Proyecto 2 – 2023 – Jerarquía de memoria de datos

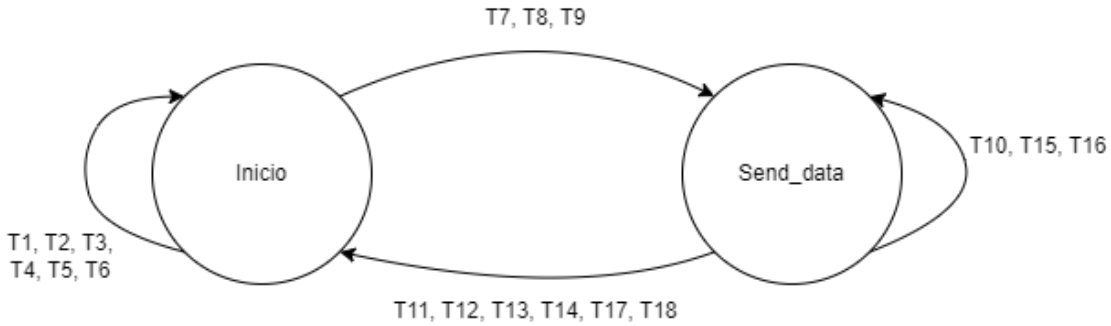
Inés Román Gracia, 820731

Álvaro Ramos Fustero, 844798

Domingo, 14 de Mayo de 2023

1. Diagrama de estados de la unidad de control

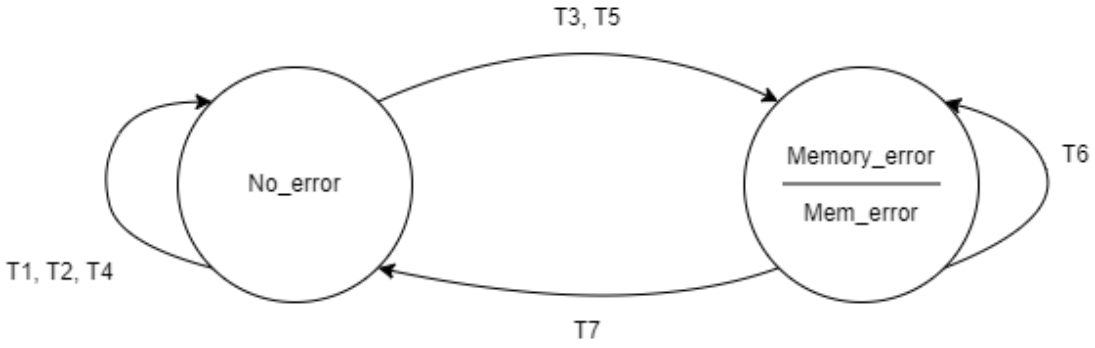
MÁQUINA DE ESTADOS PARA LA TRANSFERENCIA DE DATOS



Transición	Entradas	Salidas
T1	RE = 0 and WE = 0	Ready
T2	(RE or WE) and unaligned	Ready, load_addr_error
T3	RE and internal_addr	Ready, mux_output = 10
T4	RE and hit	Ready, mux_output = 00
T5	(RE or WE) and Bus_grant = 0	Bus_Req
T6	(RE or WE) and Bus_grant and Bus_DevSel = 0	Bus_Req, MC_send_data_ctr, Ready, load_addr_error
T7	(RE or WE) and Bus_grant and Bus_DevSel and hit = 0 and addr_non_cacheable = 0	Bus_Req, MC_send_data_ctrl, block_addr, inc_m
T8	WE and Bus_grant and Bus_DevSel and (hit or addr_non_cacheable)	Bus_Req, MC_send_data_ctrl, Mc_bus_Rd_Wr
T9	RE and Bus_grant and Bus_DevSel and addr_non_cacheable	Bus_Req, MC_send_data_ctrl

T10	Bus_TRDY = 0	Frame
T11	RE and addr_non_cacheable and Bus_TRDY	Frame, last_word, mux_output = 01, Ready
T12	WE and addr_non_cacheable and Bus_TRDY	Frame, last_word, inc_w, MC_send_data, Ready
T13	WE and hit and Bus_TRDY and hit0	Frame, inc_w, MC_WE0, MC_send_data, last_word, Ready
T14	WE and hit and Bus_TRDY and hit1	Frame, inc_w, MC_WE1, MC_send_data, last_word, Ready
T15	(WE or RE) and hit = 0 and Bus_TRDY and last_word_block = 0 and via_2_rpl = 0	Frame, count_enable, mux_origen, MC_WE0
T16	(WE or RE) and hit = 0 and Bus_TRDY and last_word_block = 0 and via_2_rpl = 1	Frame, count_enable, mux_origen, MC_WE1
T17	(WE or RE) and hit = 0 and last_word_block and Bus_TRDY and via_2_rpl = 0	Frame, count_enable, last_word, mux_origen, MC_WE0, MC_TAGS_WE
T18	(WE or RE) and hit = 0 and last_word_block and Bus_TRDY and via_2_rpl = 1	Frame, count_enable, last_word, mux_origen, MC_WE1, MC_tags_WE

MÁQUINA DE ESTADOS PARA EL CONTROL DE ERRORES

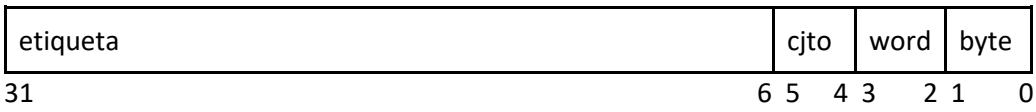


Transición	Entradas
T1	RE = 0 and WE = 0
T2	(RE or WE) and unaligned = 0
T3	(RE or WE) and unaligned

T4	(RE or WE) and MC_bus_grant and Bus_DevSel
T5	(RE or WE) and MC_bus_grant and Bus_DevSel = 0
T6	RE = 0 or internal_addr = 0
T7	RE and internal_addr

2. Descomposición de la dirección

En este sistema de memoria, la dirección se descompone de la siguiente forma para su almacenamiento correcto en la memoria caché.



3. Análisis de las latencias de las distintas transferencias en el bus

- CrB(MD) = L + 3 * R = 6 + 3 * 2 = 12
- CwW(MD) = 9
- CrW(MDscratch) = 2
- CwW(MDscratch) = 2
- CrW(internal_addr) = 0

4. Expresión de cálculo de los ciclos efectivos

Ciclos medios:

1. Fallo de lectura en MD: 1,5 + CrB(MD) = 1,5 + 12 = 13,5
2. Acierto de lectura en MD: 0
3. Fallo de escritura en MD: 1,5 + CrB(MD) + CwW(MD) = 1,5 + 12 + 9 = 22,5
4. Acierto en escritura en MD: 1, 5 + CwW(MD) = 1,5 + 9 = 10,5
5. Lectura de MDScratch: 1,5 + CrW(MDScratch) = 1,5 + 2 = 3,5
6. Escritura de MDScratch: 1,5 + CrW(MDScratch) = 1,5 + 2 = 3,5
7. Lectura registro interno de MC: 0

Ciclos medios por acceso a memoria, en el caso de que la probabilidad de que se produzca cada tipo de acceso a memoria sea el mismo para todos los tipos:

(13,5 + 0 + 22,5 + 10,5 + 3,5 + 3,5 + 0) / 7 = 7,643 ciclos

5. Programas de prueba

En cuanto a las diferentes situaciones de la máquina de estados de la cache, hemos comprobado que funcionaban correctamente mientras depuramos la máquina de estados, a partir del banco de pruebas que se nos daba en los fuentes del proyecto (*testbench_MD_mas_MC*).

Además de estas pruebas, hemos comprobado la integración correcta de la memoria cache como memoria del procesador MIPS, primero mediante un programa de prueba que probaba que la memoria cache se comportaba como debía cada una de las distintas situaciones posibles, como se explica en la siguiente tabla. Se han comprobado todas las transiciones de nuestra máquina de estados, también que si hay una dependencia y en medio hay un acceso a memoria lento que paraliza el procesador se complete correctamente mediante los atajos que controla la unidad de anticipación. El código y la memoria de datos se encuentran explicados en el Anexo.

Para terminar de asegurarnos de que la memoria caché funciona como debería, hemos realizado las distintas pruebas del *Proyecto_1*, que han funcionado como esperábamos.

6. Ejemplo de código

Calculamos el speed-up de nuestro procesador cuando ejecuta nuestro código de prueba con respecto al tiempo que tardaría si no hubiese cache y MD scratch, es decir, que todos los datos se almacenarán en la MD de este proyecto.

El tiempo que tarda el procesador del proyecto con la cache y la MD scratch en ejecutar nuestro código es de 272 ciclos.

El cálculo de cuánto tardaría nuestro código con el mismo procesador, pero solo con la MD. Lo hacemos teóricamente sustituyendo los valores de acceso a memoria de nuestro sistema por el valor que tendrían.

Los nuevos tiempos serían:

- Lectura: (corresponde al tiempo R de CrB(MD) de nuestro sistema) 6 ciclos
- Escritura: (corresponde a CwW(MD) 9 ciclos
- Conservamos la lectura inmediata del registro ADDR_Error_Reg

Para el cálculo se asume que el tiempo de acceso al bus es 1,5 ciclos. Sustituimos cada una de las lecturas y escrituras por estos nuevos valores, y el resultado es: 204 ciclos.

Así el *speedup* es 0.75, es decir, que va más lento al añadir la cache, la memoria scratch y el bus que conecta la cache con las memorias. Sin embargo, nuestro código no es un ejemplo realista de código, ya que lo hemos creado para comprobar el funcionamiento de la cache y su unidad de control y que ocasiona un gran número de fallos de lectura o escritura en la memoria, ya que uno de sus propósitos es comprobar la correcta gestión de las diferentes vías por parte de esta. Teniendo esto en cuenta, pensamos que en un código real, en el que se accediera a direcciones de memoria consecutivas, hubiéramos contado con un *speedup* bastante bueno que evidenciase las ventajas de la implementación de este tipo de memorias.

7. Cuantificación de horas dedicadas

- Estudio del nuevo sistema de memoria de datos…: Inés 5h30, Álvaro 5h
- Diseño de la máquina de estados: Inés 8h, Álvaro 2h
- Depurar máquina de estados mediante los tests: Inés 6h, Álvaro 6h
- Integración mips: Inés 1h, Alvaro 2h
- Depuración, verificación y programas de prueba: Ines 2h, Alvaro 5h
- Memoria: Inés 5h30, Álvaro 3h30

8. Conclusiones y Autoevaluación

La realización de este proyecto nos ha permitido consolidar nuestros conocimientos sobre las memorias caches y los protocolos de buses semi-síncronos.

El trabajo principal del proyecto ha sido realizar el autómata de estados de la unidad de control de la cache. Al principio ha resultado bastante complicado, ya que tiene muchas señales y era difícil saber para qué servía cada una. Integrar el nuevo sistema de memoria en nuestro procesador nos ha llevado muy poco tiempo y pocos cambios en el código del mips.

Realizar el código de pruebas para probar el funcionamiento de todos los conjuntos y vías ha llevado un rato aunque no ha supuesto una tarea demasiado difícil.

Nos habría gustado tener más tiempo para probar con un código con finalidad el verdadero speed-up que proporciona este sistema de memoria y también probar las mejoras opcionales del proyecto.

Como conclusión, consideramos que este proyecto nos ha ayudado a continuar interiorizando el funcionamiento interno de un procesador moderno, que ya trabajamos en el proyecto anterior. Además de esto, hemos aprendido el funcionamiento interno de un subsistema de memoria de datos, experimentado cómo trabajar con memorias “no teóricas”(con retardos) afecta negativamente al rendimiento de un procesador, y la importancia de una jerarquía de memoria bien diseñada a la hora de de diseñar sistemas “reales”.

En cuanto a la autoevaluación, valoramos que hemos superado satisfactoriamente el objetivo de este proyecto, ya que poder experimentar y visualizar las señales de una memoria y un procesador similares a los ya vistos en clase nos ha ayudado a interiorizar y entender su funcionamiento interno.

Anexo: Programa para probar el correcto funcionamiento

Para que el test funcione correctamente, es necesario usar nuestra versión del fichero *testbench* modificado, ya que al ser esta memoria más lenta que la del proyecto 1, no puede completarse antes de que la señal IRQ se ponga a ‘1’ continuamente.

MEMORIA DATOS

Dirección	Palabra	Dato (decimal si no indica lo contrario)
0	0	0
4	1	1
8	2	2
12	3	3
16	4	X10000000
20	5	0
24	6	X01000000
28	7	X00001000
36	9	0
40	10	0
64	16	16
80	20	20
112	28	28
116	29	29
128	32	32
144	36	36
148	37	37
192	48	48
224	56	56
228	57	57

MEMORIA INSTRUCCIONES

comentario	ensamblador	hexadecimal
RESET	BEQ R0, R0, 3	10210003
IRQ	RTE	20000000
Dabort	BEQ R0, R0, 38	10210026
Undef	BEQ R0, R0, 38	1000FFFF

//PRUEBA CONJUNTO 0 + PRUEBA VIAS + LECTURA

INICIO:

OP	tipo	conj/vía	HEX	ttccwwbb	comentario
LW R0, 4(R31)	Fallo lectura	0/0	0BE00004	0000 0100	"1"->R0
LW R1, 8(R31)	Acierto lectura	0/0	0BE10008	0000 1000	"2"->R1
LW R2, 12(R31)	Acierto lectura	0/0	0BE2000C	0000 1100	“3”->R2
LW R3, 64(R31)	Fallo lectura	0/1	0BE30040	0100 0000	"16"->R8
LW R4, 4(R31)	Acierto lectura	0/0	0BE40004	0000 1000	"1"->R0
LW R5, 128(R31)	Fallo lectura	0/1	0BE50080	1000 0000	"32"->R16
LW R3, 64(R31)	Acierto lectura	0/0	0BE30040	0100 0000	"16"->R8

//PRUEBA CONJUNTO 0 + ESCRITURA

SW R0, 4(R31)	Fallo escritura	0/1	0FE00004	0000 0100	R0->MC,MD
SW R1, 8(R31)	Acierto escritura	0/2	0FE20008	0000 0100	R1->MC,MD

//PRUEBA CONJUNTO 1

LW R0, 80(R31)	Fallo lectura	1/0	0BE00050	0101 0000	"20"->R0
LW R1, 84(R31)	Acierto lectura	1/0	0BE10054	0101 0100	"21"->R1
SW R0, 148(R31)	Fallo escritura	1/1	0FE00094	1001 0100	R0->MC,MD
SW R1, 144(R31)	Acierto escritura	1/1	0FE10090	1001 0000	R1->MC,MD

//PRUEBA CONJUNTO 2

LW R0, 224(R31)	Fallo lectura	2/0	0BE000E0	1110 0000	"56"->R0
LW R1, 228(R31)	Acierto lectura	2/0	0BE100E4	1110 0100	"57"->R1
SW R0, 36(R31)	Fallo escritura	2/1	0FE00024	0010 0100	R0->MC,MD
SW R1, 32(R31)	Acierto escritura	2/1	0FE10020	0010 0000	R1->MC,MD

//PRUEBA CONJUNTO 3

LW R0, 112(R31)	Fallo lectura	3/0	0BE00070	0111 0000	"28"->R0
LW R1, 116(R31)	Acierto lectura	3/0	0BE10074	0111 0100	"29"->R1
SW R0, 180(R31)	Fallo escritura	3/1	0FE000B4	1011 0100	R0->MC,MD

SW R1, 176(R31)	Acierto escritura	3/1	0FE100B0	1011 0000	R1->MC,MD

// SE UTILIZA DESPUÉS

LW R0, 4(R31)	Fallo lectura	0/0	0BE00004	0000 0100	"1"->R0

//PRUEBA MEMORIA SCRATCH

LW R30, 16(R31)	-	-	0BFE0010	–	X01000000 -> R30
SW R0, 4(R30)	-	-	0FC00004	–	"1"(R0)->MS
LW R6, 4(R30)	-	-	0BC60004	–	"1"-> R6

//DEPENDENCIAS DE DATOS CON ACCESO A MEMORIA FALLIDO EN MEDIO

LW R7, 20(R31)	Acierto lectura	1/0	0BE70014	0001 0100	"0"-> R7
LW R9, 224(R31)	Fallo lectura	2/0	0BE900E0	1110 0000	"56"->R9
ADD R8, R7, R0	-	-	04074000	-	R0+R7->R8, R8 = 1

//PRUEBA ERROR MEMORIA (ACCESO DIRECCIÓN ERRÓNEA)

LW R30, 24(R31)	Acierto lectura	1/0	0BFE0018	0001 1000	r30 = @addr_error_reg
LW R29, 28(R31)	Acierto lectura	1/0	0BFD001C	0001 1100	r29 = 00001000
LW R28, 0(R29)	-	-	0BBC0000	-	data_abort
BEQ R0, R0, -1	-	-	1000FFFF	-	-HALT

//RUTINA DATA_ABORT

ABORT	LW R9, 0(R30)	0BC90000
	RTE	20000000