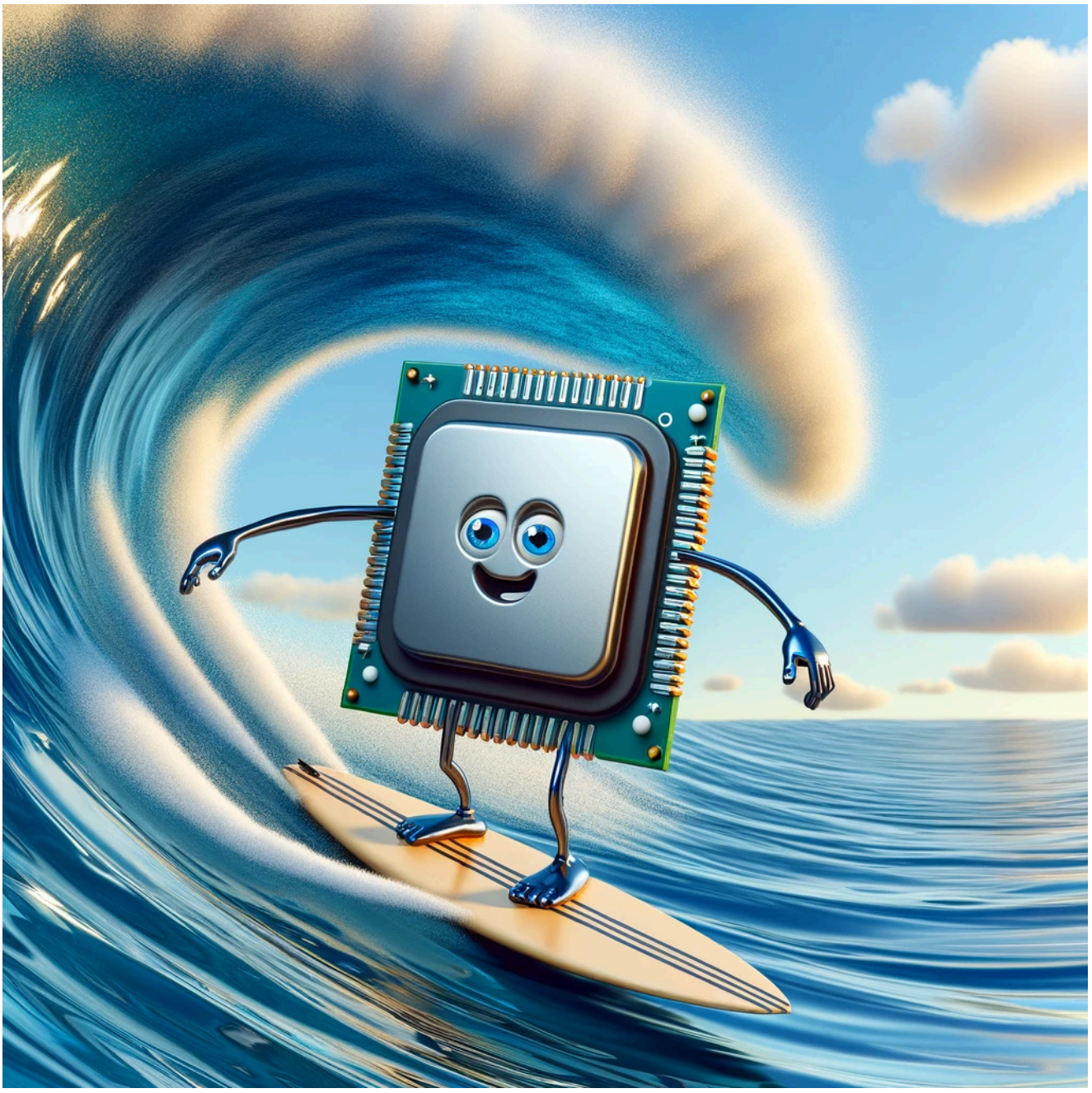
AOC2 Proyecto 2 - Jerarquía de memoria de datos

Autores: Mateo Lorente, Diego — 873338

Solana Melero, Carlos — 872815

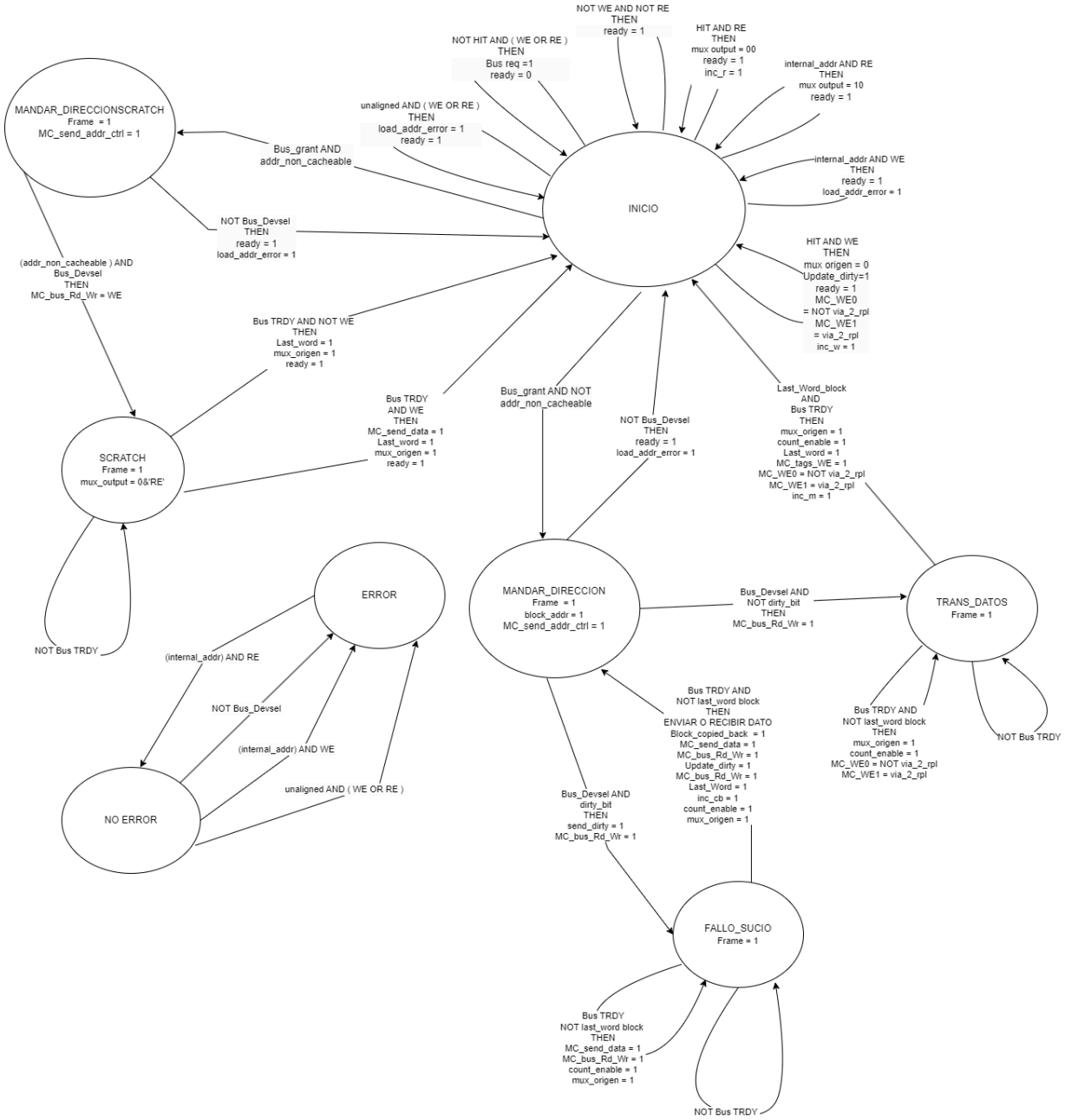
miércoles, 6 de marzo de 2024



Breve resumen

En el Proyecto 2 del curso 2023/2024 hemos modificado la gestión de memoria del proyecto 1 para que este disponga de tres tipos de memoria en vez de uno. Por un lado, tenemos la memoria principal (MP), cuyo funcionamiento va a ser de la memoria del proyecto 1. La principal diferencia respecto al proyecto 1 es la forma de acceso a dicha memoria. En este proyecto, el acceso lo hacemos a la memoria caché (MC), que contiene las direcciones de memoria de la memoria principal, pero tiene un acceso más rápido y sencillo al ser de menor tamaño. Al ser de menor tamaño, no caben todos los datos de la memoria principal, por lo que en muchos casos hay que reemplazar unas direcciones con otras. La política utilizada en este caso es la política FIFO (First In, First Out). Por último, tenemos una memoria Scratch (MD Scratch), que es más rápida que la MP, pero se accede a una dirección concreta y a una palabra concreta, y la gestión de dicha memoria se realiza mediante el bus.

Diagrama de estados de la unidad de control



La Unidad de Control que hemos diseñado para este proyecto consta de 6 estados, además de 2 estados de error. El estado inicial es el de inicio, en el que todas las transiciones hacia ese mismo estado se producen mientras no haya un miss sin errores, es decir, que haya un hit en lectura o escritura, se intente leer o escribir en un registro interno de la MC, no se lea o escriba o se lea o escriba en una dirección no alineada. Cuando llega un miss en lectura o escritura, se solicita el uso del bus compartido al árbitro activando la señal Bus_req. Una vez el árbitro te conceda el bus (Bus_grant=1), si la dirección sobre la que trabajamos no es cacheable, es decir, está en el rango de direcciones X"10000000"- X"100000FF", saltaremos al estado MANDAR_DIRECCIONSCRATCH. En caso contrario, iremos al estado MANDAR_DIRECCION. En el estado MANDAR_DIRECCION, tenemos 2 posibilidades. Por un lado, que la memoria nos confirme su dirección mediante la señal Bus_Devsel. Si eso ocurre, debemos comprobar que el dirty_bit esté o no activo. Si está activo, quiere decir que el miss es de tipo sucio, por lo que el próximo estado será FALLO_SUCIO. En él, hasta que Bus_Tready esté activo, debe estar el bus en espera. Cuando dicha señal se active, se realizará la transferencia de la primera de las 4 palabras del bloque que se va a escribir en MD. Cuando se escriba la cuarta palabra, se activará la señal last_word_block, por lo que la transferencia de datos sucios habrá terminado y volveremos al estado MANDAR_DIRECCION. Si Bus_Devsel está activo y no lo está dirty_bit, querrá decir que el fallo es de tipo limpio, y el próximo estado será TRANS_DATOS. Dicho estado va a tener un funcionamiento semejante al fallo sucio, solo que en este caso en vez de mandar datos desde MC hasta MD lo hacemos de forma inversa. Nos traemos el bloque desde MD hasta MC para reemplazar el bloque más antiguo de dicho conjunto. Una vez llegue la cuarta palabra, volveremos a INICIO. En el caso en el que la dirección sobre la que trabajemos no sea cacheable, como hemos mencionado anteriormente, saltaremos al estado MANDAR_DIRECCIONSCRATCH. Dicho estado tendrá un comportamiento semejante al de MANDAR_DIRECCION. Esperamos que la MD Scratch nos confirme que la dirección es correcta mediante la señal Bus_Devsel para pasar al estado SCRATCH. En él, al igual que en los 2 estados de transferencia, debemos esperar hasta que Bus_TRDY esté activo. Una vez lo esté, dependiendo de la operación que solicitara la instrucción. Se leerá una palabra de la Scratch o se escribirá una palabra en ella. En ambos casos transicionamos al estado de inicio. Con respecto a los 2 estados de error mencionados al principio del apartado, estos van aparte del autómata, y están conectados ambos estados entre sí. Uno de ellos será cuando no haya ningún dato en el registro interno de MC (NO ERROR) y otro que será cuando sí lo haya (ERROR). Dentro del estado de NO ERROR, para pasar al otro estado, hay 3 formas: Intentando escribir en el registro de error, cuando la dirección no está alineada o cuando no te confirma la memoria mediante Bus_Devsel que es la dirección correcta. Para pasar del estado de ERROR a NO ERROR, basta con hacer una lectura del registro interno de la MC. Una vez leído, se vaciará y pasaremos al estado de NO ERROR.

Descomposición de la dirección

Esquema con la descomposici´on de la dirección para el direccionamiento de la MC como hemos enseñado en clase (byte/word – set – tag)

La descomposición que se ha utilizado sobre la dirección para este proyecto ha sido la siguiente

```
tag <= ADDR(31 downto 6);

dir_word <= ADDR(3 downto 2) when (mux_origen='0') else palabra_UC;

dir_cjto <= ADDR(5 downto 4); -- es emplazamiento asociativo
```

Es decir:

- El tag son los 26 bits de más peso de la dirección.
- La dirección del conjunto son los 2 siguientes bits.(ya que hay 4 conjuntos)
- La dirección de la palabra son los 2 siguientes bits.(en cada conjunto hay 4 palabras)
- Los últimos 2 bits los utilizamos para especificar el byte de la palabra en cuestión.(en cada palabra hay 4 bits

Análisis de las latencias de las distintas transferencias en el bus

- CrB(MD): ciclos para leer un bloque de MD, dónde CrB(MD) = L ciclos (para la primera palabra) + 3*R ciclos (para las palabras restantes).

  CRB(MD) = 7 + 3*2 = 13 ciclos

- CwB(MD): ciclos para escribir un bloque en MD.

  CwB(MD) = 7 + 3*2 = 13 ciclos

- CrW(MDscratch): ciclos para leer una palabra de MD Scratch.

  CrW(MDscratch) = 2 ciclos

- CwW(MDscratch): ciclos para escribir una palabra en MD Scratch:

  CwW(MDscratch) = 2 ciclos

- CrW o CwW(IO REG): ciclos para leer o escribir los registros internos de entrada y salida.

  CrW = CwW = 1 ciclo

Expresión de cálculo de los ciclos efectivos

Para calcular la expresión de los ciclos efectivos de nuestro sistema, contamos los ciclos que cuesta cada una de las 4 acciones principales que se realizan en él (Crb(MD), CwB(MD), CrW(MDscratch), CwW(MDscratch)). Los ciclos que cuesta cada acción quedan representados en la siguiente tabla:

Evento	Arbitraje	Send Addr	Send Data	Total
Crb(MD)	1.5 ciclos	1 ciclo	12 ciclos	14.5 ≈ **15 ciclos**
CwB(MD)	1.5 ciclos	1 ciclo	12 ciclos	14.5 ≈ **15 ciclos**
CrW(MDscratch)	1.5 ciclos	1 ciclo	1 ciclo	3.5 ≈ **4 ciclos**
CwW(MDscratch)	1.5 ciclos	1 ciclo	1 ciclo	3.5 ≈ **4 ciclos**

Una vez calculados los ciclos totales, la expresión resultante para calcular los ciclos será la siguiente:

$$Ceff = 1 + \frac{\Sigma rm \cdot 15}{\Sigma ref} + \frac{\Sigma wm \cdot 15}{\Sigma ref} + \frac{\Sigma rscratch \cdot 4}{\Sigma ref} + \frac{\Sigma w\,scratch \cdot 4}{\Sigma ref}$$

Programas de prueba

Prueba fallo limpio en lectura y acierto en lectura, lectura Reg. interno MC y Reg. IO

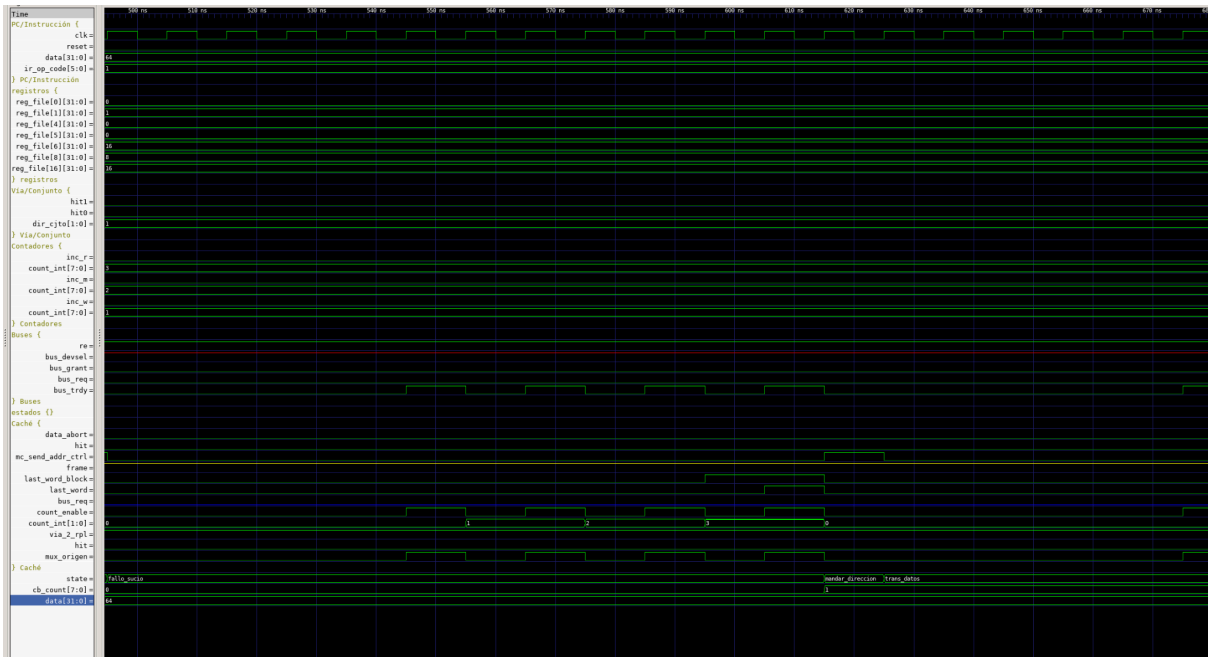
	Direccion	Codificación	Código	Comentario
	0x0	10210003	beq R1, R1, INI;	
	0x4	1021003E	beq R1, R1, RTI;	
	0x8	1021005D	beq R1, R1, RT_Abort;	
	0xC	1021006C	beq R1, R1, RT_UNDEF;	
INI	0x10	08010000	Lw R1, 0(r0)	Fallo limpio en lectura en cjto 0 via 0. r++,m++
	0x14	04212000	Add R4, R1,R1	
	0x18	04842000	Add R4, R4,R4	
	0x1C	04844000	Add R8, R4,R4	
	0x20	05088000	Add R16, R8,R8	
	0x24	06003000	Add R6, R16,R0	Despl=16
	0x28	08040004	Lw R4, 4(r0)	hit en lectura. r++
Bucle1	0x2C	08C20000	Lw R2, 0(r6)	Fallo limpio en lectura en cjto 0. r++. m++
	0x30	08C20004	LW R2, 4(r6)	Hit en lectura. r++
	0x34	06063000	Add R6, R16,R6	Despl=Despl+16
	0x38	04252800	Add R5, R1,R5	Iteración
	0x3C	1000FFFB	BEQ R0,R0, bucle1	Bucle hasta que se genere el abort por caché llena
RTI:	0x100	08010000	Lw R1, 0(r0)	R1=1
	0x104	0C017008	sw r1, 7008(r0)	INT_ACK <= 1 ; (dura un ciclo)
	0x108	20000000	rte	Se vuelve a la instrucción que se interrumpió
RT_Abort	0x180	08020104	LW R2, 104(R0)	R2=Mem(66) =0x00000AB0;
	0x184	0C027004	sw r2, 7004(r0)	IO_output <=0x00000AB0; . 1 ciclo lw-uso
	0x188	08020108	LW R2, 108(R0)	R2=Mem(67) =0x01000000;
	0x18C	08420000	LW R2, 0(R2)	R1=Error_addr_register. Leemos el registro interno de MC. La señal de error desaparece . 1 ciclo lw-uso
	0x190	0C027004	sw r2, 7004(r0)	IO_output <=Dirección que causó el abort; Lw-uso 1 ciclos de detención
	0x194	20000000	rte	Se vuelve a la instrucción que se interrumpió. Parada control retorno
RT_UNDEF	0x1C0	0802010C	LW R2, 10C(R0)	R2=Mem(68) = 0x0BAD0C0D;
	0x1C4	0C027004	sw r2, 7004(r0)	IO_output <= 0x0BAD0C0D; Parada lw-uso
	0x1C8	1000FFFF	beq R0, R0, bucleU	Bucle infinito

Este es el código suministrado por los profesores de la asignatura para probar tanto el acierto como el miss limpio en lectura. El bucle consiste en hacer un miss limpio,reemplazar el bloque por otro y probar que la siguiente lectura haga hit de lectura. Además, una vez llega el Data Abort por llenado de caché, se carga un valor en el registro interno de IO, por lo que eso también queda probado

Prueba fallo sucio en lectura

	Direccion	Codificación	Código	Comentario
	0x0	10210003	beq R1, R1, INI;	
	0x4	1021003E	beq R1, R1, RTI;	
	0x8	1021005D	beq R1, R1, RT_Abort;	
	0xC	1021006C	beq R1, R1, RT_UNDEF;	
INI	0x10	08010000	Lw R1, 0(r0)	Fallo limpio en lectura en cjto 0 via 0. r++,m++
	0x14	04212000	Add R4, R1,R1	
	0x18	04842000	Add R4, R4,R4	
	0x1C	04844000	Add R8, R4,R4	
	0x20	05088000	Add R16, R8,R8	
	0x24	06003000	Add R6, R16,R0	Despl=16
	0x28	08040004	Lw R4, 4(r0)	hit en lectura. r++
Bucle1	0x2C	08C20000	Lw R2, 0(r6)	Fallo limpio en lectura en cjto 0. r++. m++
	0x30	0CC10000	Sw R1, 0(r6)	Hit en escritura. w++
	0x34	08C20040	Lw R2, 64(r6)	Fallo limpio en lectura. r++. m++
	0x38	08C20080	LW R2, 128(r6)	Fallo sucio en lectura. r++. m++
	0x3C	06063000	Add R6, R16,R6	Despl=Despl+16
	0x40	04252800	Add R5, R1,R5	Iteración
	0x44	100000F9	BEQ R0,R0, bucle1	Bucle hasta que se genere el abort por caché llena
RTI:	0x100	08010000	Lw R1, 0(r0)	R1=1
	0x104	0C017008	sw r1, 7008(r0)	INT_ACK <= 1 ; (dura un ciclo)
	0x108	20000000	rte	Se vuelve a la instrucción que se interrumpió
RT_Abort	0x180	08020104	LW R2, 104(R0)	R2=Mem(66) =0x00000AB0;
	0x184	0C027004	sw r2, 7004(r0)	IO_output <=0x00000AB0; . 1 ciclo lw-uso
	0x188	08020108	LW R2, 108(R0)	R2=Mem(67) =0x01000000;
	0x18C	08420000	LW R2, 0(R2)	R1=Error_addr_register. Leemos el registro interno de MC. La señal de error desaparece . 1 ciclo lw-uso
	0x190	0C027004	sw r2, 7004(r0)	IO_output <=Dirección que causó el abort; Lw-uso 1 ciclos de detención
	0x194	20000000	rte	Se vuelve a la instrucción que se interrumpió. Parada control retorno
RT_UNDEF	0x1C0	0802010C	LW R2, 10C(R0)	R2=Mem(68) = 0x0BADOC0D;
	0x1C4	0C027004	sw r2, 7004(r0)	IO_output <= 0x0BADOC0D; Parada lw-uso
	0x1C8	1000FFFF	beq R0, R0, bucleU	Bucle infinito

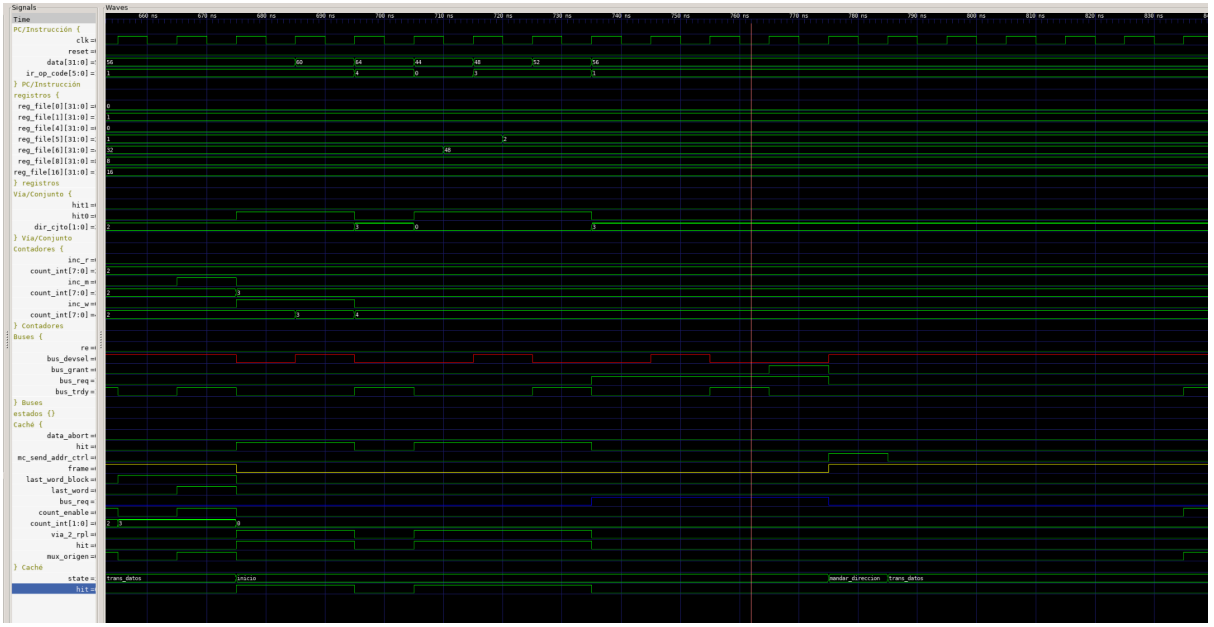
En este código, probamos de forma simple que funciona el fallo de tipo sucio en lectura. En el bucle, cargamos un bloque en MC. Después, modificamos el contenido de dicho bloque para activar dirty_bit, generamos un miss limpio para reemplazar el bloque de la vía 1 y, finalmente, generamos otro miss que reemplaza el bloque de la vía 0 y, al estar activo el dirty bit, se genera el fallo sucio correctamente.



Prueba fallo limpio en escritura y hit en escritura

	Direccion	Codificación	Código	Comentario
	0x0	10210003	beq R1, R1, INI;	
	0x4	1021003E	beq R1, R1, RTI;	
	0x8	1021005D	beq R1, R1, RT_Abort;	
	0xC	1021006C	beq R1, R1, RT_UNDEF;	
INI	0x10	08010000	Lw R1, 0(r0)	Fallo limpio en lectura en cjto 0 via 0. r++. m++
	0x14	04212000	Add R4, R1,R1	
	0x18	04842000	Add R4, R4,R4	
	0x1C	04844000	Add R8, R4,R4	
	0x20	05088000	Add R16, R8,R8	
	0x24	06003000	Add R6, R16,R0	Despl=16
	0x28	08040004	Lw R4, 4(r0)	hit en lectura. r++
Bucle1	0x2C	0CC20000	Sw R2, 0(r6)	Fallo limpio en escritura en cjto 0. w++. m++
	0x30	0CC20004	Sw R2, 4(r6)	Hit en escritura. w++
	0x34	06063000	Add R6, R16,R6	Despl=Despl+16
	0x38	04252800	Add R5, R1,R5	Iteración
	0x3C	1000FFFB	BEQ R0,R0, bucle1	Bucle hasta que se genere el abort por caché llena
RTI:	0x100	08010000	Lw R1, 0(r0)	R1=1
	0x104	0C017008	sw r1, 7008(r0)	INT_ACK <= 1 ; (dura un ciclo)
	0x108	20000000	rte	Se vuelve a la instrucción que se interrumpió
RT_Abort	0x180	08020104	LW R2, 104(R0)	R2=Mem(66) =0x00000AB0;
	0x184	0C027004	sw r2, 7004(r0)	IO_output <=0x00000AB0; . 1 ciclo lw-uso
	0x188	08020108	LW R2, 108(R0)	R2=Mem(67) =0x01000000;
	0x18C	08420000	LW R2, 0(R2)	R1=Error_addr_register. Leemos el registro interno de MC. La señal de error desaparece . 1 ciclo lw-uso
	0x190	0C027004	sw r2, 7004(r0)	IO_output <=Dirección que causó el abort; Lw-uso 1 ciclos de detención
	0x194	20000000	rte	Se vuelve a la instrucción que se interrumpió. Parada control retorno
RT_UNDEF	0x1C0	0802010C	LW R2, 10C(R0)	R2=Mem(68) = 0x0BAD0C0D;
	0x1C4	0C027004	sw r2, 7004(r0)	IO_output <= 0x0BAD0C0D; Parada lw-uso
	0x1C8	1000FFFF	beq R0, R0, bucleU	Bucle infinito

En esta prueba, comprobamos el correcto funcionamiento de la memoria de nuestro procesador gestionando el miss de tipo limpio en escritura. Creamos un bucle el cual modifica constantemente el tag, por lo que el primer sw será fallo limpio de escritura y el segundo dará hit para comprobar que todo funciona correctamente. Cuando se llene la caché, saltará el Abort y entrará en un bucle infinito.



Prueba fallo sucio en escritura

No va a ser necesario probar esto, ya que el funcionamiento es exactamente igual a lectura en sucio: mover el bloque de MC a MD. Después se aplica fallo limpio. Como ya hemos probado tanto lectura sucia como escritura limpia, damos este caso por probado.

Prueba lectura y escritura en Scratch

	Direccion	Codificación	Código	Comentario
	0x0	10210003	beq R1, R1, INI;	
	0x4	1021003E	beq R1, R1, RTI;	
	0x8	1021005D	beq R1, R1, RT_Abort;	
	0xC	1021006C	beq R1, R1, RT_UNDEF;	
INI	0x10	08010000	Lw R1, 0(r0)	Fallo limpio en lectura en cjto 0 via 0
	0x14	04212000	Add R4, R1,R1	
	0x18	04842000	Add R4, R4,R4	
	0x1C	04844000	Add R8, R4,R4	
	0x20	05088000	Lw R6, FF(r0)	Guardamos direccion no cacheable en R6
	0x24	06003000	Add R6, R16,R0	Despl=16
	0x28	08040004	Lw R4, 4(r0)	R4=0
Bucle1	0x2C	0CC20000	Sw R2, 0(r6)	Lectura en Scratch
	0x30	08CB0000	lw r11, 0(r6)	Escritura en Scratch
	0x34	06063000	Add R6, R16,R6	Despl=Despl+16
	0x38	04411000	Add R2, R2, R1	
	0x3C	1000FFFB	BEQ R0,R0, bucle1	Bucle hasta que se genere el abort por caché llena
RTI:	0x100	08010000	Lw R1, 0(r0)	R1=1
	0x104	0C017008	sw r1, 7008(r0)	INT_ACK <= 1 ; (dura un ciclo)
	0x108	20000000	rte	Se vuelve a la instrucción que se interrumpió
RT_Abort	0x180	08020104	LW R2, 104(R0)	R2=Mem(66) =0x00000AB0;
	0x184	0C027004	sw r2, 7004(r0)	IO_output <=0x00000AB0; . 1 ciclo lw-uso
	0x188	08020108	LW R2, 108(R0)	R2=Mem(67) =0x01000000;
	0x18C	08420000	LW R2, 0(R2)	R1=Error_addr_register. Leemos el registro interno de MC. La señal de error desaparece . 1 ciclo lw-uso
	0x190	0C027004	sw r2, 7004(r0)	IO_output <=Dirección que causó el abort; Lw-uso 1 ciclos de detención
	0x194	20000000	rte	Se vuelve a la instrucción que se interrumpió. Parada control retorno
RT_UNDEF	0x1C0	0802010C	LW R2, 10C(R0)	R2=Mem(68) = 0x0BAD0C0D;
	0x1C4	0C027004	sw r2, 7004(r0)	IO_output <= 0x0BAD0C0D; Parada lw-uso
	0x1C8	1000FFFF	beq R0, R0, bucleU	Bucle infinito

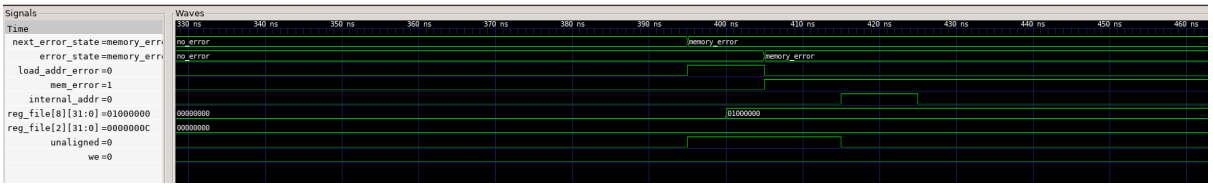
El código de esta prueba es bastante simple, antes del bucle cargamos en r6 una dirección no cacheable, y en el bucle metemos un número en la Scratch y luego lo leemos en otro registro. Por tanto, probamos lectura y escritura en Scratch

Prueba acceso no alineado y escritura en Reg. interno MC

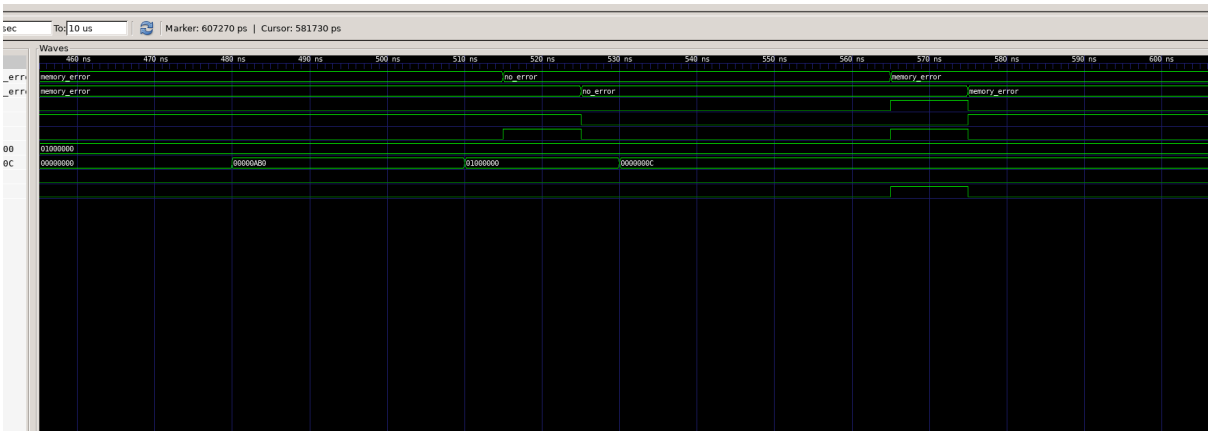
	Direccion	Codificación	Código	Comentario
	0x0	10210003	beq R1, R1, INI;	
	0x4	1021003E	beq R1, R1, RTI;	
	0x8	1021005D	beq R1, R1, RT_Abort;	
	0xC	1021006C	beq R1, R1, RT_UNDEF;	
INI	0x10	08010000	Lw R1, 0(r0)	R1=4
	0x14	08080108	Lw R8, 264(r0)	R2 = 0x01000000 (reg interno MC)
	0x18	0803000F	Lw R3, 15(r0)	Dirección no alineada
	0x1C	04210800	ADD R1,R1,R1	
	0x20	0D010000	Sw R1, 0(r8)	No puedo escribir en reg interno MC, error
	0x24	100000FF	BEQ R0,R0, -1	Bucle infinito
RTI:	0x100	08010000	Lw R1, 0(r0)	R1=1
	0x104	0C017008	sw r1, 7008(r0)	INT_ACK <= 1 ; (dura un ciclo)
	0x108	20000000	rte	Se vuelve a la instrucción que se interrumpió
RT_Abort	0x180	08020104	LW R2, 104(R0)	R2=Mem(66) =0x00000AB0;
	0x184	0C027004	sw r2, 7004(r0)	IO_output <=0x00000AB0; . 1 ciclo lw-uso
	0x188	08020108	LW R2, 108(R0)	R2=Mem(67) =0x01000000;
	0x18C	08420000	LW R2, 0(R2)	R1=Error_addr_register. Leemos el registro interno de MC. La señal de error desaparece . 1 ciclo lw-uso
	0x190	0C027004	sw r2, 7004(r0)	IO_output <=Dirección que causó el abort; Lw-uso 1 ciclos de detención
	0x194	20000000	rte	Se vuelve a la instrucción que se interrumpió. Parada control retorno
RT_UNDEF	0x1C0	0802010C	LW R2, 10C(R0)	R2=Mem(68) = 0x0BAD0C0D;
	0x1C4	0C027004	sw r2, 7004(r0)	IO_output <= 0x0BAD0C0D; Parada lw-uso
	0x1C8	1000FFFF	beq R0, R0, bucleU	Bucle infinito

En esta prueba, probamos los casos no probados anteriormente: Intentamos leer una dirección no alineada, por lo que salta el error y se guarda en el registro interno de la MC la dirección de la palabra que se intentaba meter. Después, leemos el contenido de dicho registro para quitar el valor del registro y quitar el error. Por último, intentamos escribir en dicho registro, lo que da error y pone en el registro un valor y, finalmente, volvemos a leer dicho contenido para hacer desaparecer el error. Para esta prueba se utilizará la RAM 2 de datos.

Lectura de dirección no alineada:



Escritura en reg. interno de la MC:



Cálculo speedup

Calculad el speedup que aportan la MC y la MD Scratch con respecto a un sistema similar que solo tenga la MD de este proyecto (no la MD del proyecto anterior). Podéis reutilizar uno de vuestros programas de prueba, si es apropiado para evidenciar el papel de la caché. Podéis realizar el cálculo del speedup de forma teórica, o a partir de los ciclos que obtengáis mediante simulación de la ejecución del programa sobre cada sistema (c/s caché).

Para calcular el speedup, hemos creado un programa específico sencillo para usarlo como referencia en el cálculo. Consiste en un código compuesto únicamente por lw y sw, que en cuestión de ciclos son las instrucciones con mayor diferencia de ciclos entre nuestro sistema y el sistema sin MC y Scratch. El cálculo será realizado de forma teórica. Dicho programa es el siguiente:

Dirección	Instrucción
0x0	Lw R1, 0(r0)
0x4	Lw R2, 4(r0)
0x8	sw R1, 0(r0)
0xC	Lw R1, 16(r0)
0x10	Lw R1, 20(r0)
0x14	Sw R1, 8(r0)
0x18	Lw R1, 12(r0)
0x1C	Sw R1,24(r0)
0x20	Lw R1, 28(r0)
0x24	Lw R1, 40(r0)
0x28	BEQ R0,R0,-1

Dicho programa contiene 3 miss y 7 hit, por lo que el número de ciclos será 52ciclos, mientras que si lo ejecutamos con un sistema sin estas memorias nos quedarían 9 ciclos por cada lw o sw (6 ciclos primera palabra + 1.5 ciclos arbitraje(asumo todavía implementado) + 1 ciclo envío dirección = 8.5 ciclos ≈ 9 ciclos), que multiplicado por las 10 instrucciones de nuestro programa (sin contar el bucle infinito) nos quedan 90 ciclos. Por tanto, el speedup de este programa, siendo el tiempo de ciclo Tc de ambos sistemas el mismo, quedaría así:

$$Speedup = \frac{Tex\_noMC}{Tex\_MC} = \frac{ciclos\_noMC \cdot Tc}{ciclos\_MC \cdot Tc} = \frac{ciclos\_noMC}{ciclos\_MC} = \frac{90\ ciclos}{52\ ciclos} = 1.7307$$

Cabe destacar que este Speedup puede variar bastante en función del número de miss y hit que haya en el programa. En este caso, al haber tantos hits y tan pocos misses, queda un Speedup bastante grande.

Cuantificación de horas dedicadas

- Estudio del enunciado
    - Carlos: 1
    - Diego: 1
- Creación del autómata
    - Carlos:5
    - Diego: 5
- Código en vhdl
    - Carlos: 4
    - Diego: 3
- Depuración, verificación y programas de prueba:
    - Carlos: 13
    - Diego: 10
- Memoria:
    - Carlos: 2
    - Diego: 5

Es importante recalcar que muchas de las horas que nos hemos contado individualmente han sido cuando hemos estado los 2 juntos, por lo que la gran mayoría de horas individuales son las mismas que las horas trabajadas en pareja.

Conclusiones y Autoevaluación

Consideramos que hemos cumplido los objetivos de la asignatura, ya que hemos entregado y entendido todos los ejercicios propuestos. El proyecto 1 lo entregamos correctamente. Hemos entendido profundamente el MIPS, así como maneras de optimizarlo haciéndolo multiciclo con detenciones, también hemos entendido la lentitud de la memoria principal y cómo reducirla con la caché. El trabajo nos ha gustado por lo general, aunque consideramos ligeramente tedioso la depuración con gtkwave. Creemos que la nota que nos merecemos es un 8 ya que no solo entendemos los fundamentos sino que además hemos profundizado en ellos.

Agradecimientos

Agradecemos la colaboración a la hora de responder las dudas que nos iban surgiendo a algunos compañeros. Estos compañeros son:

- Emilliano Recuenco López (868419)
- Jorge Gallardo Jaso(868801)
- Daniel Simón Gayán (870984)
- José Miguel Quílez Vergara (873499)
- Yago Torres García (878417)
- Raúl Soler Fernández (875458)
- Enrique Baldovin Cotela (869402)