AOC2 Proyecto 1 - Gestión de riesgos y excepciones en MIPS

Javier Resano, José Luis Briz

Fecha de entrega: Lunes 27 marzo 2023 a las 9:30

ADVERTENCIA

- Leer este guión atentamente te ahorrará tiempo.
  - Además te ayudará a consolidar o incluso entender aspectos que no aún no tienes claros sobre lo explicado en teoría o problemas.
  - También te servirá de referencia de estilo a la hora de escribir y organizar un informe.
- Comunica cualquier fallo que encuentres y pregunta lo que necesites, pero las cuestiones que estén explícitamente contestadas en este guión las atenderemos con prioridad nula.
- No cambies los nombres de los componentes y señales que se facilitan en los fuentes de partida sin consultarlo con un profesor.
-

Contenidos

1 Resumen

El objetivo de este proyecto es **aprender a trabajar con un lenguaje de descripción de hardware y un entorno de simulación profesional** basado en bancos de prueba y cronogramas. El caso de estudio es un procesador **MIPS de 32 bits segmentado en cinco etapas** como el expuesto en clase, con las características de partida que se indican en la Sec. 2.

La primera parte del proyecto consiste en modificar este MIPS para que soporte **dos nuevas instrucciones y sea capaz de gestionar interrupciones**. Eso requiere modificar tanto la Unidad de Control (UC) como la ruta de datos (Sec. 4.1).

A continuación hay que hacer las modificaciones necesarias para **gestionar los riesgos de datos** eliminando los que sea posible mediante anticipación de operandos con destino en la etapa de ejecución, y detectando los no eliminables deteniendo la ejecución de las etapas IF e ID hasta que desaparezca el riesgo (Sec. 4.3).

Después hay que resolver los riesgos de control (Sec. 4.4). Para hacerlo de forma eficiente el salto se predice no tomado (i.e. comienza a ejecutarse la instrucción en PC+4), eliminando esa instrucción si finalmente el salto resulta tomado.

La última modificación consiste en **gestionar adecuadamente un conjunto de contadores que monitorizan el rendimiento del procesador** (Sec. 2.2).

Es imprescindible diseñar programas de prueba que cubran todos los casos para comprobar que el diseño funciona (ver Sec. 5.1). Finalmente, hay que **preparar una memoria sujeta a los requisitos que se indican** en la Sec. 5.2.

El proyecto **se puede realizar por parejas** (ver Sec. 5.3).

2 Características del MIPS de partida

El procesador especificado en los fuentes VHDL que se proporcionan como punto de partida es una versión del MIPS segmentado explicado en las clases de teoría con las siguientes características importantes:

- No gestiona detenciones ni anticipación de operandos. Tal como se entrega, cualquier programa debe de evitar riesgos mediante planificación estática del código (reordenando instrucciones o insertando *nops*).
- El registro $R0$ no está cableado a cero y puede tomar cualquier valor.
- Los saltos se resuelven en la etapa ID.

Evento	**Excepción**	**Origen**	@ vector
Reset	*reset*	Externo (testbench)	0x0
Petición de interrupción	*irq*	Dispositivo externo (señal `IRQ` = 1)	0x4
Acceso a dirección inválida de la memoria de datos (e.g. no alineada o no existe)	*data abort*	`Data_Memory_Subsystem` (señal `Data_abort` = 1)	0x8
Código de operación desconocido	*undef*	`UC`	0xC

Table 1: Excepciones del MIPS del proyecto

- La escritura en los bancos de registros se produce en flanco de bajada.
- Tiene soporte para excepciones (Sec. 2.1).
- Tiene algunas entradas y salidas adicionales:
  - `IRQ`: Ver Sec. 2.1.
  - `IO_input` entrada externa, no se usa en este proyecto.
  - `IO_output` salida al exterior. Se utiliza una instrucción especial para volcar el contenido de un registro a esta salida.
  - `mem_ready` Indica si la memoria de datos está lista para servir la operación solicitada en ese ciclo. En este proyecto siempre lo está.

2.1 Soporte para excepciones

La Tab. 1 muestra los cuatro posibles eventos que en este procesador causan una *e*xcepción, i.e. de un salto incondicional a un código externo al programa en curso, con cambio de modo en el funcionamiento del procesador. La columna de la derecha muestra la *tabla de vectores de interrupción*, i.e. las direcciones de la memoria de datos en la que se almacena la dirección a la que el procesador salta cuando sucede la excepción correspondiente.

El procesador incluye los siguientes elementos para gestionar esas excepciones:

- Registro de estado (`status_reg`): tiene dos bits:
  - El bit más significativo permite deshabilitar (valor 1) o habilitar las excepciones (valor 0).
  - El bit menos significativo informa si estamos en modo excepción o estamos en modo normal.
- Registro de dirección de retorno (`IRQ_LR`): almacena la dirección a la que hay que retornar tras una excepción.
- Multiplexor para elegir la dirección de retorno: Elige entre las direcciones de las instrucciones que están en EX, ID e IF.
- Entradas adicionales en el MUX del PC para la dirección de la rutina de tratamiento de la excepción (almacenada en la tabla de vectores de interrupción) y la dirección de retorno (`IRQ_LR_output`).

El funcionamiento es el siguiente. Si las excepciones están habilitadas y sucede uno de los eventos de la Tab. 1 se interrumpirá el flujo actual de ejecución y el procesador atenderá la excepción correspondiente de la forma siguiente:

1. Se continúa la ejecución de las instrucciones que están en las etapas Mem y WB y se resetean el resto de etapas. Ejemplo: `reset_EX <= (reset or IRQ_accepted);`

2. Se almacena en `IRQ_LR` la instrucción a la que hay que retornar al terminar la ejecución. Esta será la siguiente instrucción válida que se estuviese ejecutando. Para ello las etapas ID y EX almacenan la dirección de la instrucción que hay en la etapa, y todas las etapas incluyen un bit de validez. Por ejemplo, en la etapa EX se guardan en: `PC_IRQ_EX` (dirección de la instrucción) y `valid_I_EX` (bit de validez de la instrucción que hay en EX).
3. Se cargará en el PC la dirección de tratamiento de la excepción correspondiente (según su *offset* en la tabla de vectores de interrupción). E.g. para una irq, `PC <= x"00000004"`.
4. Se actualiza el registro de estado indicando que estamos tratando una IRQ y desactivándolas para que no nos interrumpan de nuevo: `Status <= \11"`.

Para terminar la excepción y reanudar la ejecución se utilizará la nueva instrucción RTE (ver Sec. 4.1).[1]

2.2 Contadores de rendimiento

Para analizar el rendimiento de los programas existen utilidades denominadas *profilers*. Hoy día, los *profilers* utilizan contadores de eventos integrados en los propios microprocesadores. Para realizar nuestros cálculos de rendimiento, el procesador incluye varios contadores que contabilizan:

- Ciclos desde el último reset.
- Instrucciones ejecutadas (las instrucciones válidas que realizan su etapa WB).
- Detenciones causadas por riesgos de datos.
- Detenciones causadas por riesgos de control.
- Número de interrupciones atendidas.
- Ciclos consumidos atendiendo interrupciones.

3 Primeros pasos

1. Descargar de Moodle el material para el proyecto (fuentes, ejemplos).
2. Visionar los vídeos de apoyo
3. Estudiar el código del procesador y comprender cómo funciona. Identificar los componentes del código con los componentes del procesador de teoría (transparencias de clase).
    - *Es buena idea hacerse una copia ampliada del MIPS (e.g. en A3) y copiar los nombres de las señales usadas en VHDL para conectar y nombrar los componentes principales.*
4. Descargar e instalar el simulador de VHDL (ver Moodle).
5. Ejecutar el código de ejemplo (`codigo_retardado`) y seguir su ejecución ciclo a ciclo.
    - Entender la propagación de las señales de una etapa a otra atravesando los bancos de etapa.
    - Comprobar que los registros y la memoria tienen el valor deseado.

4 Trabajo a realizar

4.1 Adición de nuevas instrucciones

Realizar las modificaciones necesarias para ejecutar las dos siguientes nuevas instrucciones:

- `RTE` (*Retorno Tratamiento de Excepción*. No necesita ningún parámetro. Realiza las siguientes dos acciones:
    - Salto incondicional a la dirección almacenada en `IRQ_LR_output` (`PC <= IRQ_LR_output;`).

[1] Observar que sólo tiene sentido en el caso de una interrupción (excepción *irq*). Los otros tres casos abortan la ejecución del programa. Las rutinas de *data abort* y *undef* podrían usar esa dirección para informar de la fuente del error.

  - Actualización del registro de estado, activando las excepcioness de nuevo e indicando que salimos del modo excepción (`Status <= "00";`).

- `WRO` (*Write Output)*: Se utiliza para comunicar el procesador con un puerto de salida (registro `output_reg`) cuya salida está conectada a la salida del MIPS `IO_output`. En los procesadores tipo RISC lo habitual es que la entrada/salida sea mapeada en memoria, es decir, que se escriba en la salida con una instrucción *store* (cada registro de entrada/salida tiene asignada una dirección). Pero en esta versión del MIPS usar WRO nos resultará más sencillo. Las acciones que realiza WRO Rs son:

  - `PC <= PC+4;`
  - `output_reg <= Rs;`

4.2 Gestión de excepciones

Se pide implementar la gestión de excepciones utilizando el soporte y conforme al comportamiento descrito en la Sec. 2.1.

Os facilitamos un código de ejemplo para ver si vuestra implementación de las excepciones funciona. Antes de pasar al apartado siguiente, probad el código con vuestro diseño. Seguid su ejecución comprobando que cada excepción, y en su caso su retorno, se realizan según la especificación.

4.3 Gestión de riesgos de datos

Debéis diseñar los dos siguientes módulos:

- **Unidad de anticipación (UA)**.- Controla los dos muxes que encontraréis ya colocados a la entrada de la ALU en el código VHDL del MIPS base. Debéis diseñar la lógica que detecte si la instrucción que está en la etapa ID tiene sus operandos disponibles en el banco de registros, o si están en las etapas posteriores.

- **Unidad de Detención (UD)**.- Detecta los riesgos de datos que no se pueden solucionar con la anticipación (dependencias load/uso, y dependencias con instrucciones que usan sus operandos en ID como BEQ y WRO). Si existe riesgo debéis parar tanto ID como IF hasta que el riesgo desaparezca. Cuando se detiene la etapa ID, no se envía ninguna instrucción válida a la etapa EX. Esto se debe indicar desactivando con el bit de validez (`valid_I_EX_in`).

Importante

- **Tened en cuenta los bits de validez** a la hora de anticipar o detener. Si una etapa tiene una instrucción invalida no tiene sentido anticipar sus datos, y menos todavía detener el procesador.

- **Solo hay que detener si existe riesgo**. Para ello analizad qué operandos (Rs y Rt) utiliza cada instrucción (incluyendo las nuevas), y qué instrucciones escriben en el banco de registros. Por ejemplo:

  - Una NOP no tiene que parar para esperar a sus operandos dado que no los usa.
  - Un SW (que usa Rt como fuente) no debe de generar una dependencia ld-uso, provocando una detención de un ciclo, al confundirlo con un LW (que usa el Rt como destino)

- Para saber si la instrucción que está en una determinada etapa escribe o no en su registro destino, o si lee un dato de memoria, o si es un salto, lo más fácil es mirar sus señales de control. Para saber qué operandos usa hay que mirar su código de operación y conocer el funcionamiento de cada tipo de instrucción.

4.4 Gestión de riesgos de control

El MIPS de partida no gestiona detenciones (Sec. 2), y por tanto no puede ejecutar programas compilados para una arquitectura de lenguaje máquina (ALMA) no-retardada.

El MIPS a entregar en este proyecto debe sin embargo ejecutar programas para una ALMA no retardada, incluyendo el salto BEQ y el RTE, incorporando la gestión del riesgo de control a la UD. Para mejorar el rendimiento se asumirá predicción de salto no tomado (NT).

Es decir, cuando tenemos una instrucción de salto (BEQ) en ID, comenzaremos a ejecutar en IF la instrucción de PC+4. Por tanto si el salto es NT, no será preciso hacer nada. En caso contrario (salto T) deberemos anular la instrucción que acabamos de leer de memoria en la etapa IF. Para ello la anularemos utilizando su bit de validez (valid_I_IF).

La instrucción RTE supone un salto incondicional (siempre T). Por tanto, también habrá que anular la instrucción en IF.

4.5 Gestión de los contadores de rendimiento

Los contadores de rendimiento están inicialmente deshabilitados, salvo el contador de ciclos. Debéis asignar el valor correcto a sus señales de cuenta para que cada uno de ellos contabilice correctamente el parámetro que monitoriza.

5 Resultados y entregables

5.1 Programas de prueba

Debéis comprobar que el diseño funciona correctamente. **El diseño de tests y la verificación de un diseño es tan importante en cualquier proyecto como el propio diseño de la solución, y marca la diferencia entre un *trabajo profesional de ingeniería* y una chapuza**.

Os proporcionamos un banco de pruebas (`testbench_MIPS`) que ejecuta el programa que está cargado en memoria, así como programas de ejemplo (ensamblador y componentes RAM) para que podáis comenzar (Sec. 3). Para cargar un nuevo programa hay que escribir las instrucciones en hexadecimal en el componente RAM de instrucciones. Si queremos que trabaje con unos datos determinados podemos editar el contenido de la memoria de datos. En Moodle os damos un ejemplo, no un banco de pruebas exhaustivo.

Diseñad por tanto uno o varios programas de prueba en los que se compruebe que el procesador funciona correctamente para todos los casos representativos. Todos los cortos y paradas posibles deberán probarse al menos una vez. **La calidad de vuestras pruebas es un elemento clave en la evaluación de este proyecto. No entreguéis el proyecto sin estar seguros de que funciona. La calificación de un diseño que funciona mal en un caso sencillo es un 0**. En las especificaciones de la memoria (Sec. 5.2) indicamos la manera de presentar estos programas de prueba.

5.2 Memoria

Debe incluir debidamente completados todos los apartados que se indican en el documento de plantilla de memoria incluido en los materiales del proyecto en Moodle. Es un documento editable que podéis utilizar directamente como plantilla si os va bien. Podéis cambiar el formato si queréis siempre que el contenido mínimo esté presente. El propio documento contiene indicaciones sobre el contenido a completar en cada apartado.

5.3 Procedimiento de evaluación

Una vez entregados los proyectos a través de Moodle, los comprobamos uno por uno y les asignamos un estado:

S1: La memoria está completa y el código pasa un programa de prueba que diseñamos para esto y que vosotros no tenéis.[2]

S2: No cumple el anterior pero comprobamos que al menos pasa vuestro propio test, y estimamos que el fallo es subsanable

S3: El fallo es considerable, o bien el proyecto no pasa ni vuestro propio test. Esto último es grave: habéis perdido el tiempo enviando un proyecto sin comprobar, y nosotros hemos perdido el tiempo comprobándolo.

Revisados todos, establecemos turnos en Moodle para entrevista personal. En esa entrevista preguntamos cualquier cosa sobre el proyecto: sobre la memoria, sobre cualquier señal, componente expresión en VHDL. Podemos pedir que dibujéis un cronograma sobre un caso o preguntar sobre conceptos de la

[2] Vuestros bancos y programas de prueba deberían de cubrir al menos todos nuestros casos.

asignatura. Si lo realizáis en pareja, preguntamos por separado aunque acudáis juntos: ambos tenéis que controlar perfectamente la parte del/a compañero/a. Una buena memoria ayuda mucho en la entrevista. Quienes están en S2 tendrán 24 h para subsanar el problema. En cualquier caso, se puede suspender el proyecto en S1 o S2 si en la entrevista percibimos que alguien no conoce bien su proyecto o duda con conceptos elementales del curso. Por experiencia podemos deciros que es difícil que esto suceda: en general trabajáis bien, y conocéis vuestro proyecto.

Importante: al igual que en las prácticas, cada grupo debe desarrollar todo su código, y no se permite copiar código de otros grupos. Pero eso no quiere decir que no podáis colaborar entre vosotros. Todo lo contrario, os animamos a que compartáis vuestras ideas, busquéis soluciones de forma conjunta, y ayudéis a vuestros compañeros. Eso no sólo no os penaliza, sino que os ayudará entender mejor el proyecto y con ello a sacar mejor nota.

5.4 Estimación de tiempo

Nuestra estimación /recomendación basada en el esfuerzo de los estudiantes en cursos anteriores y proyectos similares es la siguiente:

1. Instalación y toma de contacto con el simulador: 4 horas
2. Adición de nuevas instrucciones: 1 h
3. Interrupciones: 2 h
4. Gestión de riesgos: 2 h
5. Depuración y ajustes: 10 h
6. Memoria: 3 h

Estos números asumen que lleváis la asignatura al día, comprendéis lo que hemos hecho hasta ahora en teoría, prácticas y problemas y leéis detenidamente este guión. En otro caso el incremento puede ser $2x, 3x, \infty$. Asumido lo anterior, diseñar la solución es lo que menos cuesta. La parte más dura es diseñar las pruebas para cubrir todos los posibles fallos, comprobar que no hay errores y en su caso solucionarlos. Comenzando el proyecto dos semanas antes del límite no lo conseguiréis.

En la memoria comparad vuestros datos de dedicación con nuestra estimación y analizar las divergencias. Sed profesionales y registrad el número de horas que dedicáis cada día en lugar de dar un número a ojo al acabar; podéis utilizar cualquier aplicación de seguimiento de proyectos. Lo tendréis que hacer en vuestra vida profesional.

5.5 Formato de las instrucciones

Importante: No utilizar formatos que no se correspondan con los del MIPS base que se proporciona.

Entre los materiales en Moodle se incluye una hoja Excel que automatiza la codificación de instrucciones.

El formato de las instrucciones es el del MIPS, tal como se ha estudiado en clase (Tema 2), con las matizaciones siguientes:

- Codificación de los códigos de operación (campo op):
  - `NOP: 000000; Arit: 000001; LW: 000010; SW: 000011`
  - `BEQ: 000100; RTE: 001000; WRO: 100000`
- Codificación de las operaciones de la ALU entera (campo *funct* y señal `ALU_ctrl`):
  - `+: 00000; -: 00001; AND:00010; OR: 00011`

Las memorias de datos (`memoriaRAM_D_test.vhd`) e instrucciones (`memoriaRAM_I_test.vhd`) se definen en los módulos VHDL como una señal tipo array de 128 palabras de 32 bits.

- *Observad que direccionamos palabras, no bytes.* El direccionamiento de un byte no alienado (en una dirección que no es múltiplo de 4) es un acceso inválido y general la excepción *data abort* (Sec. 2.1).

Cada posición en VHDL es una palabra codificada en hexadecimal (8 nibbles, e.g. `X"88010000"`, `X"00000000"...`). Las direcciones son de 32b (`ADDR`) y llegan al componente de memoria completas pero sólo se tienen en cuenta los 7 bits de direccionamiento de palabra. Estudia el código de los módulos de memoria para comprenderlo bien, y averigua la relación entre dirección y número de palabra en memoria, para que tengas soltura relacionando dirección efectiva en las instrucciones tipo load y store y la palabra en memoria que codificarás en VHDL y verás en el simulador.

5.6 Entorno de trabajo

Consulta en Moodle (Sec. Proyectos de evaluación continua) las características e instrucciones de instalación de los entornos que puedes utilizar para compilar y simular diseños VHDL visualizando las formas de ondas.