Memoria Técnica del Proyecto 2

Arquitectura y Organización de Computadores 2

2º curso, Grado de Ingeniería en Informática, 2018-2019

Patricia Briones Yus, 735576

Índice de contenidos

1. Diseño del sistema

El sistema implementado, sustituye la memoria de datos del procesador MIPS por una memoria cache y una memoria principal, conectadas entre sí a través de un bus semi-síncrono basado en el bus PCI.

La MC es de emplazamiento directo, escritura con actualización inmediata y política write_around en fallo de escritura. Es decir, en caso de ser:

- Read hit: devuelve la palabra de la MC.
- Read miss: MP transfiere el bloque a MC y se guarda en la MC y devuelve la palabra.
- Write hit: actualiza la palabra en la MC y en la MP.
- Write miss: escribe la palabra en la MP.

El controlador de la memoria cache (se trata más adelante) recibe las solicitudes del MIPS. Si puede, las responde con los datos almacenados en la citada MC. En caso contrario, realiza las transferencias necesarias por el bus con la memoria principal. El procesador MIPS no podrá avanzar hasta que haya terminado la transferencia.

Por último, para llevar el control de las causas que perjudican al rendimiento (dónde se pierden más ciclos por paradas en el programa) se utilizan varios contadores, tanto en el MIPS como en la MC.

2. Modificación del hardware

Se modifica el fichero del procesador MIPS tal y como se indica en el documento de texto proporcionado por los profesores. Se adjunta como **Anexo I**.

2.1 Señal Mem_Ready

La señal Mem_Ready evita que el procesador MIPS ejecute su siguiente instrucción en caso de que se esté procesando una transferencia por el bus. Para impedir esto, hay que modificar la señal load_PC <= '1' when (avanzar_ID and Mem_ready)='1' else '0'. Es decir, si Mem_Ready vale 0 significa que no puede cargarse la siguiente instrucción.

Debido a la posible anticipación de operandos en las instrucciones posteriores, también es necesario parar el procesador en la etapa de escritura. Para mayor legibilidad del código se ha creado una señal nueva, MemRegWrite <= '0' when Mem_ready = '0' else RegWrite_MEM, que corresponderá al valor de RegWrite_MEM del Banco_MEM_WB.

2.2 Contadores de rendimiento

Se han añadido en total 9 contadores al sistema: 3 en la MC, 6 en el procesador MIPS. Los de la MC se incrementan según las señales inc_rm, inc_wm y inc_wh de la unidad de control de la MC, siendo respectivamente fallos de lectura, fallos de escritura y aciertos de escritura.

En cuanto a los contadores del MIPS, se incrementan cuando:

- cont_ciclos: siempre.
- cont_paradas_control: hay un error de predicción en el salto.
- cont_paradas_datos: avanzar_ID vale 0 (hay riesgos de datos).
- cont_paradas_memoria: depende del valor de Mem_ready (not Mem_ready).
- cont_mem_reads: detecta que la instrucción es un LW.
- cont_mem_writes: detecta que la instrucción es un SW.

3. Diagrama de estados

Para una mejor comprensión se han separado los distintos casos que se pueden dar : fallos y aciertos de lecturas y escrituras. Los estados son los mismos en todos los casos. El diagrama de estados es de tipo Mealy para de esta manera reducir un ciclo el tiempo de respuesta.

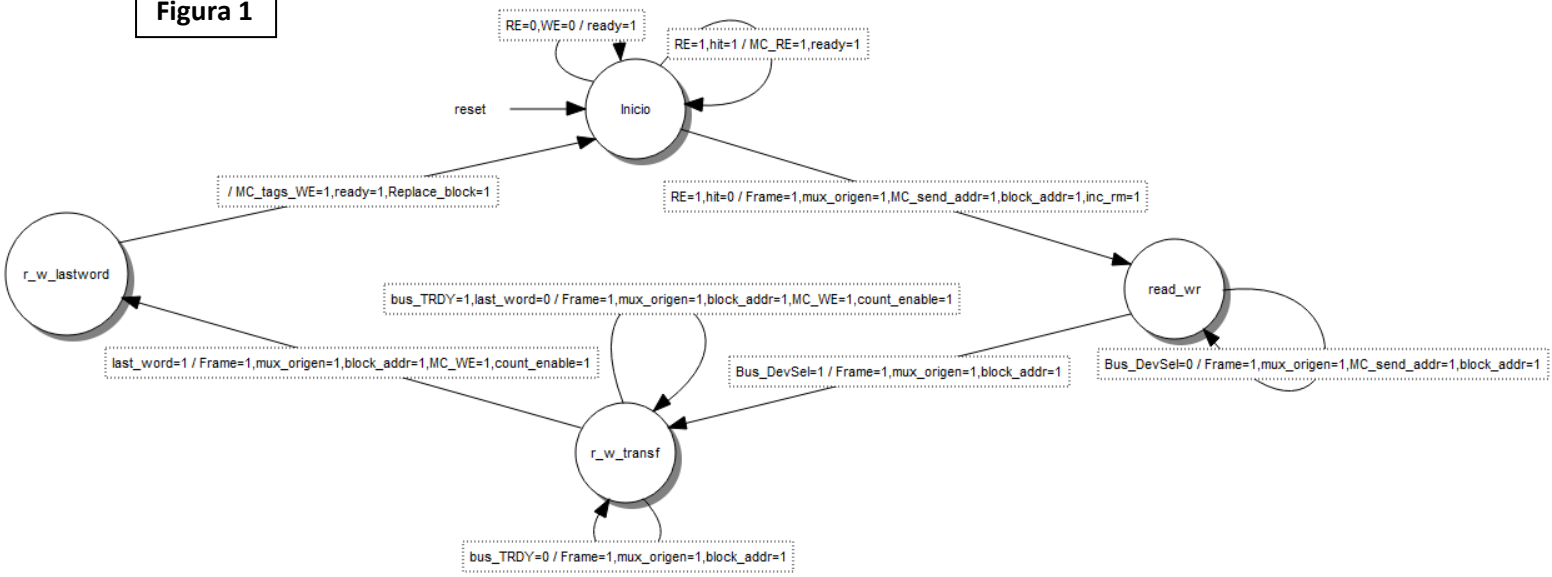
El caso general de no realizar ninguna acción está presente en las tres figuras. Por lo tanto ni se lee ni se escribe y por ende se mantiene en el mismo estado (Inicio) y ready=1.

La **figura 1** representa el acierto y fallo en lectura:

a) Si es acierto, significa que el dato que queremos obtener está en la MC y por tanto solo habría que leerlo en el mismo ciclo (MC_RE=1).
b) Si resulta ser fallo en lectura:

- Se activa la señal frame para indicar al MIPS que no puede avanzar, el bloque que se va a actualizar en la MC vendrá de la MP, así que mux_origen=1, se envía al bus la dirección del bloque (block_addr=1) y aumenta el contador de fallos de lectura.
- Se mantiene en el estado read_wr mandando la dirección hasta que el esclavo active la señal bus_DevSel (que indica que el esclavo ha reconocido la dirección) y pasa al estado siguiente, r_w_transferencia, dejando de enviar la dirección de bloque.
- Mientras la MP no esté preparada (bus_TRDY=0), MC espera. Una vez TRDY se active, el esclavo comenzará a enviar los datos del bloque a través del bus. Cuando llegue a la última palabra del bloque, pasará al estado r_w_lastword.
- Del último estado pasa al inicial sin ninguna condición, indicando que la transferencia ha terminado (frame=0), se actualizan las etiquetas de la MC y se señala que se ha reemplazado el bloque en memoria cache.

Figura 1

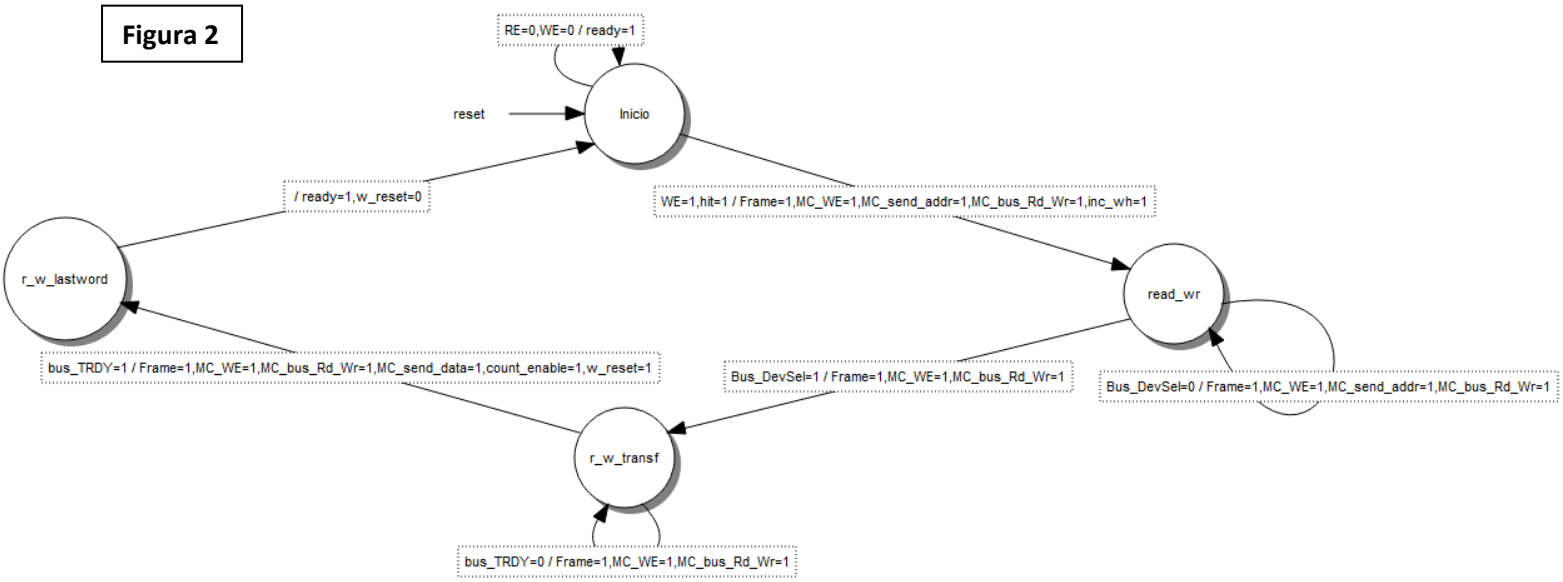


Para el caso de la figura 2 y 3 se ha añadido una nueva señal w_reset, que reinicia el contador de palabras transferidas, ya que en estos dos casos solo es necesario transmitir una palabra.

La **figura 2** representa los aciertos en escritura:

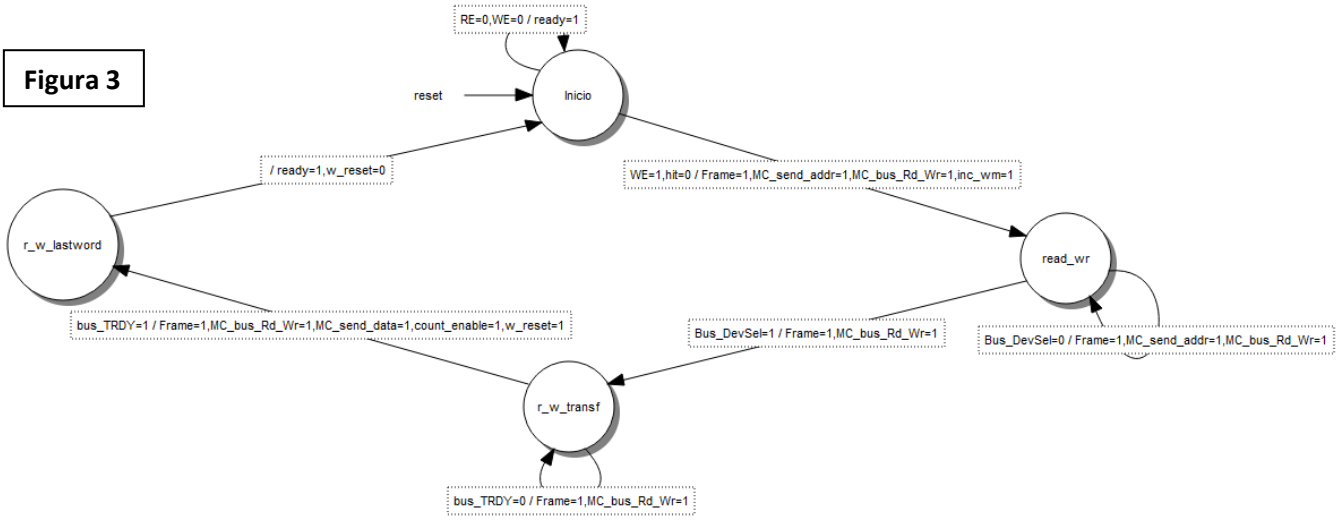
- Al ser un acierto se tendrá que actualizar la MC (MC_WE=1) y la MP (MC_bus_Rd_Wr=1), enviaremos la dirección de la palabra e incrementaremos el contador por write hit.
- Se mantiene en el estado read_wr mandando la dirección hasta que el esclavo active la señal bus_DevSel (indica que el esclavo ha reconocido la dirección) y pasa al estado siguiente, r_w_transferencia, dejando de enviar la dirección.
- Mientras la MP no esté preparada (bus_TRDY=0), MC espera. Una vez TRDY se active, el esclavo recibirá la palabra, la escribirá y reiniciará el contador de palabras a 0 (w_reset=1).
- Del último estado pasa al inicial sin ninguna condición, indicando que la transferencia ha terminado (frame=0), y vuelve a poner w_reset a 0.

Figura 2



La **figura 3** representa los fallos en escritura. Al ser un fallo, solo se actualizará el dato en la memoria principal (MC_bus_Rd_Wr=1), se enviará la dirección de la palabra y aumentará el contador de write miss. El resto de los estados hacen lo mismo que en el caso anterior.

Figura 3



3.1 Descripción algorítmica

- **Acierto de lectura:** necesita solo 1 ciclo para realizarse.

- **Fallo de lectura:** se necesita llevar el bloque de 4 palabras de MP a MC, por tanto necesita 1 ciclo para procesar que tipo de evento es y CrB(MP). Siendo CrB(MP)= 7 (ciclos para la primera palabra) + 3*2 (el nº de palabras restantes por el número de ciclos que se tarda en procesar cada una).

- **Acierto de escritura:** escribe una palabra en MC y en MP. Necesita 1 ciclo para procesar el evento y CwW(MC)+CwW(MP). Siendo CwW(MC)= 1 (ciclo que tarda en escribir en MC) y CwW(MP)= 6 (ciclos que tarda en escribir la palabra en MP).

- **Fallo de escritura:** escribe la palabra solo en MP (política write_around). Necesita 1 ciclo para procesar el evento y CwW(MP)= 7 (ciclos que tarda en escribir la palabra en MP).

A partir de este algoritmo, podemos deducir que la fórmula de los ciclos efectivos sería:

$$C_{ef} = 1 + \frac{\sum rm \cdot (CrB(MP)+1)}{\sum R} + \frac{\sum wh \cdot (CwW(MC)+CwW(MP)+1)}{\sum R} + \frac{\sum wm \cdot (CwW(MP)+1)}{\sum R}$$

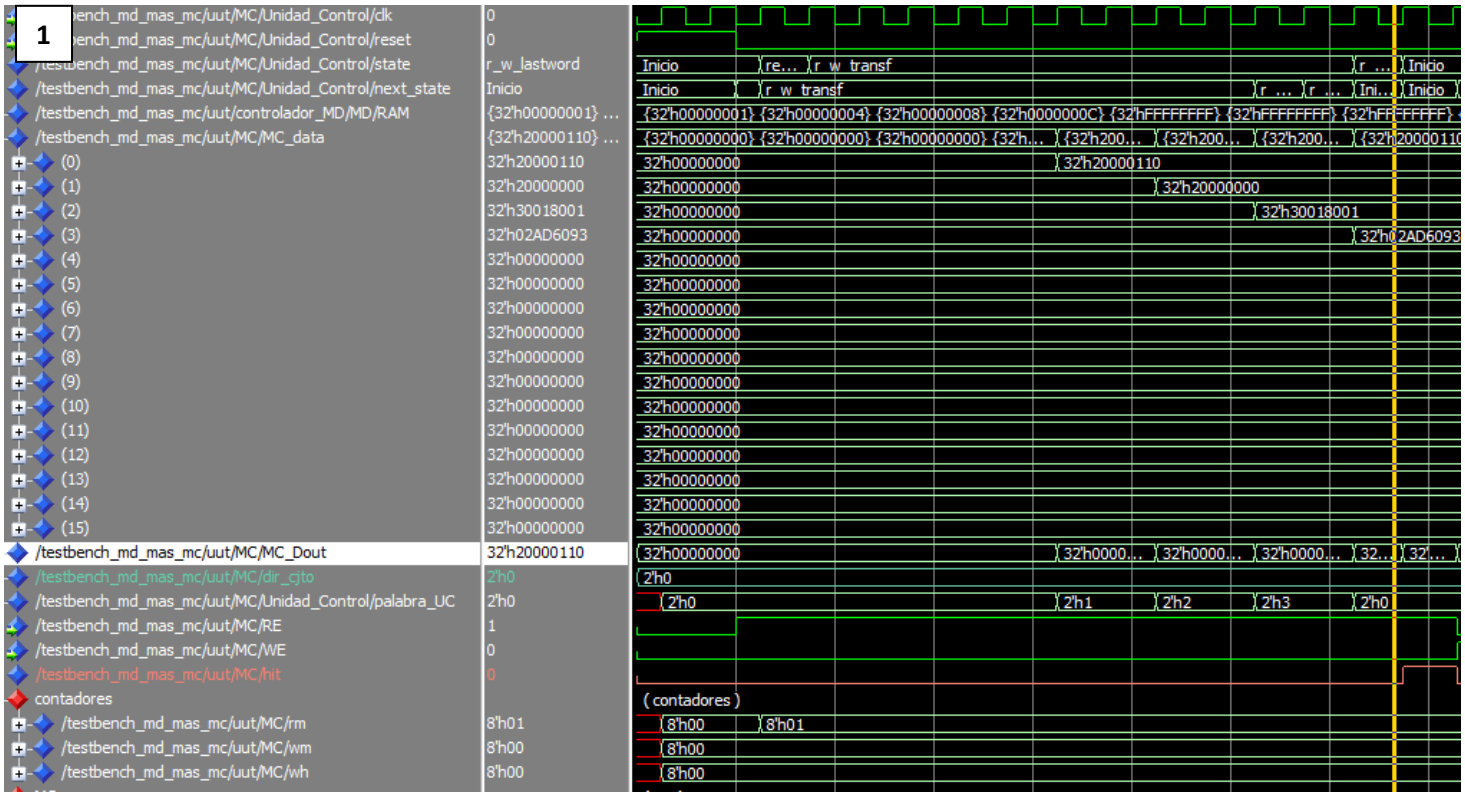
4. Verificación del correcto funcionamiento

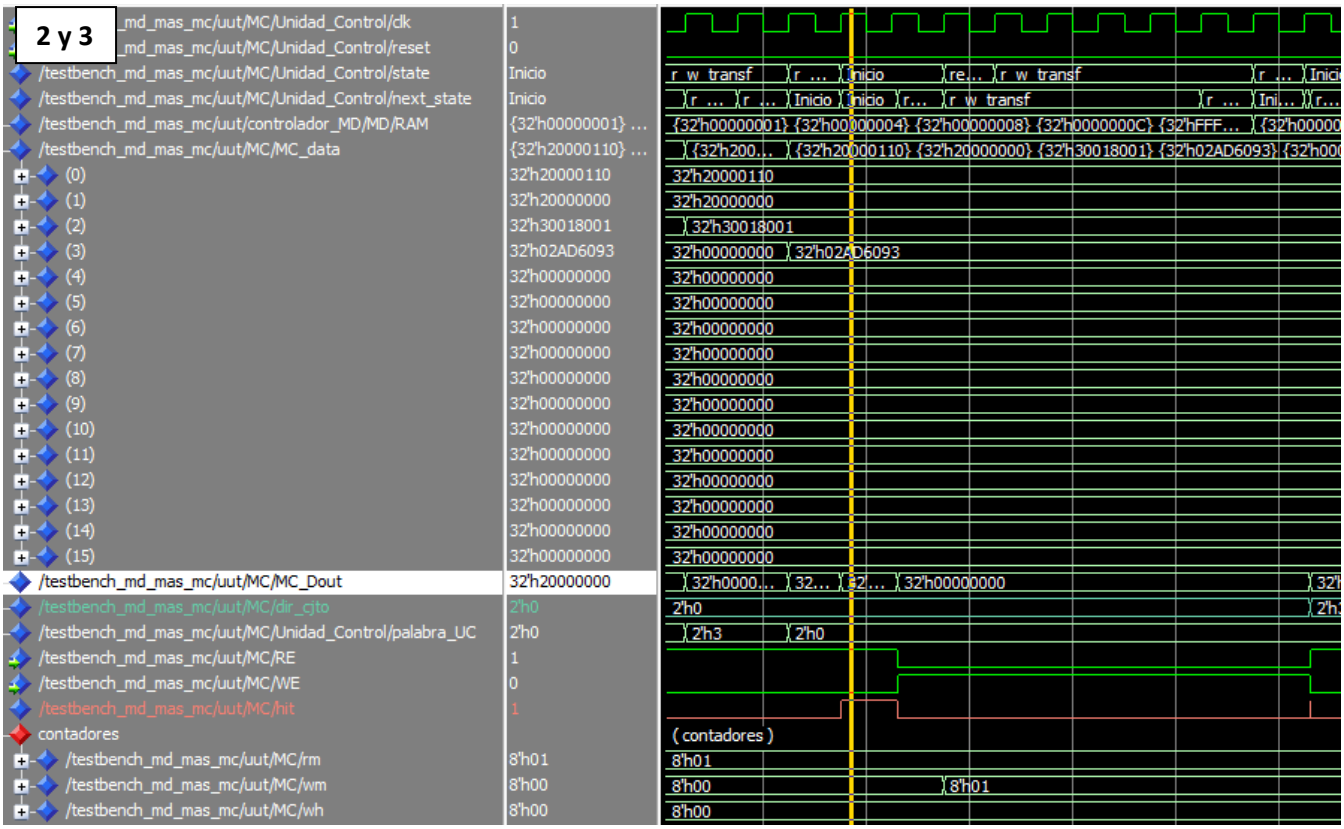
Para comprobar que el diseño del controlador funciona correctamente para todos los casos posibles, se ha programado un banco de pruebas (fichero “testbench_verificacion”) que consiste en:

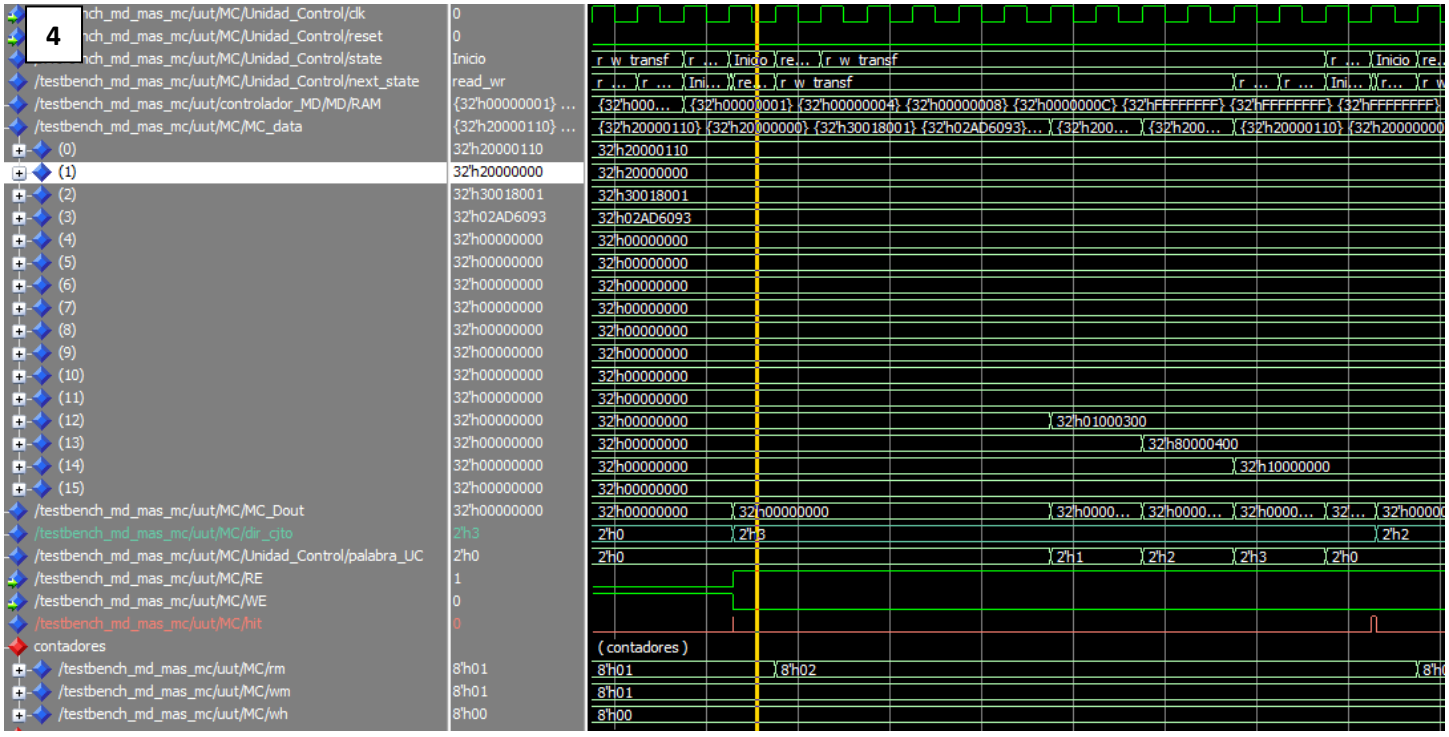
1. Fallo de lectura en el conjunto 0.
2. Acierto de lectura en el conjunto 0.
3. Fallo de escritura en el conjunto 1.
4. Fallo de lectura en el conjunto 3.
5. Fallo de lectura en el conjunto 2.
6. Acierto de escritura en el conjunto 2.
7. Acierto de escritura en el conjunto 0.

Quedando los contadores de la siguiente manera:

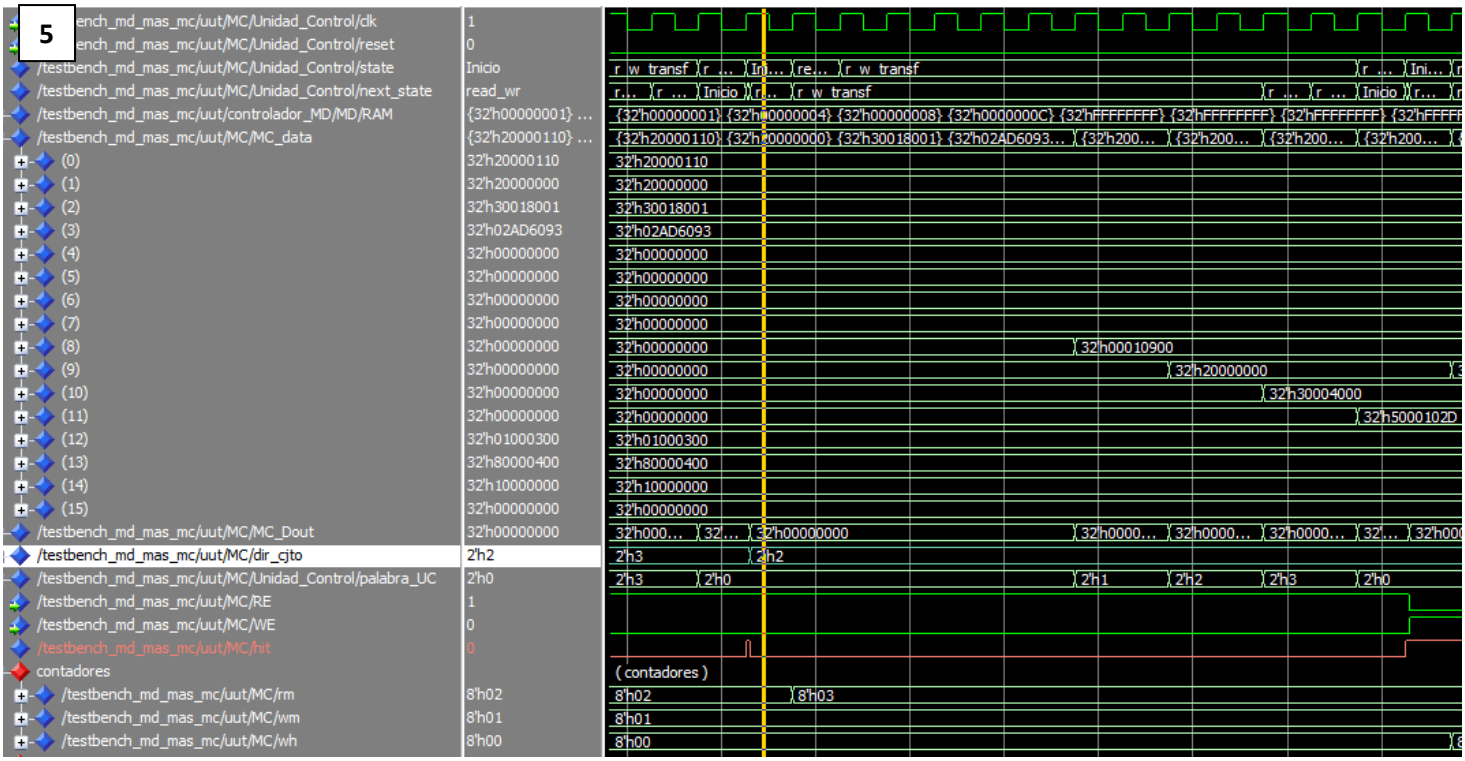
Read miss	3
Write miss	1
Write hit	2



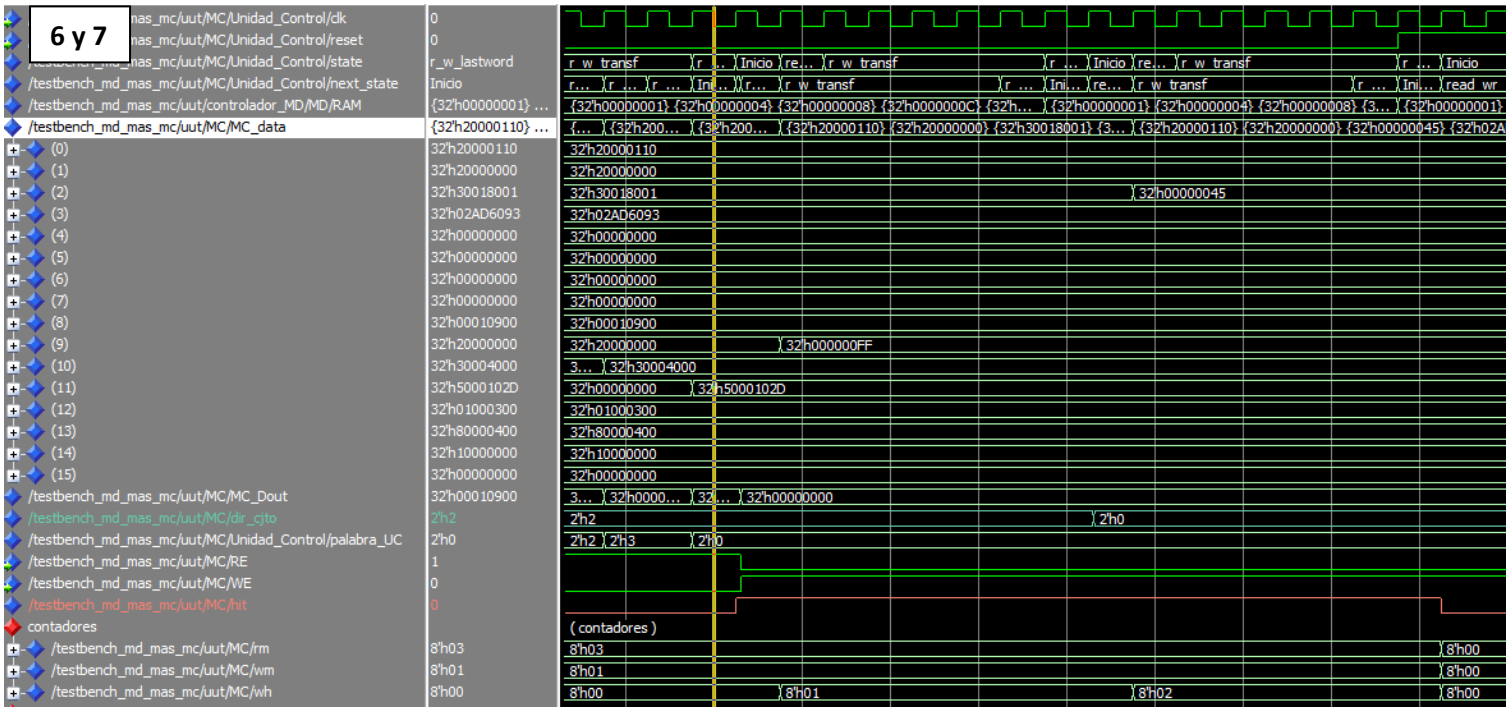

2 y 3
_md_mas_mc/uut/MC/Unidad_Control/clk
_md_mas_mc/uut/MC/Unidad_Control/reset
/testbench_md_mas_mc/uut/MC/Unidad_Control/state
/testbench_md_mas_mc/uut/MC/Unidad_Control/next_state
/testbench_md_mas_mc/uut/controlador_MD/MD/RAM
/testbench_md_mas_mc/uut/MC/MC_data
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
/testbench_md_mas_mc/uut/MC/MC_Dout
/testbench_md_mas_mc/uut/MC/dir_cjto
/testbench_md_mas_mc/uut/MC/Unidad_Control/palabra_UC
/testbench_md_mas_mc/uut/MC/RE
/testbench_md_mas_mc/uut/MC/WE
/testbench_md_mas_mc/uut/MC/hit
contadores
/testbench_md_mas_mc/uut/MC/rm
/testbench_md_mas_mc/uut/MC/wm
/testbench_md_mas_mc/uut/MC/wh
1
0
Inicio
{32'h00000001} ...
{32'h20000110} ...
32'h20000110
32'h20000000
32'h30018001
32'h02AD6093
32'h00000000
2'h0
2'h3
8'h01
8'h00
(contadores)


4
/testbench_md_mas_mc/uut/MC/Unidad_Control/clk
/testbench_md_mas_mc/uut/MC/Unidad_Control/reset
/testbench_md_mas_mc/uut/MC/Unidad_Control/state
/testbench_md_mas_mc/uut/MC/Unidad_Control/next_state
/testbench_md_mas_mc/uut/controlador_MD/MD/RAM
/testbench_md_mas_mc/uut/MC/MC_data
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
/testbench_md_mas_mc/uut/MC/MC_Dout
/testbench_md_mas_mc/uut/MC/dir_cjto
/testbench_md_mas_mc/uut/MC/Unidad_Control/palabra_UC
/testbench_md_mas_mc/uut/MC/RE
/testbench_md_mas_mc/uut/MC/WE
/testbench_md_mas_mc/uut/MC/hit
contadores
/testbench_md_mas_mc/uut/MC/rm
/testbench_md_mas_mc/uut/MC/wm
/testbench_md_mas_mc/uut/MC/wh
0
Inicio
read_wr
{32'h00000001} ...
{32'h20000110} ...
32'h20000110
32'h20000000
32'h30018001
32'h02AD6093
32'h00000000
32'h01000300
32'h80000400
32'h10000000
2'h0
2'h3
2'h1
2'h2
8'h01
8'h02
8'h00
(contadores)

5


/testbench_md_mas_mc/uut/MC/Unidad_Control/clk
/testbench_md_mas_mc/uut/MC/Unidad_Control/reset
/testbench_md_mas_mc/uut/MC/Unidad_Control/state
/testbench_md_mas_mc/uut/MC/Unidad_Control/next_state
/testbench_md_mas_mc/uut/controlador_MD/MD/RAM
/testbench_md_mas_mc/uut/MC/MC_data
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
/testbench_md_mas_mc/uut/MC/MC_Dout
/testbench_md_mas_mc/uut/MC/dir_cjto
/testbench_md_mas_mc/uut/MC/Unidad_Control/palabra_UC
/testbench_md_mas_mc/uut/MC/RE
/testbench_md_mas_mc/uut/MC/WE
/testbench_md_mas_mc/uut/MC/hit
contadores
/testbench_md_mas_mc/uut/MC/rm
/testbench_md_mas_mc/uut/MC/wm
/testbench_md_mas_mc/uut/MC/wh
1
0
Inicio
read_wr
{32'h00000001} ...
{32'h20000110} ...
32'h20000110
32'h20000000
32'h30018001
32'h02AD6093
32'h00000000
32'h00010900
32'h20000000
32'h30004000
32'h5000102D
32'h01000300
32'h80000400
32'h10000000
32'h00000000
2'h2
2'h0
8'h02
8'h03
8'h01
8'h00
(contadores)

6 y 7


/testbench_md_mas_mc/uut/MC/Unidad_Control/clk
/testbench_md_mas_mc/uut/MC/Unidad_Control/reset
/testbench_md_mas_mc/uut/MC/Unidad_Control/state
/testbench_md_mas_mc/uut/MC/Unidad_Control/next_state
/testbench_md_mas_mc/uut/controlador_MD/MD/RAM
/testbench_md_mas_mc/uut/MC/MC_data
(0)
(1)
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)
(10)
(11)
(12)
(13)
(14)
(15)
/testbench_md_mas_mc/uut/MC/MC_Dout
/testbench_md_mas_mc/uut/MC/dir_cjto
/testbench_md_mas_mc/uut/MC/Unidad_Control/palabra_UC
/testbench_md_mas_mc/uut/MC/RE
/testbench_md_mas_mc/uut/MC/WE
/testbench_md_mas_mc/uut/MC/hit
contadores
/testbench_md_mas_mc/uut/MC/rm
/testbench_md_mas_mc/uut/MC/wm
/testbench_md_mas_mc/uut/MC/wh
0
r_w_lastword
Inicio
read_wr
{32'h00000001} ...
{32'h20000110} ...
32'h20000110
32'h20000000
32'h30018001
32'h02AD6093
32'h00000000
32'h00010900
32'h30004000
32'h5000102D
32'h01000300
32'h80000400
32'h10000000
32'h00000045
32'h000000FF
32'h00000000
2'h2
2'h3
2'h0
1
8'h03
8'h01
8'h00
8'h02
(contadores)

Por último, para confirmar que los contadores del procesador MIPS detectan cada parada y referencia a memoria, se ha implementado un pequeño programa finito en el que aparecen todos los casos posibles que se pueden llegar a dar (riesgos de datos, riesgos de control, etc).

El programa se encuentra en el fichero “memoriaRAM_I_Test_contadores”.

```
1                LW R0,0(R0)
2                LA R1,4(R0)
3                BEQ R1,R2,BUC1
4          BUC1  SW R0,8(R2)
5                ADD R2,R2,R1
6                ADD R1,R3,R3
7                BEQ R1,R1,BUC2
8          BUC2  NOP
```

En la línea 1 detecta un mem_read y las paradas necesarias de la memoria (que son 13).

Riesgo de datos entre la línea 1 y 2 con el registro R0.

Riesgos de datos entre la línea 2 y 3 con el registro R1 y riesgo de control al predecir mal el salto (no tomado).

En la línea 4 aumenta el contador de mem_write y el de las paradas de memoria (que son 7).

Riesgos de datos entre las líneas 5, 6 y 7 con el registro R1 y otro riesgo de control debido a una predicción errónea (salto tomado).

Quedando los contadores de la siguiente manera:

Paradas de control	2
Paradas de datos	3
Paradas de memoria	20
Mem reads	1
Mem writes	1

5. Conclusión y gestión del proyecto

Los dos proyectos que se han tenido que realizar durante este curso, han ayudado de manera importante a comprender de una manera mucho más detallada el funcionamiento del procesador estudiado y el uso de la memoria cache para disminuir los tiempos de las instrucciones de acceso a memoria. Al poder verse los efectos plasmados en una pantalla, ha ayudado bastante a interpretar cómo se distribuyen los datos.

Una calificación acorde con el esfuerzo que ha supuesto el desarrollo y comprensión de la asignatura, tanto cuestiones prácticas como teóricas, podría calificarse con un notable.

Estudio de los fuentes	4h
Diseño inicial de la unidad de control	3h
Depuración y ajustes	15h
Memoria	4h
TOTAL	**= 26h**

ANEXO I: Documentación aportada por los profesores

En la P2 hay que trabajar con el procesador de la P1 pero sustituyendo el componente de memoria de datos por el la nueva jerarquía de memoria con MC y MD conectados por un bus.

Para ello hay que quitar el componente

```
component memoriaRAM_D is port (
                CLK : in std_logic;
                ADDR : in std_logic_vector (31 downto 0); --Dir
    Din : in std_logic_vector (31 downto 0);--entrada de datos para el puerto de escritura
    WE : in std_logic;          -- write enable
                RE : in std_logic;              -- read enable
                Dout : out std_logic_vector (31 downto 0));
end component;
```

y sustituirlo por

```
component MD_mas_MC is port (
                CLK : in std_logic;
                reset: in std_logic;
                ADDR : in std_logic_vector (31 downto 0); --Dir solicitada por el Mips
    Din : in std_logic_vector (31 downto 0);--entrada de datos desde el Mips
    WE : in std_logic;          -- write enable  del MIPS
                RE : in std_logic;              -- read enable del MIPS
                Mem_ready: out std_logic; -- indica si podemos hacer la operación solicitada
en el ciclo actual
                Dout : out std_logic_vector (31 downto 0) --dato que se envía al Mips
                ); --salida que puede leer el MIPS
end component;
```

Lo mismo al instanciarlo. Hay que sustituir:

```
Mem_D: memoriaRAM_D PORT MAP (CLK => CLK, ADDR => ALU_out_MEM, Din =>
BusB_MEM, WE => MemWrite_MEM, RE => MemRead_MEM, Dout => Mem_out);
```

por

Mem_D: MD_mas_MC PORT MAP (CLK => CLK, reset => reset, ADDR => ALU_out_MEM, Din => BusB_MEM, WE => MemWrite_MEM, RE => MemRead_MEM, Mem_ready => Mem_ready, Dout => Mem_out);

Tambien hay que declarar la señal Mem_ready

signal Mem_ready : std_logic;

Después debéis usar la señal Mem_ready para parar el procesador cuando corresponda.

A parte hay que añadir los contadores.

Para ello hay que incluir el componente counter:

```
component counter is
   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          count_enable : in  STD_LOGIC;
          load : in  STD_LOGIC;
          D_in  : in  STD_LOGIC_VECTOR (7 downto 0);
                   count : out  STD_LOGIC_VECTOR (7 downto 0));
end component;
```

e instanciar un contador para cada métrica que queramos medir:

cont_ciclos: counter port map (clk => clk, reset => reset, count_enable => '1' , load=> '0', D_in => "00000000", count => ciclos);

cont_paradas_control: counter port map (clk => clk, reset => reset, count_enable => inc_paradas_control , load=> '0', D_in => "00000000", count => paradas_control);

cont_paradas_datos: counter port map (clk => clk, reset => reset, count_enable => inc_paradas_datos , load=> '0', D_in => "00000000", count => paradas_datos);

cont_paradas_memoria: counter port map (clk => clk, reset => reset, count_enable => inc_paradas_memoria , load=> '0', D_in => "00000000", count => paradas_memoria);

cont_mem_reads: counter port map (clk => clk, reset => reset, count_enable => inc_mem_reads , load=> '0', D_in => "00000000", count => mem_reads);

cont_mem_writes: counter port map (clk => clk, reset => reset, count_enable => inc_mem_writes , load=> '0', D_in => "00000000", count => mem_writes);

Además hay que declarar las señales:

signal ciclos, paradas_control, paradas_datos, paradas_memoria, mem_reads, mem_writes: std_logic_vector(7 downto 0);

signal inc_paradas_control, inc_paradas_datos, inc_paradas_memoria, inc_mem_reads, inc_mem_writes : std_logic;