Proyecto 1 – 2023 – Gestión de riesgos y excepciones en MIPS

Álvaro Ramos Fustero, Inés Román Gracia

jueves, 26 de marzo de 2023

Breve resumen

El proyecto consiste en la implementación de la gestión de los riesgos de control y de datos del MIPS visto en clase y en el diseño hardware de la modificación del procesador para añadir dos nuevos tipos de instrucción, tres tipos de interrupciones y un nuevo modo de funcionamiento para tratar con ellas. El objetivo es verificar el correcto funcionamiento de estas características mediante diferentes casos de prueba.

Se describen los siguientes casos de prueba en los programas que se ejecutan para verificar el correcto funcionamiento que estas características dan al procesador: el cambio a modo excepción, el retorno a modo normal, la escritura del registro de salida del procesador, la anticipación de operandos, la detención de la ejecución para evitar riesgos de datos y la gestión de contadores.

Para cada caso de prueba se describe el objetivo, la implementación de la característica que se ha implementado en el procesador, y cómo se ha llevado a cabo su verificación mediante bancos de prueba específicos.

Además, nos hemos autoevaluado exponiendo nuestras dificultades a la hora de realizar el trabajo y qué partes nos han resultado más complejas, y cuales más sencillas, y hemos cuantificado el tiempo que hemos invertido en realizar este trabajo.

Verificación de los requisitos funcionales:

RF1 cambio a modo excepción

Descripción: 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.

¿**Cómo**? Cuando llega una señal externa de IRQ, o una interna de *Data_abort* o undef, y el *status_register* está configurado para aceptar excepciones, la señal *Exception_accepted* se activa. Esta señal es la que carga el registro que almacena la dirección de retorno de la IRQ o salto, y cambia el estado del *status_register* para indicar que excepción ha sido aceptada, además de cargar el PC al siguiente ciclo de reloj con los datos que se encuentran en *PC_in*, correspondientes a la excepción que ha sido aceptada.

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).

RF2 retorno a modo usuario RTE

Descripción: Al terminar la excepción se vuelve a la ejecución original con la instrucción RTE

¿**Cómo**? Al llegar la instrucción de RTE a la etapa ID, llega el código de la operación RTE a la UC, que genera la señal *RTE_ID*. Esta señal actualiza el valor de *Status Register* con el siguiente ciclo, además de colocar la dirección de retorno que se encuentra en *Exception_LR_Output* en PC_IN, a través de MUX_PC_IN y de *valid_I_ID*. Esto hará que el salto se haga efectivo con el siguiente ciclo de reloj.

Verificación: Sólo se ha realizado para IRQs. 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.
- En el banco de pruebas test_UA la única instrucción en la rutina de la interrupción IRQ es una RTE. Se puede comprobar por los resultados finales de los registros que el PC toma el valor adecuado al ejecutar esta instrucción.
- En el banco de pruebas test_UD, al igual que en el anterior, la única instrucción en la rutina de la interrupción IRQ es una RTE. Se puede comprobar por los resultados finales de los registros que el PC toma el valor adecuado al ejecutar esta instrucción.

RF3 WRO

Descripción: la instrucción WRO permite escribir el contenido de un registro en la salida del MIPs

¿**Cómo**? En la etapa ID, la UC genera la señal *Write_output*, que va directamente al load de *output_register*, lo que hace que se escriban en este los datos de *BusA* en el siguiente ciclo de reloj.

Verificación:

- En el banco de pruebas test_IRQ se realizan dos WROs (`WRO R31, WRO R2`).
- En el banco de pruebas test_UD se realiza una instrucción WRO (`WRO R3`).

RF4 Anticipación de operandos

Descripción: El procesador es capaz de anticipar los operandos para las instrucciones LW, SW y ARIT a distancia 1 (siempre que el dato a anticipar no venga de un lw) y 2.

¿**Cómo**? Para la anticipación de operandos cuando se producen dependencias se emplean cuatro señales dependiendo de si se trata del registro *rs* o *rt* y de si se quiere anticipar este operando desde la instrucción en ciclo de memoria o de la instrucción en ciclo de escritura. Para ello, en la entrada de los operandos de la ALU habrá un bloque MUX 4:1 de 32 bits y las entradas serán: bus del banco de registros (*busA* o *busB*), *ALU_out_MEM* y la salida de la MUX del ciclo de escritura que se va a escribir en el banco de registros.

Los operandos siempre pueden ser anticipados por la instrucción en ciclo de escritura que ya tiene el resultado final en el bus de escritura del banco de registros. Además, si la instrucción en ciclo de memoria se trata de un SW o ARIT también podría anticipar operandos ya que ya tiene el resultado final del registro a la salida de la ALU, esto no es posible para la instrucción LW porque aún no se conoce el valor final que queremos almacenar en el registro hasta que no se lea la memoria.

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`
- En el banco de pruebas test_UA se realizan las siguientes anticipaciones:
  - De `ADD R0, R1, R0` a `ADD R0, R0, R0: anticipación de Rs y de Rt a distancia 1`
  - De `ADD R1, R1, R1` a `ADD R1, R1, R1: anticipación de Rs y de Rt a distancia 2`

RF5 Riesgos de datos

Descripción: El procesador es capaz de detener la ejecución para evitar los riesgos causados por las dependencias en instrucciones lw-uso (distancia 1), beq o WRO (distancias 1 o 2).

¿**Cómo**? Para gestionar un riesgo de datos, la ejecución se bloquea temporalmente cuando la señal *Parar_ID* se activa. Esta señal, en el caso de los riesgos de datos, se activa cuando la instrucción es válida y se detecta un riesgo de datos (mediante la señal *riesgo_datos_ID*). Esta señal se genera al detectar los riesgos, que se dan cuando existen dependencias de datos en las instrucciones en las que no se pueden anticipar operandos, porque no existe un bus que lleve estos datos hasta las componentes hardware que gestionan estas operaciones. Esto se produce en el caso de que se vaya a usar un registro que está siendo modificado por una instrucción LW en el ciclo de ejecución y no pueden anticipar el operando que modifican porque aún no han leído su valor final de memoria, en el caso de que se vaya a ejecutar un BEQ pero alguno de sus

registros dependa de alguna instrucción anterior o si se va a ejecutar un WRO pero el registro que va a emplear está siendo modificado por alguna instrucción anterior.

Verificación:

- En el banco de pruebas test_IRQ se realizan las siguientes detenciones:
  - Test1: 1 ciclo de detención por dependencia a distancia 2 entre `LW R31, 0(R0)` y `WRO R31`.
  - Test 2: 2 ciclos de detención por dependencia a distancia 1 entre `ADD R1, R1, R1` y `beq R1, R1, main`.
  - Test 3: 1 ciclo de detención por dependencia lw-uso en varios casos. Por ejemplo: de `LW R1, 8(R0)` a `ADD r31, R1, R31`.
  - Test 4: 2 ciclos de detención por dependencia a distancia 1 entre `ADD R2, R1, R2` y `WRO R2`.
- En el banco de pruebas test_UD se realizan las siguientes detenciones:
  - 1 ciclo de detención por dependencia a distancia 1 entre `LW R0, C(R0)` y `ADD R0, R0, R1`.
  - 2 ciclos de detención por dependencia de *r*s a distancia 1 entre `ADD R2, R2, R1` y `BEQ R2, R0, 1`.
  - 2 ciclos de detención por dependencia a distancia 1 entre `ADD R0, R0, R1` y `WRO R3`.
  - 2 ciclos de detención por dependencia de *rt* a distancia 1 entre `ADD R2, R2, R1` y `BEQ R0, R2, 1`.

RF6 Riesgos de control

Descripción: El procesador es capaz eliminar los riesgos de control causados por las instrucciones de salto: BEQ y RTE.

¿**Cómo**? Para gestionar los riesgos de control, se activa la señal de salida de la UD *Kill_IF* cuando se toma un salto y la instrucción que está en la etapa ID es válida y se ejecuta en ese ciclo (es decir que no tiene dependencias de operandos con instrucciones anteriores si se trata de un BEQ).

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`
- En ambos bancos de pruebas test_UA y test_UD sihay una instrucción RTE en la rutina de la interrupción IRQ.

- En el banco de pruebas test_UA hay 5 BEQs, de los cuales 3 siempre producen un salto. Los otros dos sirven como control de flujo en los bucles en los que se aumenta el valor del registro r2 y por lo tanto en la última iteración no se produce salto.

RF7 contadores

Descripción: los contadores nos dan información de la ejecución del código

¿**Cómo**?

- Cont_cycles (contador de ciclos): su load es siempre "1", ya que debe contar todos los ciclos que ejecute el procesador independientemente de lo que esté haciendo en ellos.
- Cont_i (contador de instrucciones ejecutadas): Su load es Valid_I_WB, ya que de esta forma cuenta todas las instrucciones que se han ejecutado (las que llegan a WB siendo válidas), sin tener en cuenta las que se han interrumpido en los saltos.
- cont_data_stalls (contador de excepciones por riesgos por datos): Su load es Parar_ID, ya que esta señal se activa cuando las excepciones lo hacen, para que se pare la etapa.
- cont_control_stalls (contador de excepciones por riesgos de control): Su load es salto tomado, por que este tipos de excepciones activan esta señal.
- Exceptions (contador de número de excepciones): Se incrementa con exception_accepted, que se activa durante la gestión de una excepción.
- Exception_cycles (contador de número de ciclos que el procesador se encuentra en modo excepción): Se incrementa cuando el registro de estado se encuentra en estado de Excepción.

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.
- Exception_cycles se actualiza cada ciclo en el que el procesador está en modo excepción.

Cuantificación de horas dedicadas

- Estudio del MIPS, VHDL, entorno, instalación…: Inés 8h, Álvaro 10h
- Adición de las nuevas instrucciones: Inés 1h, Álvaro 1h
- Gestión de excepciones: Inés 3h, Álvaro 3h
- Gestión de riesgos: Inés 3h, Álvaro 3h
- Depuración, verificación y programas de prueba: Inés 12h, Álvaro 10h
- Memoria: Inés 3h, Álvaro 3h

Conclusiones y Autoevaluación

Creemos que este trabajo nos ha sido muy útil para familiarizarnos con un lenguaje y un entorno de desarrollo de hardware profesional, más allá de lo que estamos acostumbradas a hacer en Logisim.

Antes de empezar el trabajo, nos parecía una tarea casi inasumible, por qué creíamos que íbamos a tener que escribir muchísimo código, además de depurarlo, y nos parecía que íbamos a tener que invertir una gran cantidad de tiempo por que no íbamos a saber cómo hacerlo. No obstante, cuando miramos los archivos fuentes del proyecto, nos dimos cuenta de que realmente el trabajo estaba hasta cierto punto guiado y nos tranquilizamos un poco.

Lo primero que hicimos, antes de escribir nada de código, fue dedicar un tiempo simplemente a intentar entender el funcionamiento interno del MIPS del proyecto, además de intentar identificar cuál era la función de cada señal.

Una vez entendimos que hacía cada señal y tuvimos un dibujo identificándolas, la implementación de las características que se solicitaban ha sido más fácil de lo que esperábamos inicialmente. Depurar el proyecto también nos llevó algo de tiempo, pero no fue muy complicado y el mayor error que tuvimos fue porque nos olvidamos de completar un fichero.

Realmente, en nuestra opinión y como conclusión, lo más difícil de este trabajo ha sido interiorizar el funcionamiento del procesador del proyecto, y aprender a usar un entorno de desarrollo completamente nuevo, además de acostumbrarnos a imaginarnos el hardware a partir de un lenguaje de descripción de hardware en lugar de dibujos, como hacíamos hasta ahora con Logisim.

Anexo (Programas de prueba)

Ambos códigos requieren que la memoria de datos sea la misma que la del banco de pruebas test_IRQ.

Test_UA

Este código tiene como objetivo activar todas las señales de dependencias de operandos que pueden ser resueltas mediante anticipación de operandos. Todas las señaes que se encargan de esto son activadas en algún punto del código: *Corto_A_Mem*, *Corto_B_Mem*, *Corto_A_WB* y *Corto_B_WB*.

```
LW    R1, 4(R1)             -- R1=1

ADD   R0, R1, R0            -- R0=1

ADD   R0, R0, R0            -- R0=2

ADD   R1, R1, R1            -- R1=2
```

```
ADD   R2, R0, R1          -- relleno para crear dependencia de
                             distancia 2

ADD   R1, R1, R1

BEQ   R0, R0, -1          -- FIN
```

Test_UD

Este código tiene como objetivo activar todas las señales de dependencias de operandos que son resueltas mediante la parada de instrucciones o la cancelación de instrucciones dirigidas por la unidad de detención mediante las señales: *KILL_IF*, *ld_uso_rs, ld_uso_rt*, *BEQ_rs*, *BEQ_rt* y *WRO_rs.*

```
LW    R0, C(R0)           -- R0=8

ADD   R0, R0, R1          -- produce load_uso

LW    R1, 4(R1)           -- R1=1

LW    R2, 4(R2)           -- R2=1

ADD   R2, R2, R1          -- R2=1..8

BEQ   R2, R0, 1           -- produce BEQ_rs

BEQ   R0, R0, -3

ADD   R3, R2, R1

WRO   R3                  -- produce WRO_rs

ADD   R0, R0, R1          -- R0=9

ADD   R2, R2, R1          -- R2=9

BEQ   R0, R2, 1           -- produce BEQ_rt

BEQ   R0, R0 -3

BEQ   R0, R0, -1          -- FIN
```