Proyecto 1 – 2024 – Gestión de riesgos y excepciones en MIPS

Autores: Mateo Lorente, Diego — 873338

Solana Melero, Carlos — 872815

miércoles, 6 de marzo de 2024

Breve resumen

En este Proyecto 1 del curso 2023/2024 hemos abordado la gestión de riesgos y excepciones en MIPS utilizado en clase y en las prácticas de la asignatura. Hemos implementado nuevas funcionalidades como instrucciones JAL y RET, anticipación de operandos, detección y manejo de riesgos de datos y control, cambio a modo excepción y retorno a modo usuario RTE, además de contadores para registrar instrucciones ejecutadas y bloqueos. Las instrucciones JAL y RET han sido implementadas de manera similar a las de la práctica 3. En ellas permitimos el uso de pseudo-subrutinas, cambiando la ruta de datos, y la unidad de detección y anticipación. Para la realización del proyecto, hemos utilizado los vídeos y presentaciones proporcionados por la asignatura en caso de haber alguna duda con respecto a la teoría, así como las guías ya incluidas en el código que nos guiaban en todos los apartados que debían realizarse en MIPS. La parte del retorno a modo usuario RTE para volver de una interrupción lo dejamos para el final, justo antes de empezar con la depuración de las pruebas proporcionadas, la creación de pruebas propias y la escritura de esta memoria del proyecto. Este proyecto ha mejorado nuestra comprensión de la arquitectura MIPS, VHDL y la gestión de excepciones.

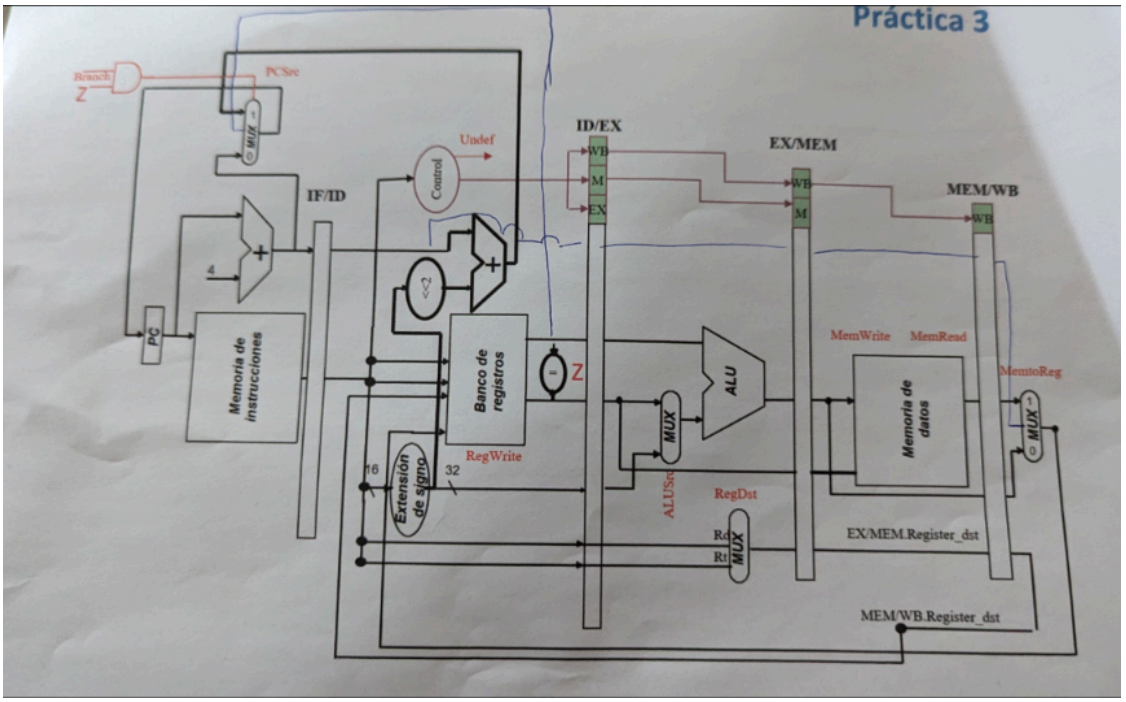
RF1 JAL y RET

Las 2 instrucciones a añadir son las que ya hemos visto en la prácticas, el JAL tiene que guardar la dirección del pc + 4 actual en el registro rt, luego debe saltar a la dirección indicada por el inmediato sumándole el pc + 4 actual, en teoría se usa para saltar a una subrutina simplificada . Para volver de dicha subrutina hemos implementado la operación ret, dicha operación carga en el pc la dirección guardada en el registro rs, se supone que se carga una dirección previamente guardada por un jal para retornar de la subrutina.

Para lograr este comportamiento hemos modificado 2 muxes de la ruta de datos y hemos ido propagando dos señales por los bancos de registros intermedios. Para el JAL necesitamos guardar la dirección del pc + 4 por lo que hemos guardado la señal de PC4_ID en uno de los puertos de extensión de palabra del banco de registros ID/EX, luego hemos ido propagando esa señal por los bancos de de EX/MEM Y MEM/WB mediante las señales PC4_EX, PC4_MEM y PC4_WB, hasta llegar al mux de escritura en WB y ahí lo hemos añadido a una de las entradas libres del mux para poder escribir en un registro el PC4. Para seleccionar qué entrada del mux seleccionar también hemos propagado la señal JAL_ID para saber que estamos en un JAL, la hemos propagado de la misma manera que el PC4, pero esta vez hemos usado uno de los puertos libres de bit. La manera que tenemos de seleccionar una entrada en el mux es poniendo el primer bit a 1 si estamos en un JAL_WB, y el segundo bit a 1 si estamos en un MEMTOREG, como nunca podremos tener estas señales juntas debido a la manera que hemos programado la unidad de control tendremos un 10 en un jal, un 01 al cargar de memoria y un 00 al actualizar un registro.

Para el RET hemos llevado la señal del bus_A, que es el contenido del registro rs, y lo hemos puesto como entrada en el mux del pc para que seleccione esa entrada si nos encontramos en un RET_ID

Aqui tenemos un diagrama de las señales añadidas.



Para comprobar su correcto funcionamiento hemos usado aparte de la prueba 1 proporcionada lo hemos probado también nuestro código de la práctica 3 detallado a continuación.

jal r8, 8	// Guardo PC+4 en r8 y salto a 0x24	14080008
add r2,r3,r2	// r2=r3+r2	04431000
add r0,r2,r0	// r0=r2+r0	04020000
add r0,r4,r0	// r0=r4+r0	04040000
add r0,r5,r0	// r0=r5+r0	04050000
add r0,r6,r0	// r0=r6+r0	04060000
add r0,r7,r0	// r0=r7+r0	04070000
sw r0,0(r1)	// mem(0)=r0	0C200000
beq r0,r0,65535	// Bucle infinito	1000FFFF
ld r2 0(r1)	// r2=mem(0)	08020000
ld r3 4(r1)	// r3=mem(4)	08030004
ld r4 8(r1)	// r4=mem(8)	08040008
ld r5 12(r1)	// r5=mem(12)	0805000C
ld r6 16(r1)	// r6=mem(16)	08060010
ld r7 20(r1)	// r7=mem(20)	08070014
ret r8	// Salto a 0x4	19000000

RF2 Anticipación de operandos

El objetivo de la anticipación de operandos es para no tener que esperar varios ciclos a que un dato que vamos a usar esté cargado en un registro si lo acabamos de utilizar, para ello hacemos que la salida de la ALU o la memoria de datos alimente la siguiente operación, además de la memoria. Tenemos 2 muxes para elegir la entrada de la alu por ejemplo MUX_ctrl_A vale 01 si anticipamos un dato que sale de la ALU, 10 si es un dato que sale de memoria o 00 si no anticipamos, de igual manera para el MUX_ctrl_B. Para elegir las entradas del mux hemos puesto las siguientes condiciones: para detectar que podemos anticipar en la etapa de memoria vemos a ver si el registro rs o rt son iguales que el registro que se va a escribir, si tenemos una instrucción válida y si vamos a escribir el resultado en un registro. Es de la misma manera para Rs y Rt solo cambiando las señales y para la etapa de memoria y de writeback. El jal también lo anticipamos debido a que es un salto incondicional y sabemos exactamente a qué dirección saltará, además de no requerir el valor de ningún registro para su ejecución.

Verificación:

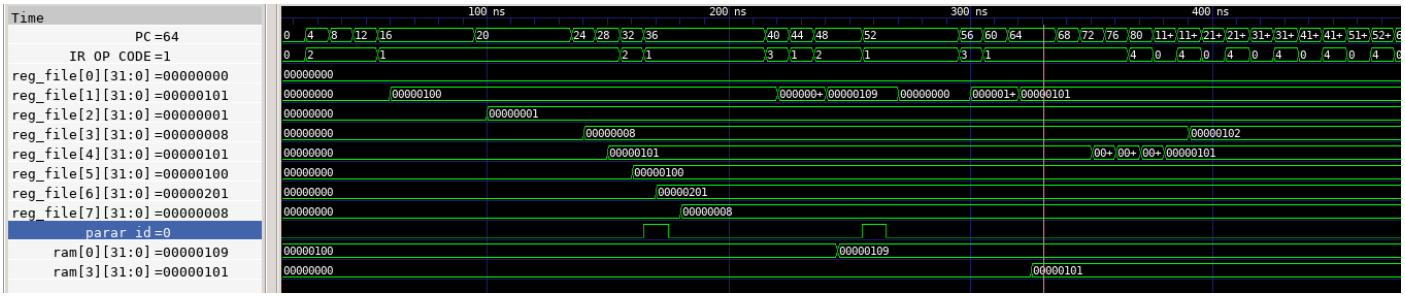
- En el banco de pruebas test_IRQ se realizan las siguientes anticipaciones:
  - De `LW R1, 8(R0)` a `ADD r31, R1, R31`: anticipación de Rs distancia 2 (tras detención de un ciclo del ADD)
  - De `SUB r31, R31,R1` a `LW R1, 0(R31)`: anticipación de Rs distancia 1.
  - `De ADD R2, R1, R2 a SW R2, 7008(R6)`
- En el banco de pruebas 6, hemos probado el resto de casos posibles de anticipación de operandos que no han sido probados en las 5 primeras pruebas. El código de la prueba es el siguiente:

```
lw r1,0(r0)     // r1=0x100          08010000
lw r2,4(r0)     // r2=0x1            08020004
lw r3,8(r0)     // r3=0x8            08030008
add r4,r1,r2    // r4=0x101          04222000
sub r5,r4,r2    // r5=0x100          04822801
add r6,r5,r4    // r6=0x201          04A43000
add r7,r7,r3    // r7=0x8            04E33800
lw r1,0(r7)     // r1=0x8            08E10000
add r1,r1,r4    // r1=0x109          04810800
sw r1,0(r0)     // mem(0)=0x109      0C010000
sub r1,r1,r1    // r1=0              04210801
lw r1,0(r1)     // r1=0x109          08210000
```

sub r1,r1,r7	// r1=0x101	04270801
sw r1,12(r0)	// mem(12)=0x101	0C01000C
add r4,r4,r4	// r4=0x202	04842000
sub r4,r4,r7	// r4=0x1FA	04872001
sub r4,r4,r1	// r4 = 0xF9	04812001
add r4,r4,r3	// r4=0x101	04832000
add r3,r4,r2	// r3=0x102	04821800
beq r0,r0,-1	// Bucle infinito	100000FF

En esta prueba, probamos los siguientes casos de anticipación:

- Anticipación de operandos al hacer add r4,r1,r2 y después sub r5,r4,r2.
- Anticipación de operandos al hacer sub r5,r4,r2 y después add r6,r5,r4.
- Anticipación de operandos al hacer add r7,r7,r3 y después lw r1,0(r7).
- No hay anticipación de operandos al hacer add lw r1,0(r7) y después add r1,r1,r4. Esto se debe a que r1 estará actualizado después del acceso a memoria, y no hay cortos en ese punto hasta el siguiente ciclo, por lo que se deberá parar el FETCH y DECODE 1 ciclo, aunque esto lo explicaremos más adelante.
- Anticipación de operandos al hacer add r1,r1,r4 y después sw r1,0(r0).
- Anticipación de operandos al hacer sub r1,r1,r1 y después lw r1,0(r1).
- No hay anticipación de operandos al hacer lw r1,0(r1) y después sub r1,r1,r7 por la razón mencionada anteriormente.
- Anticipación de operandos al hacer sub r1,r1,r7 y después sw r1,12(r0).
- Anticipación de operandos al hacer sub r4,r4,r7 y después sub r4,r4,r1.
- Anticipación de operandos al hacer add r4,r4,r3 y después add r3,r4,r2.



RF3 Riesgos de datos

Para los riesgos de datos de datos primero detectamos si hay dependencias entre los operandos, por ejemplo en dep_rs_EX señalamos si hay una dependencia entre la instrucción en ejecución (EX) y la instrucción en la etapa de decodificación (ID) que escribe en el registro "rs". Utiliza las señales valid_I_EX, valid_I_ID (Para saber si son instrucciones válidas), Reg_Rs_ID = RW_EX (Para ver si el registro Rs usa el registro que en el que vamos a escribir) , RegWrite_EX (Para saber si vamos a escribir en registro), también excluimos las instrucciones JAL, RTE y NOP ya que no usan rs. Para le etapa de memoria es igual pero fijándonos en el registro RW_MEM

Para las dependencias en rt tenemos dep_rt_EX queetecta si existe una dependencia entre la instrucción en ejecución (EX) y la instrucción en ID que escribe en el registro "rt". Utiliza las señales valid_I_EX para verificar si la instrucción en EX es válida, valid_I_ID para la validez de la instrucción en ID, Reg_Rt_ID = RW_EX para verificar si el registro Rt en ID es igual al que se va a escribir en EX, y RegWrite_EX para determinar si se va a escribir en un registro en EX. Se excluyen las instrucciones NOP, LW y RTE ya que no hacen uso de rt, RET tampoco hace uso de rt pero nos limitamos a no usar rt más tarde.

Riesgos de Datos:

Una vez que ya tenemos las dependencias pasamos a ver los riesgos para cada instrucción que son los siguientes:

- lw_uso: Se produce si hay una dependencia y vamos a leer de memoria. Depende de dep_rs_EX y MemRead_EX.
- BEQ: Indica un riesgo si hay dependencias y la instrucción en ID es una instrucción de salto condicional (BEQ). Utiliza IR_op_code, dep_rs_EX y dep_rs_Mem.
- RET: Similar al riesgo BEQ pero para el RET, se limita a considerar la dependencia en el registro "rs". Se basa en IR_op_code, dep_rs_EX y dep_rs_Mem.

Para el jal uso hemos decidido no parar el procesador

La señal riesgo de datos_ID en caso de detectar cualquier riesgo y paramos el procesador si la instrucción en ID es válida.

Si la memoria no está preparada paramos el procesador entero hasta que lo esté mediante Mem_ready.

Verificación:

- En el banco de pruebas test_IRQ se realizan las siguientes detenciones:
  - Test 1: 1 ciclo de detención por dependencia a distancia 2 entre `ADD R1, R1, R1` y `beq R1, R1, main.`
  - Test 2: 1 ciclo de detención por dependencia lw-uso en varios casos. Por ejemplo: `de LW R1, 8(R0) a SW r1, 7004(r6)`

- Test 3: No se para por riesgo de datos entre las instrucciones `SW r1, 7004(r6)` y `ADD R1, R1, R1.`

- Hemos realizado 2 pruebas extra para probar todos los casos de riesgo de datos y de control que no han sido probados en las 5 primeras pruebas. Dichas pruebas son las siguientes:

  - El banco de pruebas 7 prueba casi todos los casos los casos de riesgo de datos y control restantes. El código de dicha prueba es el siguiente:

```
        lw r1,0(r0)      // r1=mem(0)=0x100      0x08010000
        beq r1,r1,1      // Salto a 0xC          0x10210001
        NOP
        lw r2,4(r0)      // r2=mem(4)=0x1        0x08020004
        beq r2,r1,1      // No salta             0x10410001
        sub r3,r1,r2     // r3=0xFF              0x04221801
0x18:   beq r3,r3,38     // Salto a 0xB4         0x10630026

0x48:   lw r2,8(r0)      // r2=0x8               0x08020008
        lw r3,0(r2)      // r3=0x8               0x08030000
        beq r0,r0,-1     // Bucle infinito  0x100000FF

0xB4:   sw r3,12(r0)     // mem(12)=0xFF         0x0C03000C
        lw r4,12(r0)     // r4=0xFF              0x0804000C
        add r5,r4,r4     // r5=0x1FE             0x04842800
        lw r6,0(r0)      // r6=0x100             0x08060000
        lw r7,16(r0)     // r7=0xC               0x08070010
        add r7,r7,r7     // r7=0x18              0x04E73800
        add r8,r7,r7     // r8=0x30              0x04E74000
        add r8,r8,r8     // r8=0x60              0x05084000
        sub r8,r8,r7     // r8=0x48              0x05074001
        RET r8           // Salto a 0x48         0x19000000
```

    En esta prueba, probamos los siguientes casos de riesgo de datos y de control:

    - Detención de 2 ciclos por riesgo de control entre las instrucciones lw r1,0(r0) y beq r1,r1,1.

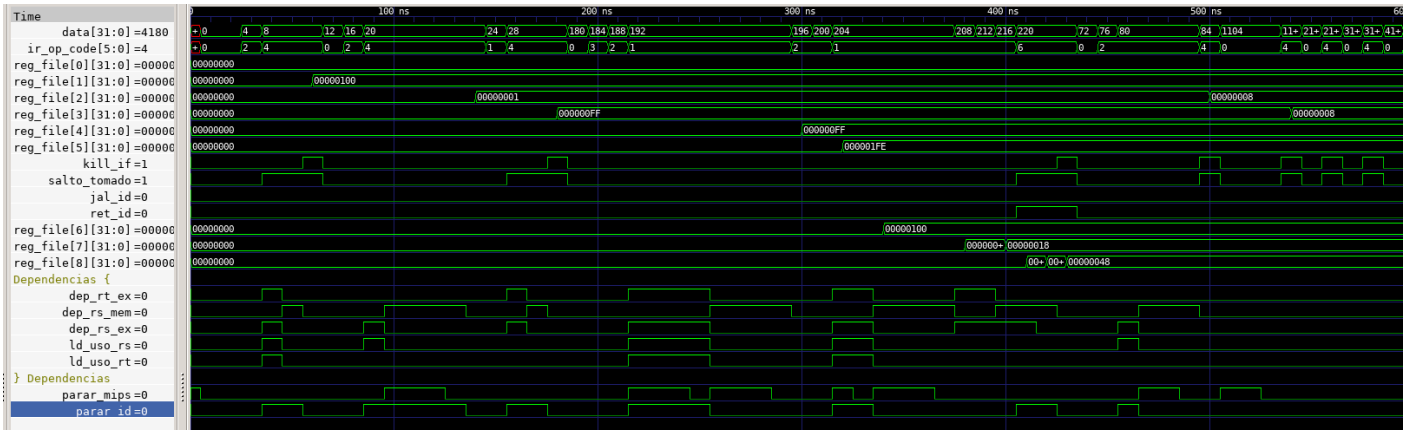
- Detención de 2 ciclos por riesgo de control entre las instrucciones sub r3,r1,r2 y beq r3,r3,38.

- No hay ningún riesgo de datos entre las instrucciones sw r3,12(r0) y lw r4,12(r0), ya que cuando el lw llega a la etapa MEM, el dato introducido ya está en memoria, y la memoria está lista para ser leída.

- Detención de 1 ciclo por riesgo de datos entre las instrucciones lw r4,12(r0) y add r5,r4,r4.

- Detención de 3 ciclos por parada en memoria entre las instrucciones lw r6,0(r0) y lw r7,16(r0).

- Detención de 2 ciclos por riesgo de control entre las instrucciones sub r8,r8,r7 y RET r8.



- El banco de pruebas 8 prueba los casos que tampoco se han probado en el banco de pruebas 7 por dividir todas las pruebas en 2 programas distintos. El programa contiene el siguiente código:

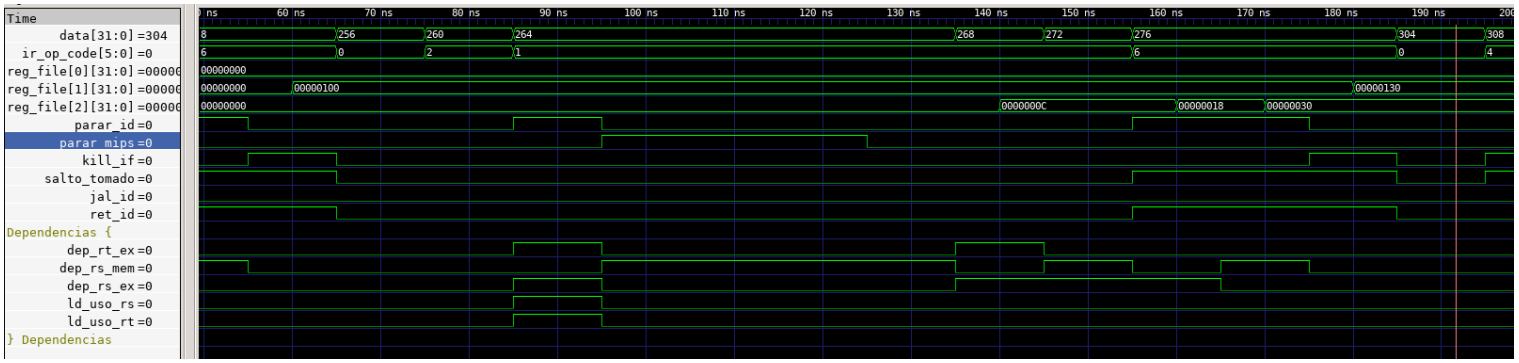
```
        lw r1,0(r0)      // r1=0x100          08010000
        RET r1           // Salto a 0x100     18200000

0x100:  lw r2,16(r0)     // r2=0xC            08020010
        add r2,r2,r2     // r2=0x18           04421000
        add r2,r2,r2     // r2=0x30           04421000
        add r1,r1,r2     // r1=0x130          04410800
        RET r1           // Salto a 0x130     18200000

0x130:  beq r0,r0,-1     // Bucle infinito    100000FF
```

  En esta prueba, probamos los siguientes casos de riesgo de control:

  - Detención de 2 ciclos por riesgo de control entre las instrucciones lw r1,0(r0) y RET r1.
  - Detención de 2 ciclos por riesgo de control entre las instrucciones add r1,r1,r2 y RET r1.



RF4 Riesgos de control

Los riesgos de control surgen cuándo tomamos un salto y tenemos que cancelar la instrucción que ha entrado en fetch para que no se ejecute. Para ello tenemos que la señal kill if que eliminará esta instrucción, dicha señal se activa cuando salto tomado es 1, la instrucción en ID (el salto) es válida y no existe ningún riesgo, para que en caso de que no estén sus operandos preparados espere hasta que estén disponibles y cuando lo estén toma la decisión

Verificación:

- En el banco de pruebas test_IRQ se realizan varios saltos tomados en los que sólo se ejecutan las instrucciones adecuadas:
    - `beq R1, R1, main`
    - `rte`
- Un ejemplo donde hay riesgo de control, pero no se salta es en el banco de pruebas 7. Esto ocurre en las instrucciones lw r2,4(r0) y beq r2,r1,1

- En el caso de JAL, no hay ningún riesgo de control, ya que la instrucción simplemente escribe en un registro y salta a una dirección marcada por el inmediato que codifique dicha instrucción, por lo que no tiene necesidad de realizar ninguna parada. Para el caso de RET, tanto en el banco de pruebas 7 como en el 8 hay casos de riesgo de control. Estos casos han sido explicados en el apartado RF3. Para verificar el correcto funcionamiento del JAL y del RET, aunque el RET ha sido probado en pruebas anteriores, hemos realizado un noveno banco de pruebas. Es importante decir que esta prueba, a diferencia del resto, utiliza una RAM de datos distinta, que es la segunda en el archivo IO_MD_subsystem.vhd. El código de la prueba es el siguiente:

  jal r8, 8 // Guardo PC+4 en r8 y salto a 0x24 14080008

  add r2,r3,r2 // r2=r3+r2 04431000

  add r0,r2,r0 // r0=r2+r0 04020000

  add r0,r4,r0 // r0=r4+r0 04040000

  add r0,r5,r0 // r0=r5+r0 04050000

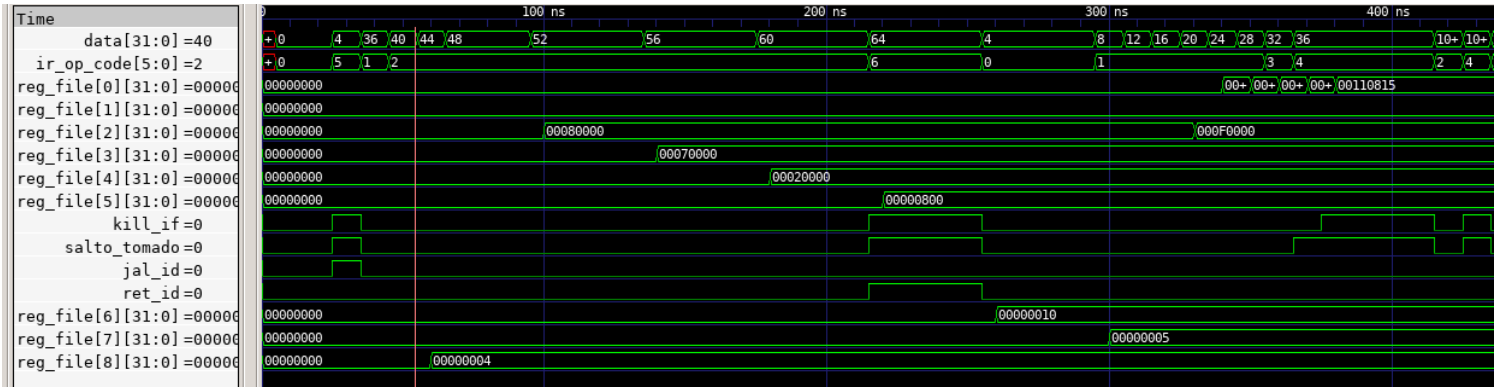
  add r0,r6,r0 // r0=r6+r0 04060000

  add r0,r7,r0 // r0=r7+r0 04070000

  sw r0,0(r1) // mem(0)=r0 0C200000

  beq r0,r0,-1 // Bucle infinito 1000FFFF

ld r2 0(r1)	// r2=mem(0)	08020000
ld r3 4(r1)	// r3=mem(4)	08030004
ld r4 8(r1)	// r4=mem(8)	08040008
ld r5 12(r1)	// r5=mem(12)	0805000C
ld r6 16(r1)	// r6=mem(16)	08060010
ld r7 20(r1)	// r7=mem(20)	08070014
ret r8	// Salto a 0x4	19000000



RF5 cambio a modo excepción

Si el procesador está en modo usuario y recibe IRQ, ABORT o UNDEF pasa al modo correspondiente y ejecuta la rutina que indica la tabla de vectores de excepción. Si está en modo excepción ignora las señales hasta volver a modo usuario.

Para seleccionar el que vamos a cargar en el pc tenemos un mux que lo decide. Si hemos aceptado la excepción cargamos la dirección correspondiente al tipo que sea necesario. Continuación de la Ejecución: Cuando ocurre una excepción, las instrucciones en las etapas de memoria (Mem) y escritura de registros (WB) deben continuar su ejecución, mientras que el resto de las etapas se detienen.

Para detectar la ocurrencia de una excepción. Si alguna de las señales de pedir interrupción está activa y las excepciones están habilitadas (indicado por el bit 1 del registro MIPS_status), y el procesador no está en estado de parada (parar_MIPS = '0'), entonces se acepta la excepción.

Además, se determina cuándo actualizar el registro de estado del procesador. Esto ocurre cuando el procesador no está parado y se detecta una instrucción de retorno (RTE_ID = '1') o se acepta una excepción. En estos casos, se activa la señal de actualización update_status.

Finalmente, se utiliza un multiplexor para seleccionar la entrada del registro de estado. Cuando se acepta una excepción internamente, se selecciona una entrada que indica que las excepciones están deshabilitadas y el procesador está en modo de excepción, sino se selecciona una entrada que indica que las excepciones están habilitadas y el procesador está en su estado normal.

Verificación:

- IRQ: banco de pruebas test_IRQ en el que se realizan diversas IRQs. Todas ellas se atienden si y sólo si el procesador está en modo usuario. Se ejecuta una RTI que contabiliza el número de IRQs. Se verifica su funcionamiento correcto.
- Data Abort: en la memoria de instrucciones se incluyen dos bancos de pruebas en el que se realizan un acceso no alineado, y un acceso a una dirección fuera de rango. En ambos casos la ejecución salta a la rutina Data Abort (bucle infinito).
- Undef: en la memoria de instrucciones se incluye un banco de pruebas en el que se ejecuta una instrucción con un código de instrucción desconocido. La ejecución salta a la rutina UNDEF (bucle infinito).

RF6 retorno a modo usuario RTE

Después de manejar la excepción, se debe elegir la siguiente instrucción válida para ejecutar. Esto se hace a través de un multiplexor que selecciona entre la dirección de la próxima instrucción en las etapas de excepción (EX) e identificación (ID), o la dirección del contador de programa (PC) si no hay instrucciones válidas disponibles. Además, si la instrucción en la etapa EX es una instrucción de retorno (RTE), no se elige como la siguiente instrucción válida, ya que el retorno se realiza en la etapa ID y la instrucción RTE en EX ya ha completado su ejecución.

La dirección a la que debe retornar después de manejar la excepción se almacena en un registro llamado Exception_LR. Esta dirección se selecciona usando el multiplexor anterior y se carga en este registro cuando se acepta la excepción (Exception_accepted_internal).

Luego en el propio mips al llegar la instrucción RTE se carga

Verificación: Sólo tenéis que verificarlo para las IRQs. El código debe funcionar aunque se interrumpa repetidas veces. Para el resto de excepciones se asume que es un fallo irresoluble y se debe resetear el sistema, pero la gestión es idéntica que para las IRQs. En el banco de pruebas test_IRQ se puede comprobar que se retorna a la ejecución sin alterarla siempre que se siga un esquema de prólogo y epílogo apropiado (guardando y restaurando los registros en pila). Para ello se utiliza el registro 31 como SP.

RF7 contadores

Aumentamos en 1 el contador de instrucciones ejecutadas si valid_I_WB es '1' y el mips no está parado, hemos elegido la señal de valid_I_WB porque una vez que una instrucción está en esa etapa ya no puede ser eliminada ni detenida de ninguna manera, solo tenemos que comprobar que el mips no está detenido para evitar contar varias veces la misma instrucción.

Incrementamos en 1 el contador de bloqueos de datos cuando `Kill_IF` es '1'. Verificamos que el procesador no esté completamente detenido para distinguir las paradas de datos de las relacionadas con la memoria, también hacemos lo mismo con las paradas de datos.

Cada vez que aceptamos una excepción (Exception_accepted) aumentamos el contador de excepciones. Ya que solo si la hemos aceptado se va a procesar

Para las paradas de memoria aumentamos el contador cada vez que paramos el mips completamente, que indica que estamos esperando a la memoria.

Verificación: En el banco de pruebas test_IRQ se ha comprobado que:

- Por cada instrucción válida que hace su etapa WB se incrementa el contador de instrucciones Ins. Si la instrucción no es válida no se incrementa.
- Data_stalls contabilizan bien los ciclos debidos a los riesgos de datos incrementando el número indicado en los test mencionados en el RF5.
- Control_stalls contabilizan bien los ciclos debidos a los riesgos de control, incrementando un ciclo en cada salto tomado (ver pruebas en el RF6).
- Exception_accepted contabiliza las excepciones aceptadas. Su cuenta coincide con la cuenta que realiza el propio código.
- Memory_stalls: cada vez que se pide una operación a memoria, y esta no esta “ready” se incrementa este contador.
- IMPORTANTE: las paradas deben incrementar sólo un contador. Si hay hay varios motivos, se cuenta el de mayor jerarquía : memory stall> data_stall> control_stall. Ya que el otro se mantendrá y se contabilizará cuando toque.

Cuantificación de horas dedicadas

- Estudio del MIPS, VHDL, entorno, instalación…:
  - Carlos: 1
  - Diego: 1
- Adición de las nuevas instrucciones:
  - Carlos:4
  - Diego: 1
- Gestión de excepciones:
  - Carlos: 3
  - Diego: 4

- Gestión de riesgos:
    - Carlos: 8
    - Diego: 9
- Depuración, verificación y programas de prueba:
    - Carlos: 9
    - Diego: 15
- Memoria:
    - Carlos: 4
    - Diego: 1

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

En general el trabajo nos ha parecido interesante y hemos profundizado ampliamente nuestro conocimiento sobre el MIPS. Ahora tenemos mucho más claras sus partes, así como varias técnicas para optimizarlo y evitar errores. La única pega que podríamos sacar al proyecto sería sobre la parte de las excepciones, ya que apenas las hemos visto en clase, por lo que hemos notado cierta disparidad entre los contenidos de clase y el trabajo. Pese a ello nos ha gustado el trabajo. La calificación que nos otorgamos es de un 9 ya que no solo funcionan las pruebas proporcionadas, sino que también hemos probado extensamente las capacidades del procesador y hemos verificado su correcto funcionamiento.

Agradecimientos

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

- Juan José Serrano Mora (870282)
- Emilliano Recuenco López (868419)
- Raúl Soler Fernández (875478)
- Daniel Simón Gayán (870984)
- Adrián Navarro Marín (874912)