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

Guillermo Bajo Laborda – 842748

Álvaro de Francisco Nievas - 838819

domingo, 26 de marzo de 2023

## Breve resumen

En este proyecto se ha implementado sobre un procesador MIPS soporte para interrupciones. Cuando llega una excepción (IRQ, data abort, undefined) se ejecutará la rutina de tratamiento correspondiente, y con la nueva instrucción RTE se retomará la ejecución del programa principal. Esta instrucción, como BEQ, genera riesgos en un procesador segmentado y se deberá invalidar la instrucción en la etapa IF cuando se salte de vuelta a la ejecución principal. Se ha incorporado también la instrucción WRO, que permite escribir en un puerto de salida del procesador.

Además, se ha añadido una Unidad de Anticipación y una Unidad de Detención para gestionar los riesgos de datos y de control provocados por las dependencias entre productores y consumidores. Con ello, se evita el uso de instrucciones NOP y se gestionan las dependencias mediante el hardware. La UA detecta cuando se hace uso en una instrucción de un registro cuyo valor se modifica en otra instrucción anterior (y provoque dependencia) y transmite a la ALU los valores adecuados. La UD detecta casos, como dependencias load-uso y operaciones que utilizan los operandos en la etapa ID, que no se pueden resolver mediante anticipación y retrasa la ejecución de las instrucciones que necesitan dichos operandos.

Por último, se han añadido contadores que almacenan parámetros como el nº de instrucciones ejecutadas, el nº de paradas por riesgo de datos y otros para cuantificar el rendimiento del procesador.

## 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**?

Detectamos cuando recibimos una excepción o cuando queremos volver de una excepción y activamos la lectura del registro donde almacenamos el estado. La entrada del registro será “11” (modo excepción con excepciones deshabilitadas) cuando hayamos detectado una excepción y “00” (modo normal con excepciones habilitadas) en el caso contrario.

**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**?

Cuando ejecutamos la instrucción RTE recibimos de la UC la señal “RTE_ID”. Si además el salto se ha tomado (señal “salto_tomado”=1) cargaremos en el registro PC la dirección almacenada en Exception_LR, que habrá sido guardada al entrar en modo excepción.

**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.

## RF3 WRO

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

¿**Cómo**?

Cuando ejecutamos una instrucción WRO recibimos de la UC la señal “write_output_UC” y comprobamos que no se ha detenido la ejecución comprobando la señal “parar_ID”, que recibimos de la UD cuando hay riesgo de datos.

Si se cumplen las condiciones anteriores, activamos la señal load del registro “output_reg” y cargamos el dato que recibimos de “BusA” en el vector de salida.

**Verificación:** En el banco de pruebas test_IRQ se realizan dos WROs (WRO R31, WRO R2).

## 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**? Explicar brevemente la gestión de anticipación de operandos.

Cuando la UA detecta una dependencia entre la instrucción en fase EX y las que están en la fase MEM o WB activamos un corto que, mediante un multiplexor situado ambas entradas de la ALU, cambia el dato con el que operamos. Los multiplexores tienen como entrada el valor del registro correspondiente, el dato que utilizamos como fuente para la memoria de datos en la etapa MEM y el dato que escribimos en el BR en la etapa de WB.

Es importante destacar el caso de que detectemos una dependencia con las dos instrucciones siguientes. En ese caso, cogeremos el dato más reciente (etapa MEM).

**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

- PRUEBA DE ANTICIPACIÓN DE OPERANDOS Y DOBLE DEPENDENCIA:

| | | | | |
|---|---|---|---|---|
| RESET | @0x0 | 10210003 | beq r1, r1, INI | |
| IRQ | @0x4 | 1021003E | beq r1, r1, RTI | |
| DATA_ABORT | @0x8 | 1021005D | beq r1, r1, RT_AB | |
| UNDEF | @0xC | 1021006C | beq r1, r1, RT_UN | |
| INI | @0x10 | 08010004 | lw r1, 4(r0) | R1 = mem(4) |
| | @0x14 | 08020008 | lw r2, 8(r0) | R2 = mem(8) |
| MAIN | | | | |
| | @0x18 | 04221800 | add r3, r1, r2 | R3 = r1 + r2 |
| | @0x1C | 04431801 | sub r3, r2, r3 | R3 = r2 – r3 |
| | @0x20 | 0C03000C | sw r3, 12(r0) | mem(12) = r3 |
| | @0x24 | 04621800 | add r3,r3, r1 | R3 = r3 + r1 |
| | @0x28 | 80600000 | wro r3 | IO_output <= r3 |

En este ejemplo encontramos anticipación de operandos en varias ocasiones, como en el add de @0x18, donde ambos operandos van a ser anticipados por las etapas de WB y Mem, respectivamente. Posteriormente, se juega con el registro r3 y el sub de @0x1C lo obtiene como operando de forma anticipada, ya que el add anterior lo acaba de modificar. Lo mismo ocurre con las instrucciones siguientes.

Fuente:  “PRUEBA DE ANTICIPACIÓN DE OPERANDOS Y DOBLE DEPENDENCIA” en la Memoria de instrucciones.

## 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**?

Detendremos la ejecución cuando una instrucción valida que se encuentre en la etapa ID tenga dependencias que no podamos resolver mediante anticipación de operandos. La UD enviará una señal que detendrá las etapas ID y anteriores parando la escritura de nuevos datos y deteniendo la lectura de una nueva instrucción.

Para detectar riesgos en load/uso comprobamos que no existe una dependencia entre el registro en fase de ejecución y el que está en fase ID al mismo tiempo que estamos leyendo de memoria.

Los riesgos que encontramos en BEQ y WRO ocurren porque ambas instrucciones leen sus operandos en ID, y en este procesador no disponemos de anticipación de operandos para esta fase. Por tanto, si hay una dependencia con un registro usado en la instrucción siguiente en fase de ejecución o de escritura en memoria pararemos la ejecución.

**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.

- PRUEBA DE ANTICIPACIÓN DE OPERANDOS Y DOBLE DEPENDENCIA:

| | | | | |
|---|---|---|---|---|
| RESET | @0x0 | 10210003 | beq r1, r1, INI | |
| IRQ | @0x4 | 1021003E | beq r1, r1, RTI | |
| DATA_ABORT | @0x8 | 1021005D | beq r1, r1, RT_AB | |
| UNDEF | @0xC | 1021006C | beq r1, r1, RT_UN | |
| INI | @0x10 | 08010004 | lw r1, 4(r0) | R1 = mem(4) |
| | @0x14 | 08020008 | lw r2, 8(r0) | R2 = mem(8) |
| MAIN | | | | |
| | @0x18 | 04221800 | add r3, r1, r2 | R3 = r1 + r2 |
| | @0x1C | 04431801 | sub r3, r2, r3 | R3 = r2 – r3 |
| | @0x20 | 0C03000C | sw r3, 12(r0) | mem(12) = r3 |
| | @0x24 | 04621800 | add r3,r3, r1 | R3 = r3 + r1 |
| | @0x28 | 80600000 | wro r3 | IO_output <= r3 |

En este ejemplo también encontramos bastantes casos de detención por riesgos de datos, estos se dan por ejemplo en el add de @0x18, donde se ha de esperar a la lectura de r2 por parte del load. En el sub también se produce una detención causada por riesgo de datos en el registro r3. Lo mismo ocurre con el store y el WRO.

Fuente: "PRUEBA DE ANTICIPACIÓN DE OPERANDOS Y DOBLE DEPENDENCIA" en la Memoria de instrucciones.

## 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 aumentar el rendimiento de nuestro MIPS, predecimos que los saltos no se toman. Por tanto, cuando sí que se toman tenemos que invalidar la ejecución de las instrucciones que se sitúan después de nuestra instrucción de salto y que nuestro procesador ya está ejecutando.

La UD enviará una señal que matará la instrucción leída en FETCH cuando se tome un salto, siempre y cuando ese calculo no tenga dependencias (ya que podría haberse calculado con operandos incorrectos) y la instrucción sea BEQ o RTE.

**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

PRUEBA DE RIESGOS DE CONTROL:

| | | | | |
|---|---|---|---|---|
| RESET | @0x0 | 10210003 | beq r1, r1, INI | |
| IRQ | @0x4 | 1021003E | beq r1, r1, RTI | |
| DATA_ABORT | @0x8 | 1021005D | beq r1, r1, RT_AB | |
| UNDEF | @0xC | 1021006C | beq r1, r1, RT_UN | |
| INI | @0x10 | 08010000 | lw r1, 0(r0) | R1 = mem(0) |
| | @0x14 | 08010004 | lw r1, 4(r0) | R1 = mem(4) -> 1 |
| | @0x18 | 08020008 | lw r2, 8(r0) | R2 = mem(8) -> 8 |
| MAIN | | | | |
| | @0x1C | 04221800 | add r3, r1, r2 | R3 = 9 |
| | @0x20 | 10620004 | beq r3, r2, bucInf | No debería saltar (9!=8) |
| | @0x24 | 04221000 | add r2, r1, r2 | R2 = 9 |
| | @0x28 | 10620001 | beq r3, r2, sigueOK | Salta (9==8) |
| | | 10000001 | beq r0, r0, bucInf | No ejecuta esta instrucción |
| sigueOK | @0x2C | 0c020000 | sw r2, 0(r0) | |
| bucInf | @0x30 | 100000FF | beq r0, r0, bucInf | |

Con este programa comprobamos que el programa ejecuta los saltos como debe, invalidando el primer y tercer BEQ y tomando el segundo.

Fuente "PRUEBA DE RIESGOS DE CONTROL" en la memoria de instrucciones

## RF7 contadores

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

¿**Cómo**? Explicar brevemente la gestión de los contadores.

Disponemos de contadores para ciclos totales, instrucciones ejecutadas, detenciones por riesgo de datos y de control, nº de excepciones y ciclos en rutinas de excepciones.

Para el contador de instrucciones, comprobamos que la instrucción en la última etapa es válida. Para los contadores de detenciones utilizaremos las señales que paran la ejecución en las etapas ID o EX, y la señal que mata la instrucción en la fase IF, para sumar detenciones por datos y por control, respectivamente. El contador de número de excepciones aumentará cuando se acepte una excepción, y el contador de ciclos en rutinas de excepciones aumentará mientras el estado del MIPS sea 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

| | Guillermo Bajo Laborda | Álvaro de Francisco Nievas |
|---|---|---|
| Estudio del MIPS, VHDL, entorno, instalación | 2h30min | 4h |
| Adición de las nuevas instrucciones | 2h | 1h30min |
| Gestión de excepciones | 1h | 45min |
| Gestión de riesgos | 2h | 30min |
| Depuración, verificación y programas de prueba | 7h | 2h |
| Memoria | 1h | 3h |

## Conclusiones y Autoevaluación

Esta primera parte del proyecto nos ha resultado muy útil para repasar conceptos y consolidar la materia dada en clase hasta ahora. Además, consideramos que sirve de gran ayuda al estudiante poder ver todo a un nivel más bajo para entender mejor el funcionamiento de diversas cosas.

Las prácticas con Logisim, aunque ayudan mucho a visualizar lo aprendido en clase, no tiene ni remotamente la potencia de VHDL. Creemos que trabajar con herramientas más cerca de lo que se usa en la vida real nos aporta mucho.

Respecto a la autoevaluación, consideramos que hemos rendido satisfactoriamente y hemos comprendido lo solicitado por el guion, así como aspectos del MIPS que tal vez no teníamos del todo asentados a la hora de empezar con esta primera parte del proyecto.