Escuela de Ingeniería y Arquitectura Universidad Zaragoza

JERARQUÍA DE MEMORIA DE DATOS

AUTORES:
ALEJANDRO TERRÓN ÁLVAREZ, NIP:761069
DIEGO MARCO BEISTY, NIP:755232

ÍNDICE

RESUMEN

El objetivo de este proyecto consiste en incorporar una memoria caché (MC) al procesador MIPS trabajado en el proyecto anterior. Para ello se ha cambiado el antiguo componente de memoria de datos (MD) por la nueva jerarquía de memoria con MC y MD conectados por un bus. Posteriormente se ha diseñado el controlador de la MC para poder gestionar las peticiones del procesador, haciendo las transferencias a la memoria principal que sean necesarias.

Se ha gestionado con la unidad de detención un nuevo riesgo de datos producido cuando la memoria tarda más de un ciclo en gestionar una petición y finalmente se han incorporado 9 contadores, tanto en el MIPS como en la MC para extraer datos de interés.

Paso 1: Trabajo inicial

A partir del mips trabajado en el proyecto anterior, hemos cambiado el componente de la MD por la nueva jerarquía de memoria con MC y MD conectados por un bus.

Fichero MIPs_proyecto2_2019.vhd :

1. Sustituimos el componente anterior por el nuevo.

```
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;
```

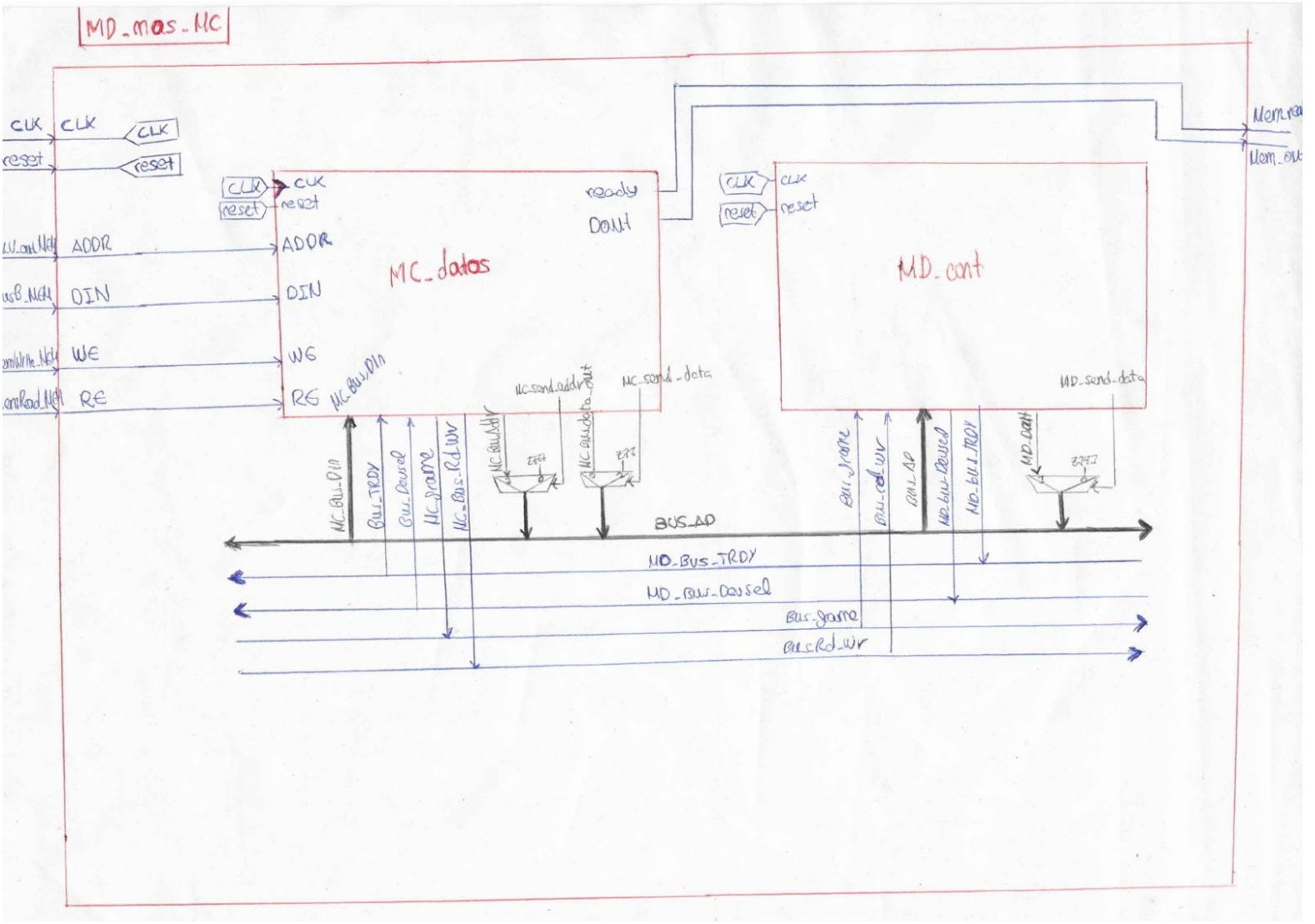
2.Sustituimos la instancia anterior por la nueva.

```
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);
```

3.Declaramos la nueva señal Mem_ready. Esta señal vale 1 cuando la memoria caché puede atender la petición del procesador en un ciclo de reloj, en caso contrario vale 0.

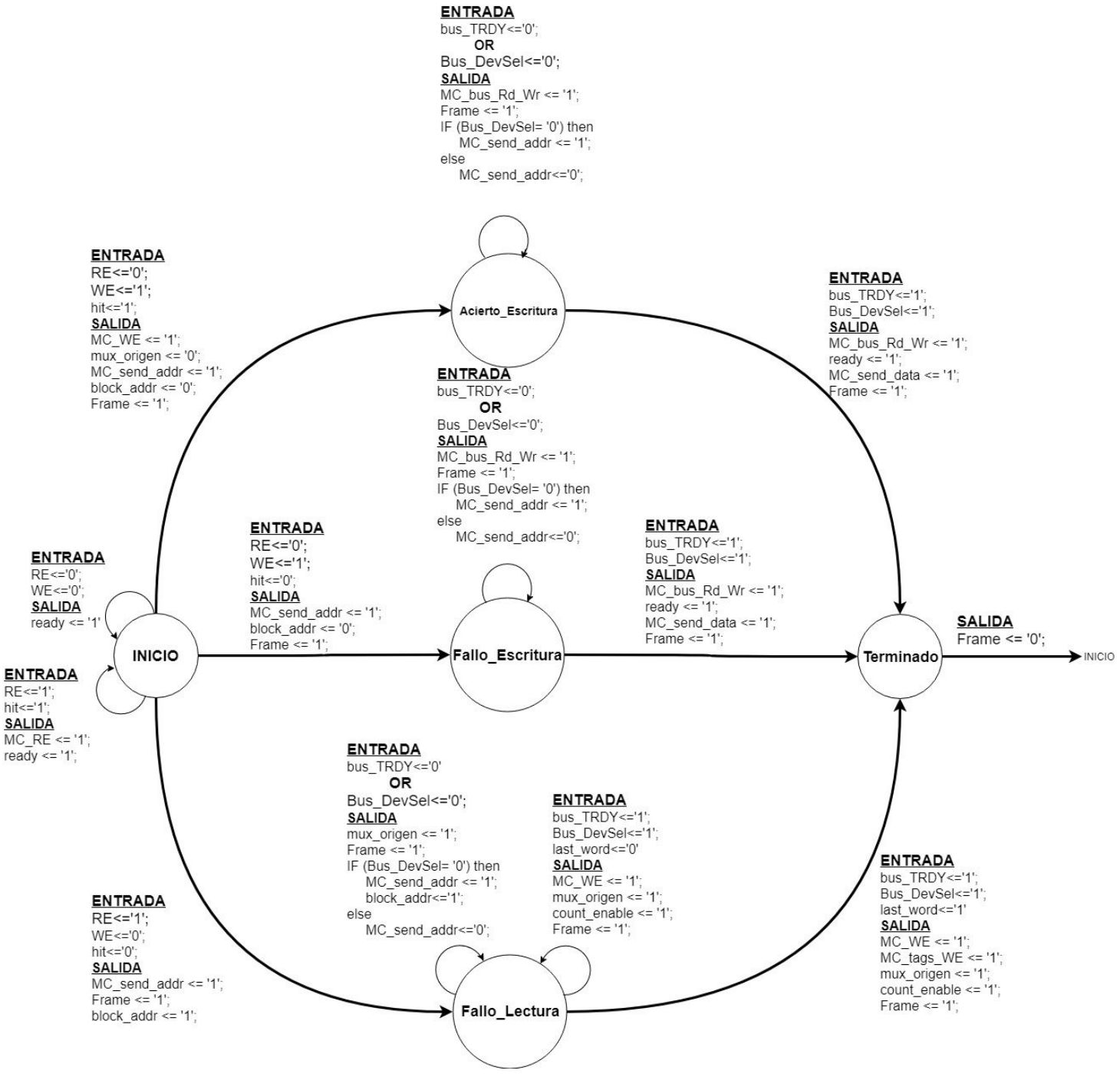
```
signal Mem_ready : std_logic;
```

Después hemos hecho un esquema de las conexiones internas del nuevo componente MD_mas_MC para visualizarlo mejor:



Paso 2: Autómata de control

Nuestro autómata de control es un autómata de Mealy de 5 estados el cual presentamos en la siguiente imagen, hemos obviado las señales de salida con valor 0 que no son importantes en la transición.



Estado	Descripción
INICIO	Estado de inicio.
Acierto_Escritura	Estado de acierto de escritura en el que se aplica Write-through, (escritura en MC y en MD).
Fallo_Escritura	Estado de fallo en la escritura. Se aplica la política Write_around, (escritura en MD).
Fallo_Lectura	Estado de fallo en la lectura. Se desplaza un bloque de MP a MC.
Terminado	Estado que pone todas la señales a 0 para finalizar la transmisión entre la MC y la MD.

Estado **INICIO:**

1ª transición

```
if (state = Inicio and RE= '0' and WE= '0') then -- si no piden nada no hacemos nada
        next_state <= Inicio;
        ready <= '1';
```

Caso en que la instrucción que se encuentra en fase MEM no es una instrucción LW o SW, (ni de lectura, ni de escritura). Por lo tanto no hace uso de memoria y se activa la señal ready a 1 para indicar que ha terminado de procesar la petición del mips.

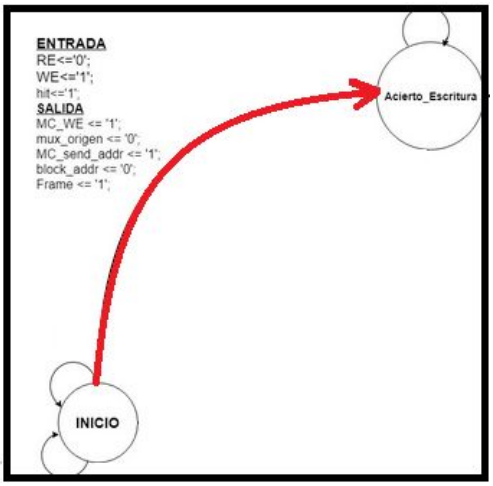
2ª transición

```
elsif(state = Inicio and RE= '1' and  hit = '1') then -- si acierto lectura -> Inicio
        next_state <= Inicio;
            MC_WE <= '0';
            MC_bus_Rd_Wr <= '0';
            MC_tags_WE <= '0';
            MC_RE <= '1';
            ready <= '1';
            mux_origen <= '0';
            MC_send_addr <= '0';
            MC_send_data <= '0';
            count_enable <= '0';
            Frame <= '0';
            Replace_block <= '0';
            block_addr <= '0';
            inc_rm <= '0';
            inc_wm <= '0';
            inc_wh <= '0';
```

Caso en que la instrucción que se encuentra en la fase MEM es un LW (instrucción de lectura) y además el tag de la dirección pedida coincide con un tag almacenado en MC_tags, es decir, que el bloque de la palabra que lee se encuentra disponible en la MC . Por lo tanto se activa la señal MC_RE a 1, para poder leer la palabra de la memoria caché y la señal ready a 1, para indicar que ha finalizado la petición del mips.

3ª Transición

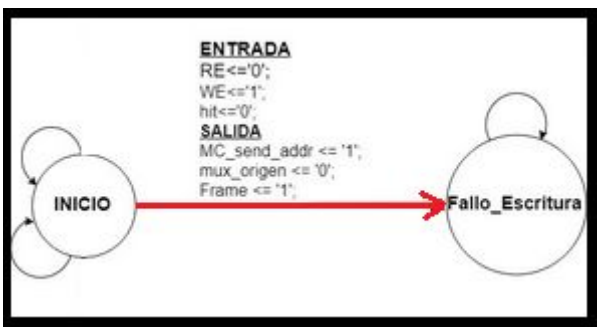
```
elsif(state = Inicio and RE= '0' and  WE= '1' and hit = '1') then -- si acierto escritura -> Acierto_Escritura
        next_state <= Acierto_Escritura;
            MC_WE <= '1';
            MC_bus_Rd_Wr <= '0';
            MC_tags_WE <= '0';
            MC_RE <= '0';
            ready <= '0';
            mux_origen <= '0';
            MC_send_addr <= '1';
            MC_send_data <= '0';
            count_enable <= '0';
            Frame <= '1';
            Replace_block <= '0';
            block_addr <= '0';
            inc_rm <= '0';
            inc_wm <= '0';
            inc_wh <= '1';
```



Caso en que la instrucción que se encuentra en la fase MEM es un SW y además el tag de la dirección pedida coincide con un tag almacenado en MC_tags, es decir, que el bloque de la palabra donde escribe se encuentra disponible en la MC. Por lo tanto activa la señal MC_WE a 1 para escribir en la MC. Además inicia una nueva transmisión con la MD activando frame a 1 , MC_send_addr a 1 y Block_addr a 0 para mandarle la dirección de memoria de la palabra donde quiere escribir en la MD. Se activa inc_wh para contar el acierto de escritura.

4ª transición

```
elsif(state = Inicio and WE='1' and hit='0') then -- si fallo escritura -> Fallo_Escritura
            next_state <= Fallo_Escritura;
            MC_WE <= '0';
            MC_bus_Rd_Wr <= '0';
            MC_tags_WE <= '0';
            MC_RE <= '0';
            ready <= '0';
            mux_origen <= '0';
            MC_send_addr <= '1';
            MC_send_data <= '0';
            count_enable <= '0';
            Frame <= '1';
            Replace_block <= '0';
            block_addr <= '0';
            inc_rm <= '0';
            inc_wm <= '1';
            inc_wh <= '0';
```

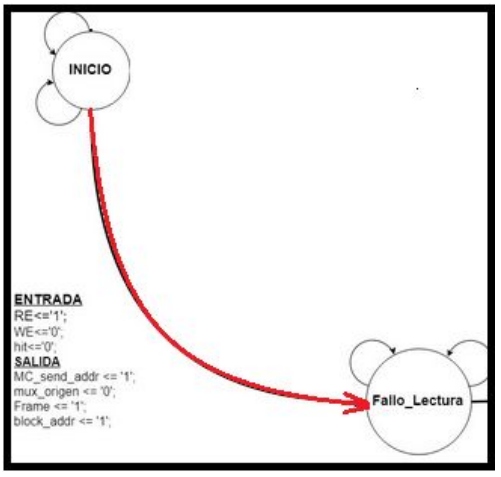


Caso en que la instrucción que se encuentra en la etapa MEM es un SW y la dirección donde quiere escribir no se encuentra en la MC. Por lo tanto inicia una nueva transmisión con la MD activando frame a 1, MC_send_addr a 1 y block_addr a 0 para mandarle la dirección de memoria de la palabra donde quiere escribir en la MD.

Además inc_wm se activa para contabilizar el fallo de escritura.

5ª transición

```
elsif(state = Inicio and RE='1' and hit='0') then -- si fallo lectura -> Fallo_Lectura
        next_state <= Fallo_Lectura;
        MC_WE <= '0';
        MC_bus_Rd_Wr <= '0';
        MC_tags_WE <= '0';
        MC_RE <= '0';
        ready <= '0';
        mux_origen <= '0';
        MC_send_addr <= '1';
        MC_send_data <= '0';
        count_enable <= '0';
        Frame <= '1';
        Replace_block <= '0';
        block_addr <= '1';
        inc_rm <= '1';
        inc_wm <= '0';
        inc_wh <= '0';
```



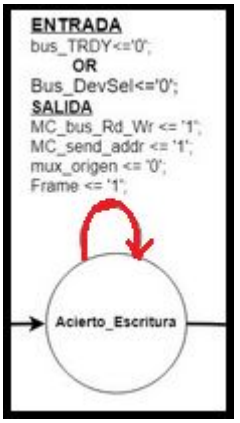
Caso en que la instrucción que se encuentra en la etapa MEM es un LW y la dirección donde que quiere leer no se encuentra en la MC. Por lo tanto inicia una nueva transmisión con MD activando frame a 1, MC_send_addr a 1, y block addr a 1 para mandarle la dirección del bloque que quiere cargar en la MC.

Además activa la señal inc_rm para indicar fallo de lectura.

Estado **Acierto_Escritura:**

6ª transición

```
elsif(state = Acierto_Escritura and (Bus_DevSel= '0' or bus_TRDY= '0')) then -- si MD no ha reconocido dirección de MC ó
        next_state <= Acierto_Escritura;                                     -- MD no preparada -> Acierto_Escritura
        MC_WE <= '0';
        MC_bus_Rd_Wr <= '1';
        MC_tags_WE <= '0';
        MC_RE <= '0';
        ready <= '0';
        mux_origen <= '0';
        IF (Bus_DevSel= '0') then  -- si MD no ha reconocido dirección de MC
            MC_send_addr <= '1';
        else
            MC_send_addr<='0';
        end if;
        MC_send_data <= '0';
        count_enable <= '0';
        Frame <= '1';
        Replace_block <= '0';
        block_addr <= '0';
        inc_rm <= '0';
        inc_wm <= '0';
        inc_wh <= '0';
```

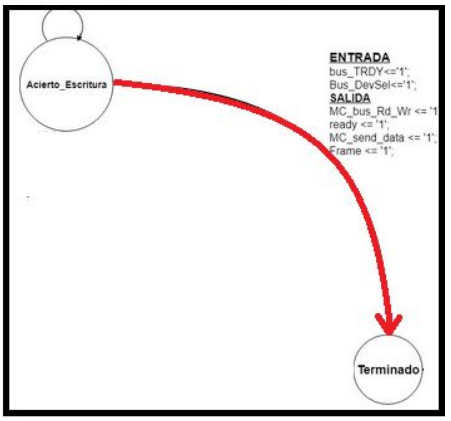


- A esta transición se entra por que la MD no ha reconocido la dirección que le ha enviado la MC (Bus_DevSel = 0) por lo tanto sigue mandando la dirección (MC_send_addr permanece a 1), mantenemos la transmisión (frame a 1), mandamos el código de operación (MC_bus_Rd_Wr a 1).

-También se entra porque la MD ha reconocido la dirección pero todavía no está preparada, (bajamos MC_send_addr a 0 y las demás señales se quedan igual).

7ª transición:

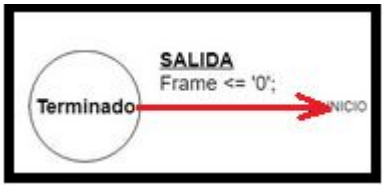
```
elsif(state = Acierto_Escritura and Bus_DevSel= '1' and bus_TRDY= '1' ) then -- si MD preparada en acierto escritura -> Terminado
        next_state <= Terminado;
        MC_WE <= '0';
        MC_bus_Rd_Wr <= '1';
        MC_tags_WE <= '0';
        MC_RE <= '0';
        ready <= '1';
        mux_origen <= '0';
        MC_send_addr <= '0';
        MC_send_data <= '1';
        count_enable <= '0';
        Frame <= '1';
        Replace_block <= '0';
        block_addr <= '0';
        inc_rm <= '0';
        inc_wm <= '0';
        inc_wh <= '0';
```



Caso la MD está preparada para procesar la petición de escritura de la MC. Se mantiene el código de operación (MC_bus_Rd_Wr a 1), activamos el permiso MC_send_data para mandar el dato que queremos escribir en la MD, mantenemos frame a 1 para continuar la transmisión. Ponemos ready a 1 para indicar al mips que ya puede pasar la siguiente instrucción a MEM. (No se atenderá hasta que la UC_MC vuelva al estado INICIO).

8ª transición:

```
elsif(state = Terminado) then -- si terminado -> Inicio
        next_state <= Inicio;
end if;
```

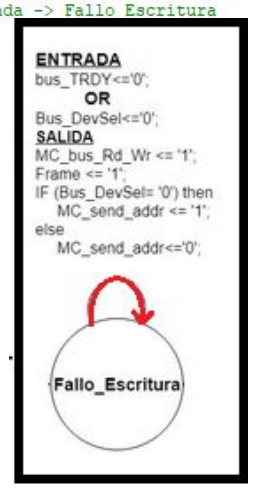


Caso en que ha terminado la escritura de la palabra en MD por el acierto de escritura o fallo de escritura, o ha terminado la lectura de la última palabra de la MD por el fallo de lectura. En este estado se ponen todas las señales a 0, (frame se pone a 0), para cerrar la transmisión con la MD y se vuelve al estado INICIO.

Estado **Fallo_Escritura:**

9ª transición:

```
elsif(state = Fallo_Escritura and (Bus_DevSel= '0' or bus_TRDY= '0')) then -- si MD no ha reconocido dirección de MC (
        next_state <= Fallo_Escritura;                                     -- MD no preparada -> Fallo_Escritura
        MC_WE <= '0';
        MC_bus_Rd_Wr <= '1';
        MC_tags_WE <= '0';
        MC_RE <= '0';
        ready <= '0';
        mux_origen <= '0';
        IF (Bus_DevSel= '0') then -- si MD no ha reconocido dirección de MC
            MC_send_addr <= '1';
        else
            MC_send_addr<='0';
        end if;
        MC_send_data <= '0';
        count_enable <= '0';
        Frame <= '1';
        Replace_block <= '0';
        block_addr <= '0';
        inc_rm <= '0';
        inc_wm <= '0';
        inc_wh <= '0';
```

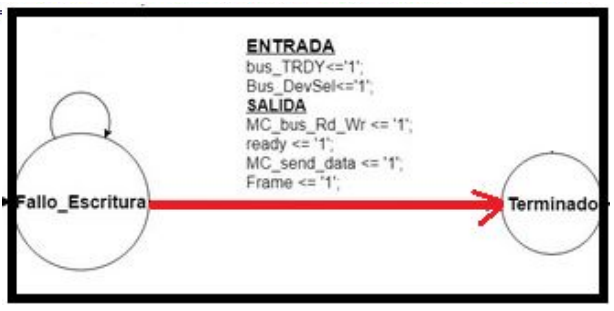


- A esta transición se entra porque la MD no ha reconocido la dirección que le ha enviado la MC (Bus_DevSel = 0) por lo tanto sigue mandando la dirección (MC_send_addr permanece a 1), mantenemos la transmisión (frame a 1), mandamos el código de operación (MC_bus_Rd_Wr a 1).

-También se entra porque la MD ha reconocido la dirección pero todavía no está preparada, (bajamos MC_send_addr a 0 y las demás señales se quedan igual).

10ª transición:

```
elsif(state = Fallo_Escritura and Bus_DevSel= '1' and bus_TRDY=
        next_state <= Terminado;
        MC_WE <= '0';
        MC_bus_Rd_Wr <= '1';
        MC_tags_WE <= '0';
        MC_RE <= '0';
        ready <= '1';
        mux_origen <= '0';
        MC_send_addr <= '0';
        MC_send_data <= '1';
        count_enable <= '0';
        Frame <= '1';
        Replace_block <= '0';
        block_addr <= '0';
        inc_rm <= '0';
        inc_wm <= '0';
        inc_wh <= '0';
```

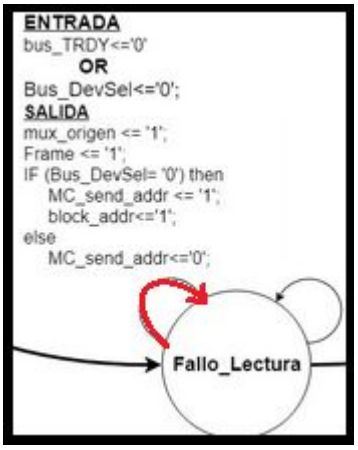


Caso la MD está preparada para procesar la petición de escritura de la MC. Se mantiene el código de operación (MC_bus_Rd_Wr a 1), activamos el permiso MC_send_data para mandar el dato que queremos escribir en la MD, mantenemos frame a 1 para continuar la transmisión. Ponemos ready a 1 para indicar al mips que ya puede pasar la siguiente instrucción a MEM. (No se atenderá hasta que la UC_MC vuelva al estado INICIO).

Estado **Fallo_Lectura:**

11ª transición:

```
elsif(state = Fallo_Lectura and (Bus_DevSel= '0' or bus_TRDY= '0')) then -- si MD no ha reconocido dirección de MC ó
            next_state <= Fallo_Lectura;                                  -- MD no preparada.
            MC_WE <= '0';
            MC_bus_Rd_Wr <= '0';
            MC_tags_WE <= '0';
            MC_RE <= '0';
            ready <= '0';
            mux_origen <= '1';
            IF (Bus_DevSel= '0') then  --si MD no ha reconocido dirección de MC
                MC_send_addr <= '1';
                block_addr <= '1';
            else
                MC_send_addr<='0';
            end if;
            MC_send_data <= '0';
            count_enable <= '0';
            Frame <= '1';
            Replace_block <= '0';
            block_addr <= '0';
            inc_rm <= '0';
            inc_wm <= '0';
            inc_wh <= '0';
```

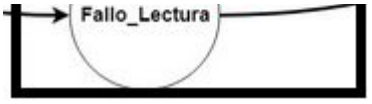


- A esta transición se entra porque la MD no ha reconocido la dirección que le ha enviado la MC (Bus_DevSel = 0) por lo tanto sigue mandando la dirección (MC_send_addr permanece a 1), block_addr permanece a 1, mantenemos la transmisión (frame a 1), mandamos el código de operación (MC_bus_Rd_Wr a 0).

-También se entra porque la MD ha reconocido la dirección pero todavía no está preparada, (bajamos MC_send_addr a 0, block_addr a 0, y las demás señales se quedan igual).

12ª transición:

```
elsif(state = Fallo_Lectura and Bus_DevSel= '1' and bus_TRDY= '1' and last_word= '0') then -- si MD preparada en fallo lectura y
            next_state <= Fallo_Lectura;                                                    -- NO última palabra -> Fallo_Lectura
            MC_WE <= '1';
            MC_bus_Rd_Wr <= '0';
            MC_tags_WE <= '0';
            MC_RE <= '0';
            ready <= '0';
            mux_origen <= '1';
            MC_send_addr <= '0';
            MC_send_data <= '0';
            count_enable <= '1';
            Frame <= '1';
            Replace_block <= '0';
            block_addr <= '0';
            inc_rm <= '0';
            inc_wm <= '0';
            inc_wh <= '0';
```

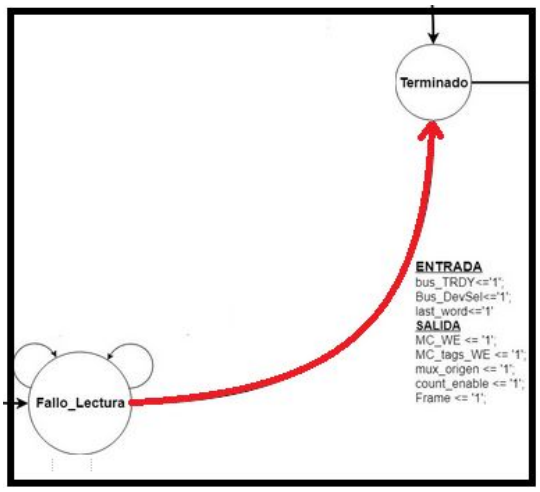


Caso MD preparada para leer una palabra y mandarla a la MC.

Se pone el permiso de escritura de la MC a 1 (MC_WE), se pone mux_origen a 1 para recibir los datos de la MD y no de la entrada de la MC. Además se activa el contador de palabras para incrementarlo en 1. Frame permanece a 1 para continuar con la transmisión.

13ª transición:

```
elsif(state = Fallo_Lectura and Bus_DevSel= '1' and bus_TRDY= '1' and last_word= '1') then -- si MD preparada en fallo lectura y
            next_state <= Terminado;                                                       -- última palabra -> Terminado
            MC_WE <= '1';
            MC_bus_Rd_Wr <= '0';
            MC_tags_WE <= '1';
            MC_RE <= '0';
            ready <= '0';
            mux_origen <= '1';
            MC_send_addr <= '0';
            MC_send_data <= '0';
            count_enable <= '1';
            Frame <= '1';
            Replace_block <= '0';
            block_addr <= '0';
            inc_rm <= '0';
            inc_wm <= '0';
            inc_wh <= '0';
```



Caso MD preparada para leer la última palabra y mandarla a la MC.

Se pone el permiso de escritura de la MC a 1 (MC_WE), se activa la señal MC_tags_WE a 1 para actualizar el tag del bloque en MC_tags. Mux_origen se pone a 1 para recibir el dato de la MD y no de la entrada de la MC. Mantenemos count_enable a 1 para contar la última palabra, además del frame para no cortar la transmisión.

Paso 3: Modificaciones Unidad de Detención

Fichero **UD_Mips.vhd**

```
entity UD is
    Port ( Reg_Rs_ID: in STD_LOGIC_VECTOR (4 downto 0);--registro rs de etapa ID
           Reg_Rt_ID: in STD_LOGIC_VECTOR (4 downto 0);--registro rt de etapa ID
           RW_MEM: in  STD_LOGIC_VECTOR (4 downto 0);--registro destino etapa MEM
           RW_EX: in  STD_LOGIC_VECTOR (4 downto 0);--registro destino etapa EX
           MemRead_EX: in STD_LOGIC;--Señal control lectura memoria etapa EX
           MemRead_MEM: in STD_LOGIC;--Señal control lectura memoria etapa MEM
           MemWrite_MEM: in STD_LOGIC;--Señal control escritura memoria etapa MEM
           RegWrite_MEM: in STD_LOGIC;--Señal control escritura memoria etapa MEM
           RegWrite_EX: in STD_LOGIC;--Señal control escritura memoria etapa EX
           Op_code_ID: in  STD_LOGIC_VECTOR (5 downto 0);--Código operación etapa ID
           Mem_ready: in std_logic; -- indica si podemos hacer la operación solicitada en el ciclo actual
           avanzar_ID: out STD_LOGIC;
           avanzar_cache: out STD_LOGIC); -- Detiene el mips si la caché no puede hacer la operación solicitada en el ciclo actual);
end UD;
```

La unidad de detención (**UD**) se encarga de parar la ejecución de la instrucción que se encuentre en la fase de decodificación así como de parar la actualización del PC para evitar la entrada de nuevas instrucciones. Esta unidad permite evitar los riesgos de datos explicados en el primer proyecto. Al cambiar el módulo de memoria MD por uno nuevo que integra una MC y una MD unidas por un bus, aparece un nuevo riesgo de datos:

➢ Caso en que entra una instrucción de lectura o escritura en etapa MEM, y la nueva memoria (MD_mas_MC) no puede resolver la petición en un ciclo de reloj.

```
avanzar_cache <= '0' when Mem_ready='0' AND (MemRead_MEM='1' OR MemWrite_MEM='1') else
                 '1';
```

La unidad de detención para los cuatro bancos de registros, (IF/ID, ID/EX, EX/MEM, MEM/WB) y el PC. Los tres primeros y el PC para que no se pierdan instrucciones en etapas anteriores. El banco MEM/WB lo paramos por si hay una anticipación de la instrucción en etapa EX con la instrucción en etapa WB.

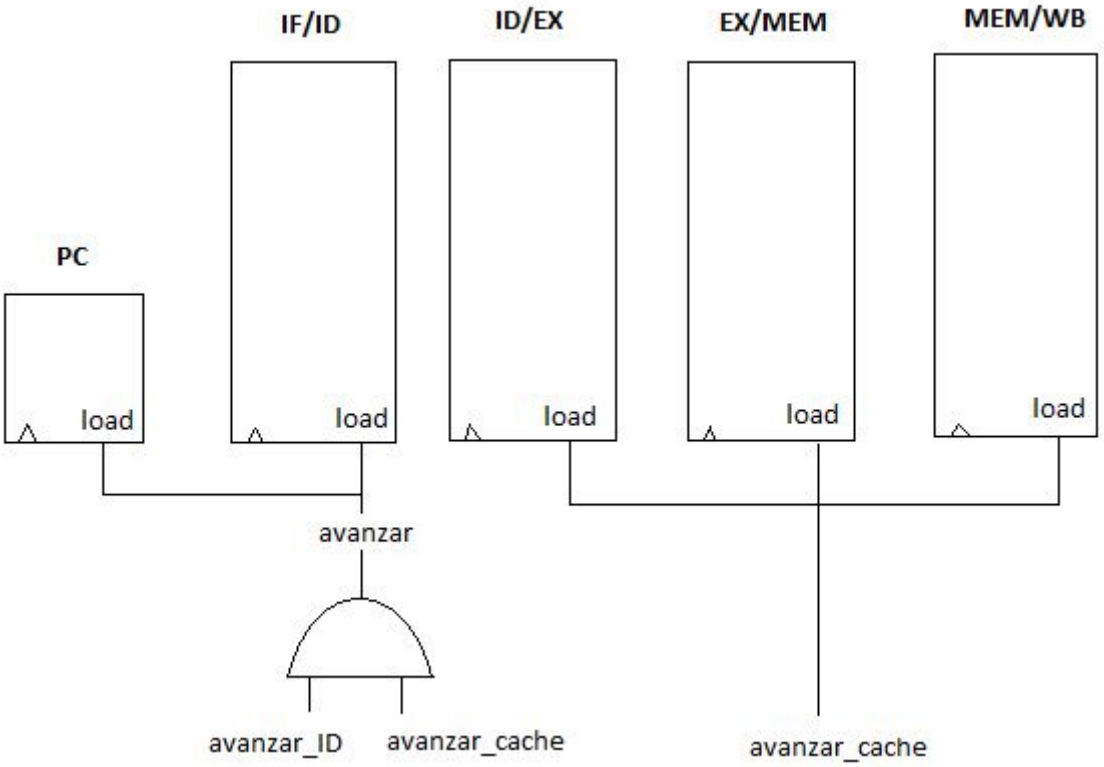
Para poder parar los cinco bancos de registros y el PC, hemos añadido las siguientes señales a al UD.

Mem_ready: señal de entrada de la UD, conectada con la señal externa del mips Mem_ready Indica a la unidad de control si la memoria puede hacer la operación solicitada en un ciclo de reloj.

avanzar_cache: señal de salida de la UD, conectada con la señal externa del mips avanzar_cache. Detiene el mips si Mem_ready cuando la memoria no puede hacer la operación actual en el ciclo actual.

Además de las señales MemRead_Mem y MemWrite_Mem para identificar que la instrucción en etapa MEM es de lectura o de escritura, (LW o SW).

Este esquema muestra la forma en la que hemos conectado las señales a los bancos y PC para pararlos con el nuevo riesgo de datos:



Paso 4: Contadores

Se han implementado 9 contadores con la finalidad de monitorizar la actividad del procesador y probar su correcto funcionamiento. Para los implementados en el fichero MIPs_proyecto2_2019.vhd se han creado una serie de señales con el fin de controlar el funcionamiento de los mismos.

```
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);
```

```
inc_paradas_control <= '1' when (predictor_error = '1' and avanzar_cache = '1' and avanzar_ID = '1') else '0';
inc_paradas_datos <= '1' when (avanzar_ID = '0' and avanzar_cache = '1') else '0';
inc_paradas_memoria <=  '1' when (avanzar_cache = '0') else '0';
inc_mem_reads <= '1' when (Memread_MEM = '1' and Mem_ready = '1') else '0';
inc_mem_writes <= '1' when (MemWrite_MEM = '1' and Mem_ready = '1') else '0';
```

➤ inc_paradas_control: señal encargada de activar el contador cuando exista algún tipo de error por parte del predictor de saltos y a su vez la pipeline no se encuentre paralizada en ninguno de sus tramos.

➤ inc_paradas_datos: señal encargada de activar el contador cuando se produce algún tipo de riesgo de datos el cual no se puede solucionar mediante el anticipador y por lo tanto hay que paralizar el banco IF/ID y PC.

➤ inc_paradas_memoria: señal encargada de activar el contador cuando alguna operación que precisa de intervención de la memoria no se puede resolver en un ciclo.

➤ inc_mem_reads: señal encarga de de activar el contador que lleva la cuenta de las operaciones que leen de memoria, es decir, de los LW.

➤ inc_mem_writes: señal encarga de de activar el contador que lleva la cuenta de las operaciones que escriben en memoria, es decir, de los SW.

Mientas que los implementados en el fichero MC_datos.vhd tiene la función de llevar el control de los eventos en la memoria caché. Los cuales se rigen mediante señales provenientes de la UC las cuales indican fallo de lectura (inc_rm), fallo de escritura (inc_wm) y acierto de escritura (inc_wh).

```
cont_rm: counter port map (clk => clk, reset => reset, count_enable => inc_rm , load=> '0', D_in => "00000000", count => rm);
cont_wm: counter port map (clk => clk, reset => reset, count_enable => inc_wm , load=> '0', D_in => "00000000", count => wm);
cont_wh: counter port map (clk => clk, reset => reset, count_enable => inc_wh , load=> '0', D_in => "00000000", count => wh);
```

Descripción algorítmica

```
look-up (@x);                                                    [1]
if miss (@x) {
        switch (proc_r/w)
                case  proc_r: { Mp(rB, @x); waitfor Mp; Mc +x;   [CrB(Mp) +1]
                case proc_w: Mp(wW,X`);                          [CwW]
}
else {
        switch (proc_r/w)
                case proc_r: ret x`;                             [0]
                case proc_w: {Mc + x`; Mp(wW, X`); }             [CwW]
}
```

En caso de acierto de lectura supone 1 ciclo de memoria.

En caso de fallo de lectura la penalización será CrB(Mp) + 1 donde CrB(Mp) = 8 ciclos (para la primera palabra) + 3* 2 ciclos (para las palabras restantes).

En caso de acierto de escritura o fallo de escritura, la penalización será CwW = 6 ciclos.

Ceff = Ca + (Σwh * CwW) / ΣRefs + (Σ wm * CwW) / ΣRefs + (Σrm * (Crb + 1)) / ΣRefs

Ceff = 1 + (Σw * CwW) / Σ Refs + (Σrm * (Crb + 1)) / ΣRefs

wh = write hit
wm = write miss
rm = read miss
w = write (ya sea miss o hit)

Pruebas

test_cache.vhd

```
001000  00011   00011   0000000000010000        La  R3, 16(R3)      20630010
001000  00000   00000   0000000000000100        La  R0, 4(R0)       20000004
000011  00001   00000   0000000000000000        SW  R0, 0(R1)       0C200000
000010  00001   00010   0000000000000000        LW  R2, 0(R1)       08220000
000011  00001   00001   0000000000000000        SW  R1, 0(R1)       0C210000
000001  00001   00011   00001   00000   000000  ADD R1, R1, R3      04230800
000100  00100   00100   1111111111111011        beq r4, r4, dir3    1084FFFB
```

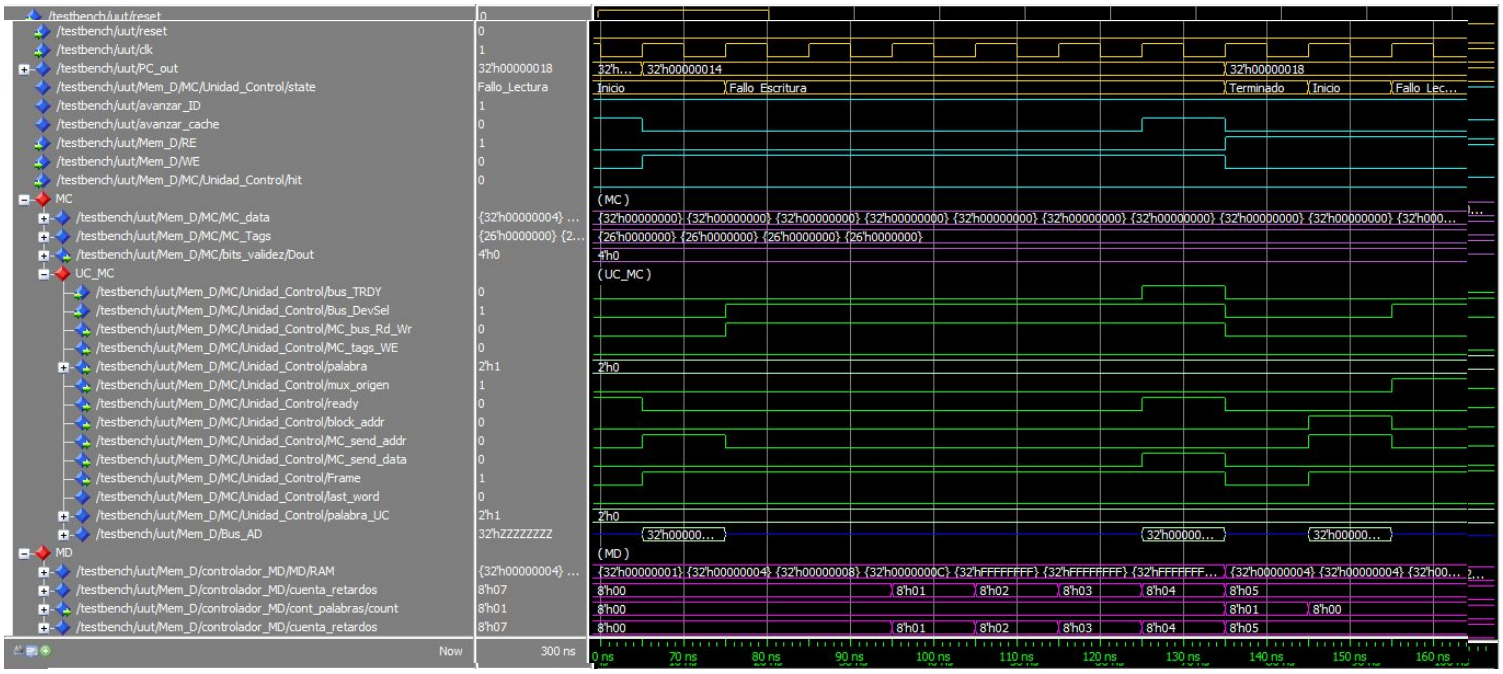
```
LA
LA
SW      -- wm
LW      -- rm and rh
SW      -- wh
ADD
BEQ
```

Este test comprueba todos los casos que se pueden dar con la manipulación de datos tanto en MD como en MC (wm, wh, rm, rh). En las primeras 4 iteraciones se usa un conjunto distinto de la MC y en la 5ª iteración se comienzan a sobreescribir los conjuntos ya usados.

En el ciclo 4 entra en la etapa de memoria la instrucción LA. Como no escribe ni lee en memoria, la UC_MC se queda en el estado INICIO por lo que ready permanece a 1 para recibir la siguiente instrucción en el siguiente ciclo.

En el ciclo 4 entra la siguiente instrucción que es también LA y se repite el mismo comportamiento.

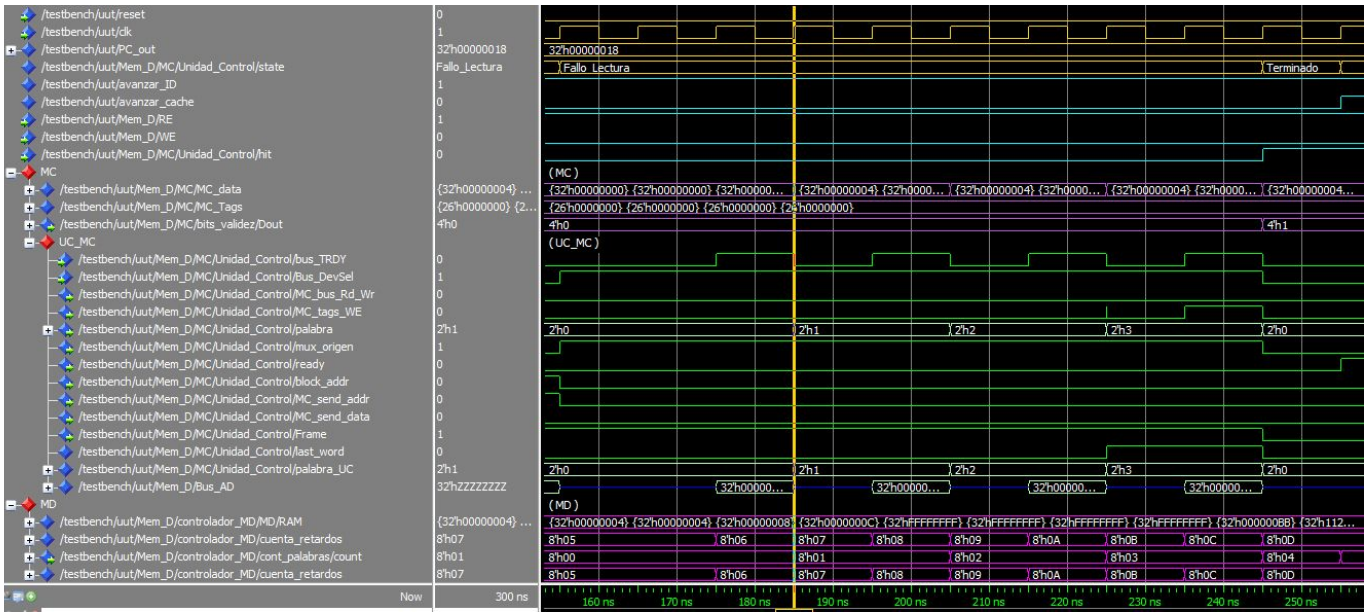
<u>1ª iteración:</u>



Write miss:

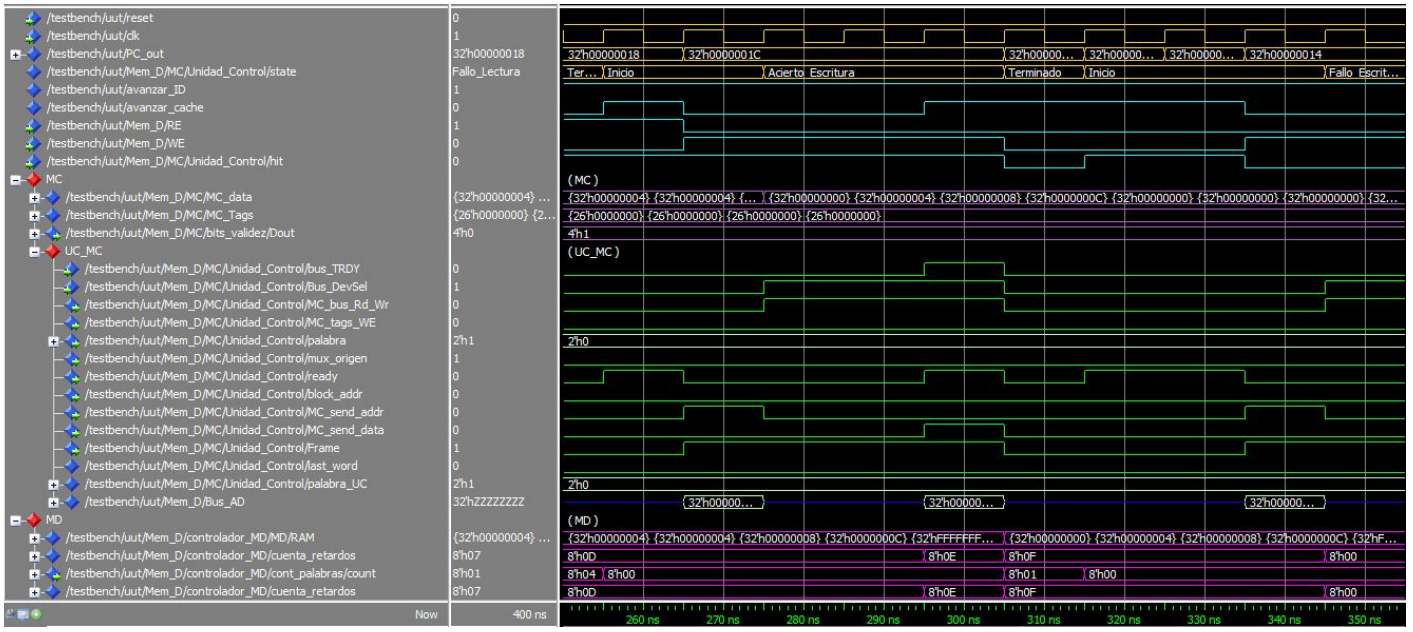
Cuando entra el primer SW en la etapa de memoria, se produce un fallo de escritura porque el bit valid de la dirección del tag al que apunta está a cero. Por lo tanto inicia una

nueva transmisión con la MD, ready permanece a 0, al no ser la misma dirección de acceso que la última vez la memoria genera un delay de cuatro ciclos, la MD escribe la palabra, y desbloquea de nuevo el mips con la señal ready a 1 para recibir la siguiente instrucción.



Read miss:

Cuando entra el LW en la fase de memoria, el bit valid del tag al que apunta se encuentra a 0, por lo tanto se produce un fallo de lectura y se trae el bloque de memoria palabra por palabra guardándolo en el primer conjunto de la MC y actualiza el conjunto con su tag correspondiente y activando el vid valid del mismo. En este caso como estamos accediendo a la misma posición en memoria de manera consecutiva, la memoria no genera ningún tipo de delay artificialmente, por eso todas las palabras son transferidas en dos ciclos.



Read hit:

Cuando el LW anterior vuelve al estado INICIO después del rm, vuelve a intentar leer el dato en la MC produciéndose un acierto de lectura y por lo tanto pone la señal ready a 1 desbloqueando el mips.

Write hit:

Cuando la segunda instrucción SW entra en la fase de memoria, el tag de la dirección donde quiere escribir coincide con el tag almacenado en la MD por lo que se produce un acierto de escritura. Escribe en el dato en la MC y posteriormente en la MD

En las siguientes 3 iteraciones, se produce el mismo comportamiento que en la primera iteración pero sobre los tres siguientes conjuntos de la MC.

A partir de la 5ª iteración se sobreescriben los conjuntos de la MC en los fallos de lectura debido a que todos ellos están ocupados.

test_contadores.vhd

Este test comprueba el funcionamiento de los contadores y tratamiento del nuevo riesgo de datos por la UD.

```
000011  00100   00011   0000000000001000        SW  R3, 8(R4)       0C830008
000010  00010   00001   0000000000000000        LW  R1, 0(R2)       08410000
000010  00100   00011   0000000000000000        LW  R3, 0(R4)       08830000
000011  00100   00011   0000000000001000        SW  R3, 8(R4)       0C830008
000101  00001   00011   1111111111111011        bne r1, r3, dir0;   1423FFFB
000010  00100   00011   0000000000000100        LW  R3, 4(R4)       08830004
000101  00001   00011   1111111111111001        bne r1, r3, dir0;   1423FFF9
```

Esta ejecución está pensada para activar mínimo una vez cada uno de los contadores.

Con la primera instrucción(sw) provocamos un write miss, entrando en el estado Fallo_escritura y activando la señal del contador wm ,debido a que el dato buscado no se encuentra cargado en la MC, provocando la activación del contador paradas_memoria durante seis ciclos, los que la MD necesita para resolver la petición. Al final de esta acción se aumenta el contador de mem_writes en uno al ponerse mem_ready a uno.

paradas_memoria	6
paradas_datos	0
paradas_control	0
mem_writes	1
mem_reads	0
wm	1
wh	0
rm	0

En la segunda instrucción (lw) se da un read miss, lo que provoca que la ejecución de nuestro autómata entre en el estado Fallo_lectura y se active la señal del contador rm, debido a que el dato buscado no se encuentra cargado en la MC. Esto causa la activación del contador paradas_memoria durante 14 ciclos en los cuales se carga en la MC el bloque donde se encuentra la palabra deseada más un ciclo adicional en el cual se vuelve a acceder a MC de forma satisfactoria con hit a

uno. Al final de esta acción se aumenta el contador de mem_writes en uno al ponerse mem_ready a uno.

paradas_memoria	21
paradas_datos	0
paradas_control	0
mem_writes	1
mem_reads	1
wm	1
wh	0
rm	1

La tercera instrucción (lw) se da un read hit, el contador mem_read aumenta en uno. A su vez en este ciclo se produce una parada del banco IF/ID y actualización del PC debido a que existe un riesgo de datos entre el LW en fase MEM y el SW en fase Decod, aumentando su consiguiente contador.

paradas_memoria	21
paradas_datos	1
paradas_control	0
mem_writes	1
mem_reads	2
wm	1
wh	0
rm	1

La cuarta instrucción (Sw) es un write hit, el contador wh aumenta en uno. Como se accede a MD se cuentan seis ciclos de paradas_memoria y aumenta en uno el contador mem_writes.

paradas_memoria	27
paradas_datos	1
paradas_control	0

mem_writes	2
mem_reads	2
wm	1
wh	1
rm	1

La quinta instrucción (bne) no provoca ningún tipo de riesgo ya que no afecta de ningún modo a la ejecución de instrucciones posteriores debido a que aunque no tiene el misma tag que el predictor de saltos como este por defecto cuando los tags son distintos predice salto no tomado y el bne no salta no es necesario ningún tipo de corrección en la ejecución de las instrucciones.

paradas_memoria	27
paradas_datos	1
paradas_control	0
mem_writes	2
mem_reads	2
wm	1
wh	1
rm	1

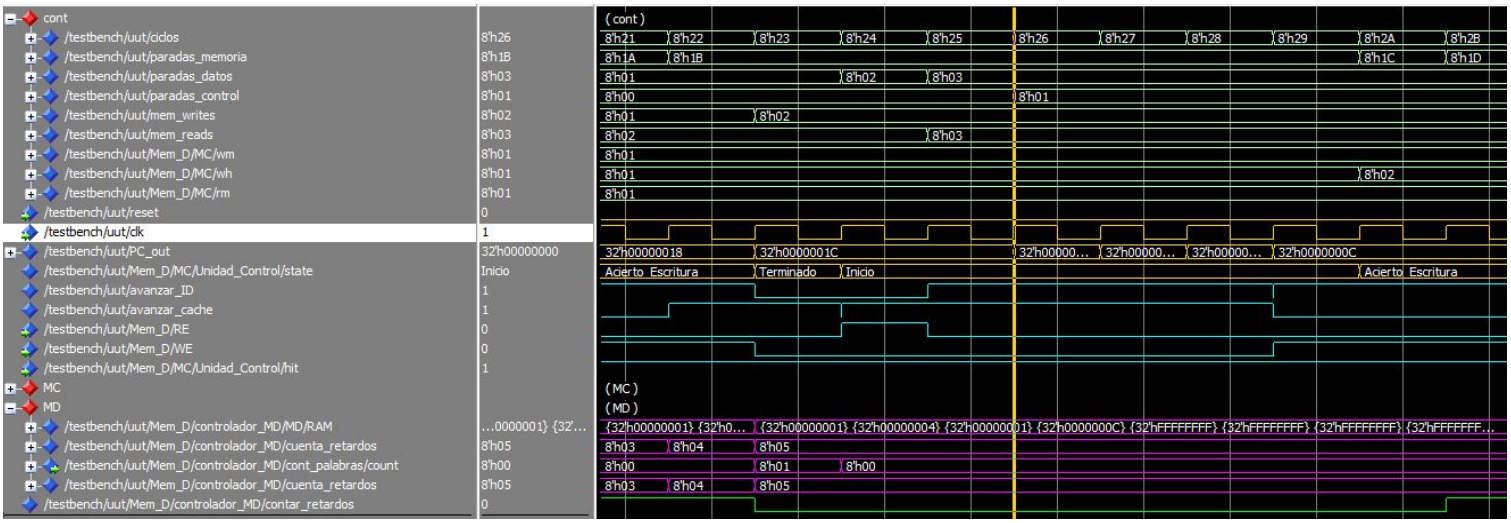
La sexta instrucción (Lw) es un read hit, por lo tanto el único contador que se ve modificado es el de mem_reads ya que su petición puede ser gestionada por el conjunto MD y MC en un ciclo.

paradas_memoria	27
paradas_datos	1
paradas_control	0
mem_writes	2
mem_reads	3
wm	1
wh	1

rm	1

La séptima instrucción (Bne) produce un riesgo de datos debido a que uno de sus registros depende de la ejecución del Lw en fase MEM por lo que se paran las etapas de fecht y decode durante dos ciclos (paradas_datos + 2), además existe un riesgo de control ya que a diferencia del anterior este si que tiene intención de saltar por lo que es necesario insertar nops en la etapa id y corregir la dirección del pc.

paradas_memoria	27
paradas_datos	3
paradas_control	1
mem_writes	2
mem_reads	3
wm	1
wh	1
rm	1



En iteraciones posteriores a esta no se producirá ningún tipo de fallo ni de escritura ni de lectura y tampoco existirán riesgos de control debido a que el predictor estará actualizado ya que es un bucle infinito y siempre acertara, por el contrario después de un salto siempre se introducirán dos ciclos de riesgo de datos.

APORTACIONES Y TIEMPO DEDICADO

Diego Marco

Estudio de los fuentes	3h
Diseño inicial de la unidad de control	7h
Depuración y ajustes	15h
Memoria	6h
TOTAL:	**31h**

Alejandro Terrón

Estudio de los fuentes	3h
Diseño inicial de la unidad de control	6h
Depuración y ajustes	15h
Memoria	6h
TOTAL:	**30h**

Autoevaluación

Diego Marco

¿Crees que has cumplido los objetivos de la asignatura?

Sí, a lo largo de este cuatrimestre he aprendido a diseñar y entender la características de los procesadores monociclo, multiciclo y segmentados. Además soy capaz de comprender la utilidad y el funcionamiento de las memorias caché así como de los buses y de algunos componentes de entrada/salida. El conjunto de prácticas y proyectos que he tenido que realizar me han supuesto una gran inversión de tiempo que me ha servido para comprender mejor todos los conceptos vistos en las clases de teoría, y entender la manera de trabajar en un entorno real, como es el caso de modelsim.

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

Me pondría un 8, ya que ha sido la asignatura en la que más tiempo he invertido. Además, en general he conseguido buenos resultados en las pruebas de diagnóstico. Aunque en la última práctica he necesitado ayuda de mis compañeros para corregir algunos errores.

Alejandro Terrón

¿Crees que has cumplido los objetivos de la asignatura?

Si, esta asignatura me ha servido para comprender mejor el funcionamiento de un procesador hasta el punto de poder implementar uno y entender sus características y comportamientos ante ciertas acciones, a su vez me ha servido para terminar de comprender algunos conceptos de asignaturas anteriores como el de los buses visto por encima en asignaturas como IC y AOC 1. Tanto los trabajos en modelsim como las prácticas en logisim me han llevado bastante más tiempo que en otras asignaturas lo que a mi parecer me ha preparado para realizar proyectos de mayor envergadura a los que estaba acostumbrado.

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

Yo me pondría un 7 debido a que aunque haya sacado todas las prácticas y trabajos hacia delante he de reconocer que en algunas he necesitado algo de apoyo de mis compañeros para terminar de entender el comportamiento que se me describia en los enunciados y así poder luego realizar dichas prácticas de manera satisfactoria.

Observaciones

A la hora de implementar el autómata nos dimos cuenta de que los estados Fallo_escritura y Acierto_Escritura tienen esencialmente las mismas funciones, en lo único que se diferencian es en la transición de entrada al estado la cual varía dependiendo si es hit o miss. Pese a ello decidimos no fusionarlos ya que no suponía ninguna mejora al rendimiento y así nos resultaba más fácil hacer el debug del diseño. Otro de los problemas que tuvimos que solventar mientras realizamos las pruebas de nuestra UC_MC fue que no teníamos como tal un estado para cerciorarnos que el bus DevSel estuviese a uno, es decir, que la MD reconozca la dirección que le hemos mandado y así poder empezar a realizar las

peticiones. Esto se solucionó añadiendo a la transición de espera a que la memoria esté preparada para procesar la petición,(en los estados Acierto_Escritura, Fallo_Escritura y Fallo_Lectura), una cláusula por la cual si el bus_DevSel está a cero mantenga la dirección cargada hasta que la MD la reconozca.

El último problema que tuvimos en el diseño del autómata fue poner Frame a cero después de terminar las transmisiones de la MC con la MD, el cual como se puede ver en el segundo apartado de la memoria se a resuelto mediante un estado de transición llamado “Terminado” el cual se encarga de poner todas las señales a cero.