en ese conjunto a memoria principal por si se ha actualizado de alguna manera, y que luego se trae de memoria principal los datos correspondientes. Miramos que limpia bien el dirty_bit al enviar los datos MP, y que pone valid_bit a 0.

Al principio realizamos el autómata en tres estados no considerando necesario el que el Frame estuviera a '0' un ciclo debido a que se cumplía y funcionaba, lo que sucedía es que activaba la señal de ready y el procesador continuaba sin tener el dato correcto, para no complicar más el estado de inicio optamos por añadir un estado más (Estado de Transición).

4.2 Prueba 2

Tabla de datos

| | | | | | |
|-----|----------|-----|----------|-----|----------|
| @0 | 3e4cccd | @20 | 00000009 | @50 | 10000000 |
| @4 | 00000002 | @24 | 0000000A | @54 | 20000000 |
| @8 | 00000003 | @28 | 0000000B | @58 | 30000000 |
| @C | 00000004 | @2C | 0000000C | @5C | 40000000 |
| @10 | 00000005 | @30 | 0000000D | @70 | 50000000 |
| @14 | 00000006 | @34 | 0000000E | @74 | 60000000 |
| @18 | 00000007 | @38 | 0000000F | @78 | 70000000 |
| @1C | 00000008 | @3C | 00000010 | @7C | 80000000 |

Código y lo que hace

```
@0x0 LW R1, (R0) ;fallo limpio lleno el conjunto 00 <-(1,2,3,4)
@0x4 LW R1, 16(R0) ;fallo limpio lleno el conjunto 01 <-(5,6,7,8)
@0x8 LW R1, 32(R0) ;fallo limpio lleno el conjunto 10 <-(9,A,B,C)
@0xC LW R1, 48(R0) ;fallo limpio lleno el conjunto 11 <-(D,E,F,10)
@0x10 ADDFP R3, R1, R1 ;R3 = 0.4
```

| | |
|------------------------|--|
| @0x14 STR R3, 8(R0) | ;acierto conjunto 00 palabra 8 |
| @0x18 STR R3, 24(R0) | ;acierto conjunto 01 palabra 8 |
| @0x1C STR R3, 44(R0) | ;acierto conjunto 10 palabra c |
| @0x20 STR R3, 52(R0) | ;acierto conjunto 11 palabra 4 |
| @0x24 LW R1, 32768(R0) | ;dirección que no está en memoria fuera de rango |
| @0x28 STR R3, 84(R0) | ;fallo sucio conjunto 01 p4 ->(5,6,7,8) y <-(10000000,20000000,30000000,40000000) |
| @0x2C STR R3, 120(R0) | ;fallo sucio conjunto 11 p8 ->(D,E,F,10) y <=(50000000,60000000,70000000,80000000) |

Que se quiere comprobar con esta prueba:

- Con las 4 primeras instrucciones (igual que en el caso anterior)
- Con la instrucción número 5

Se busca si realiza el addfp teniendo a la vez en ejecución la interacción de la memoria cache con la memoria principal, activada debido a un fallo limpio,

Se observó que funcionaba de forma correcta aun que se perdía la señal de done como el procesador estaba parado completamente no afectaba de ninguna manera.

- Con las 6-9 instrucciones

Al igual que en la prueba anterior son aciertos pero esta vez con la instrucción store nos fijamos de que se modifican de forma correcta las regiones de memoria seleccionadas y vemos que funciona correctamente.

- Con la instrucción número 10 (@0x24 LW R1, 32768(R0))

Con instrucción se busca ver cómo reacciona la memoria cache al pasarle una región de memoria que no debería de procesar al estar fuera de su rango, la idea que tuvimos fue que la unidad de control de la cache la detectase y que no hiciera nada que no diera ninguna salida, lo que supone que el procesador coja como valor el del bus_B.

Decidimos hacer esta modificación porque al hacer una prueba con un valor que la MP no detectaba el master se quedaba esperando la respuesta del slave lo que daba lugar a un bloqueo de las instrucciones, lo que consideramos un problema.

- Con las 2 ultimas instrucciones (igual que en el caso anterior) pero store

4.3 Prueba 3

Tabla de datos

| | | | | | |
|-----|----------|-----|----------|-----|----------|
| @0 | 3e4cccd | @20 | 00000009 | @50 | 10000000 |
| @4 | 00000002 | @24 | 0000000A | @54 | 20000000 |
| @8 | 00000003 | @28 | 0000000B | @58 | 30000000 |
| @C | 00000004 | @2C | 0000000C | @5C | 40000000 |
| @10 | 00000005 | @30 | 0000000D | @70 | 50000000 |
| @14 | 00000006 | @34 | 0000000E | @74 | 60000000 |
| @18 | 00000007 | @38 | 0000000F | @78 | 70000000 |
| @1C | 00000008 | @3C | 00000010 | @7C | 80000000 |

Código y lo que hace

```

@0x0  LW  R1, C(R0)    ;fallo limpio lleno el conjunto 00 <-(3e4cccd,2,3,4) palabra c =>(4)
@0x4  ADD R2,R1,R1      ;R2 = 4 + 4 = 8
@0x8  LW  R1, 8(R0)     ;acierto conjunto 00 palabra 8 =>(3)
@0xC  ADD R3,R2,R2      ;R3 = 8 + 8 = 16
@0x10 LW  R2, 48(R0)    ;fallo limpio lleno el conjunto 11 <-(D,E,F,10) palabra 0 =>(D)
@0x14 STR R3, 120(R0)   ;fallo sucio conjunto 11 p8 ->(D,E,F,10) y <=
                               (50000000,60000000,70000000,80000000)
@0x18 ADD R3,R2,R2      ;R3 = 13 + 13 = 26
@0x1C LW  R1, 0(R0)     ;acierto conjunto 00 palabra 0 =>(3e4cccd)
@0x20 LW  R2, C(R0)     ;fallo limpio lleno el conjunto 01 <-(5,6,7,8) palabra c =>(8)
@0x24 ADDFP R3, R1, R1   ;R2 = 0.4

```

Que se quiere comprobar con esta prueba:

- Con la instrucción (@0x4 ADD R2,R1,R1)

Ver si se realizan de forma las paradas, si la unidad de detención inserta una nop por medio al haber dependencia y si se produce la anticipación de forma correcta, cuando se produce un fallo limpio.

- Con la instrucción (@0xC ADD R3,R2,R2)

Si se produce bien la anticipación con un acierto en este caso desde la zona de escritura del procesador (WB).

- Con la instrucción (@0x18 ADD R3,R2,R2)

Si se produce la anticipación de forma correcta desde escritura (WB) cuando se debe de mantener al estar el procesador para por un fallo sucio.

Aquí tuvimos que solucionar un error que al principio no considerábamos necesario el parar el banco de registros Mem/WB y vimos que si debido a que la operación se tenía que mantener hasta que finalizase tanto un fallo sucio como un fallo limpio. (De ahí la variable load_Mem)

- Con la instrucción (@0x24 ADDFP R3, R1, R1)

Se busca ver lo mismo que con la instrucción anterior pero para un ADDFP y vemos que el resultado se mantiene de forma correcta.

4.4 Prueba 4

Tabla de datos

| | | | | | |
|-----|----------|-----|----------|-----|----------|
| @0 | 00000001 | @20 | 00000009 | @40 | 11111111 |
| @4 | 00000002 | @24 | 0000000A | @44 | 22222222 |
| @8 | 00000003 | @28 | 0000000B | @48 | 33333333 |
| @C | 00000004 | @2C | 0000000C | @4C | 44444444 |
| @10 | 00000005 | @30 | 0000000D | @70 | 50000000 |
| @14 | 00000006 | @34 | 0000000E | @74 | 60000000 |
| @18 | 00000007 | @38 | 0000000F | @78 | 70000000 |
| @1C | 00000008 | @3C | 00000010 | @7C | 80000000 |

Código y lo que hace

```
@0x0  LW  R1, C(R0)           ;fallo limpio lleno el conjunto 00 <-(1,2,3,4) palabra c =>(4)
@0x4  BEQ  R1, R2 #+1=sal1    ;R1 = R2 salto al ADD (hasta que no termine el load son iguales)
@0x8  ADD  R2,R1,R1           ;R2 = 4 + 4 = 8
@0xC  ADD  R2,R2,R2           ;R2 = 8 + 8 = 16 y si hubiera salto seria R2 = 0 + 0 = 0
@0x10 LW  R1, 8(R0)          ;acierto conjunto 00 palabra 8 =>(3)
@0x14 BEQ  R1, R1 #+1=sal2    ;R1 = R1 salto al ADD
@0x18 ADD  R2,R1,R1           ;R2 = 3 + 3 = 6
@0x1C ADD  R2,R1,R1           ;R2 = 3 + 3 = 6
@0x20 LW  R1, 72(R0)          ;fallo sucio conjunto 00 ->(1,2,3,4) y <=
                                   (11111111,22222222,33333333,44444444) palabra 8 =>(33333333)
@0x24 BEQ  R1, R1 #+1=sal3    ;R1 = R1 salto al ADD
@0x28 ADD  R2,R1,R1           ;R2 = 33333333 + 33333333 = 66666666
@0x2C ADD  R2,R1,R1           ;R2 = 33333333 + 33333333 = 66666666 hexadecimal
@0x14 LW  R2, 16(R0)          ;fallo limpio lleno el conjunto 01 <-(5,6,7,8)
@0x18 beq  R2,R2, fin         ;R3 = 33333333 + 33333333 = 66666666
```

Que se quiere comprobar con esta prueba:

- Con la instrucción (@0x4 BEQ R1, R2 #+1=sal1)

Con este salto probamos si hay una espera correcta tras tener una dependencia del dato obtenido de un fallo limpio.

- Con la instrucción (@0x14 BEQ R1, R1 #+1=sal2)

Para que se produzca el salto de forma correcta se debe de producir el load de forma correcta al haber dependencia, en este caso es un acierto y se producirá de forma rápida.

- Con la instrucción (@0x24 BEQ R1, R1 #+1=sal3)

Esta instrucción la utilizamos para ver si el salto espera tras haber una dependencia de un fallo sucio.

- Con la instrucción (@0x18 beq R2,R2, fin)

Esta ultima la realizamos para ver si al estar produciéndose un fallo limpio o fallo sucio es decir al tardar el load varios ciclos, y el beq no tener dependencia si se ejecuta de forma correcta y luego se espera hasta que termine.

4.5 Prueba 5

Tabla de datos

| | | | | | |
|-----|----------|-----|----------|-----|----------|
| @0 | 00000001 | @20 | 00000009 | @40 | 0000000A |
| @4 | 00000002 | @24 | 0000000A | @44 | 0000000B |
| @8 | 00000003 | @28 | 0000000B | @48 | 0000000C |
| @C | 00000004 | @2C | 0000000C | @4C | 0000000D |
| @10 | 00000005 | @30 | 0000000D | @70 | 50000000 |
| @14 | 00000006 | @34 | 0000000E | @74 | 60000000 |
| @18 | 00000007 | @38 | 0000000F | @78 | 70000000 |
| @1C | 00000008 | @3C | 00000010 | @7C | 80000000 |

Código y lo que hace

```

@0x0  LW  R1, C(R0)           ;fallo limpio lleno el conjunto 00 <-(1,2,3,4) palabra c =>(4)
@0x4  ADD R2,R1,R1           ;R2 = 4 + 4 = 8
@0x8  ADD R3,R2,R2           ;R3 = 8 + 8 = 16
@0xc  LW  R4, 72(R0)         ;fallo sucio conjunto 00 ->(1,2,3,4) <-
                                (0000000A,0000000B,0000000C,0000000D) => palabra 8 (0000000C)
@0x10 ADD R3,R3,R3           ;R3 = 16 + 16 = 32
@0x14 LW  R2, 16(R0)         ;fallo limpio lleno el conjunto 01 <-(5,6,7,8)
fin    @0x18 beq R3,R3, fin    ;R3 = R3 salto obligatorio

```

Que se quiere comprobar con esta prueba:

Al igual que en los casos anteriores esta prueba lo que busca es si en el final del programa se produce el salto de forma correcta al actualizar el registro durante una parada del procesador y producirse el salto, entonces se debe de esperar a cargar la siguiente instrucción hasta que finalice el fallo limpio.

4.6 Prueba 6

Tabla de datos

| | |
|-----|----------|
| @0 | 00000008 |
| @4 | 0000000C |
| @8 | 00000000 |
| @C | 00000003 |
| @10 | 3e4ccccd |
| @14 | 00000001 |
| @18 | 00000000 |
| @1C | 00000002 |

Código y lo que hace

```

@0x0  LW  R1, 0(R0)           ;fallo limpio lleno el conjunto 00 <-(8,C,0,3) R1= palabra 0 =>(8)
@0x4  ADD R2,R1,R1           ;R2 = 8 + 8 = 16
inm    @0x8  LW  R3, 4(R0)       ;acierto conjunto 00 palabra 4 =>(C)    R3 = C
@0xC  ADD R4,R2,R2           ;R4 = 16 + 16 = 32
@0x10 ADD R5,R4,R3           ;R4 = 32 + 12 = 44
@0x14 SW  R5, 8(R0)           ;acierto conjunto 00 palabra 8 =>(0) = (44)
@0x18 BEQ R5, R0 #-5=inm       ;R5 != R0 no se produce salto R0=0 != R5=44
@0x1C ADD R0,R0,R0           ;R0 = 0 + 0 = 0
@0x20 LW  R0, 0(R1)           ;acierto conjunto 00 palabra 8 =>(44)    R0 = 44
@0x24 NOP
@0x28 LW  R0, 12(R6)          ;acierto conjunto 00 palabra C =>(3)    R0 = 3
@0x2C BEQ R3, R0 #-1          ;no se produce salto R0=3 != R3=12
@0x30 LW  R1, 16(R6)          ;fallo limpio lleno el conjunto 01 <-(3e4ccccd,1,0,2)
R1=palabra 0 =>(3e4ccccd)
@0x34 ADDFP R2, R1, R1        R2 = 0.2 + 0.2 = 0.4
@0x38 ADDFP R3, R2, R1        R3 = 0.4 + 0.2 = 0.6
@0x3C SW  R2, 20(R6)          ;acierto conjunto 01 palabra 4 =>(1) = (0.6)
fin    @0x40 BEQ R0, R0 #-1 = fin ;se produce salto R0=3 != R3=12

```

Que se quiere comprobar con esta prueba:

Esta prueba es el mismo código que nos entregó el profesor para solucionar los problemas que podíamos tener en el proyecto 1. Con esta prueba queremos ver que el funcionamiento del procesador no se ve afectado con respecto a los resultados de acuerdo al MIPS.

5) Conclusiones y cuantificación de horas dedicadas por cada miembro.

El proyecto en si ha sido asequible, se ha buscado programar el autómata de forma correcta y ver que la base del funcionamiento de la memoria cache era la correcta como se ha demostrado en las pruebas 1 y 2. Luego hemos intentado comprobar todas las instrucciones en el procesador aritmética, saltos, loads y stores, y ver como actuaban en los diferentes casos que se pueden presentar, fallo limpio, fallo sucio o acierto, con anticipación mientras se está produciendo un parada del procesador debida a la cache, y con salto a la vez que hay una parada. También se ha buscado el tener una memoria algo más especializada y esquematizada de acuerdo a la entregada en el proyecto 1.

Con respecto a la cuantificación de las horas dedicadas:

- A la parte del estudio de las fuentes y el entender los componentes que se nos daban y para que servían las diferentes señales de la cache 3 horas.
- Para el diseño de la unidad de control hemos tardado algo unas 4 horas pero porque a la vez hemos realizado pruebas y hemos ido mirando problemas, no solo hemos realizado el esquema.
- Al apartado de depuración y ajustes hemos invertido unas 18 horas debido al tiempo que nos ha supuesto el realizar las pruebas y el diseñar los códigos para que se produjesen los diferentes casos que se pueden dar.
- La memoria ha sido algo más larga de lo estimado unas 4,5 horas debido a los hipervínculos y el diseño del autómata principalmente.