ARQUITECTURA DE COMPUTADORES II – Curso 2018

PRÁCTICA 3: ARQUITECTURAS RISC

Bibliografía de referencia:

[1] Apéndice B "Instruction Set Principles and Examples" del libro "*Computer Architecture – A Quantitative Approach*" de J. Hennessy y D. Patterson, quinta edición.

[2] Apéndice K "Survey of Instruction Set Architectures" del libro "*Computer Architecture – A Quantitative Approach*" de J. Hennessy y D. Patterson, quinta edición. Descarga: http://booksite.elsevier.com/9780123838728/references.php

[3] Capítulo 2 "*The Basics*" del libro "*Microprocessor Architecture – From Simple Pipelines to Chip Multiprocessors*" (primera edición), por Jean-Loup Baer.

[4] Capítulo 4 "*The MIPS R2000 Instruction Set*" del libro de apuntes "*MIPS Assembly Language Programming*", por Daniel J. Ellard.

[5] Capítulo 4 "*ARM Organization and Implementation*" del libro "*ARM System-on-Chip Architecture*" (segunda edición), por Steve Furber.

1) Codificación de las palabras de instrucciones en procesadores RISC.

a) Represente gráficamente la codificación de las palabras de instrucciones utilizadas en los procesadores MIPS (formato I, formato R y formato J).
b) Describa la utilidad de los campos que componen cada formato.
c) ¿Qué relación existe entre la codificación de las palabras de instrucciones y la cantidad de registros disponibles en la ISA del procesador?

2) Acerca de la segmentación clásica RISC de 5 etapas utilizada en MIPS:

a) Enumere las etapas explicando su utilidad.
b) Para cada uno de los tipos de instrucciones describa la evolución de la instrucción a medida que atraviesa cada una de las etapas del pipeline.
c) ¿Qué riesgos pueden generarse con esta organización de pipeline?
d) Compare la respuesta del inciso anterior con la organización alternativa del pipeline de 5 etapas propuesta en la sección 2.1.5 de la referencia [3].

3) Siguiendo la ref. [5] compare las segmentaciones de 3 etapas utilizada hasta el ARM7 y la de 5 etapas utilizadas en las arquitecturas ARM posteriores:

a) ¿Qué tipo de arquitectura de memoria utiliza cada organización?
b) ¿por qué la versión de 3 etapas no requiere de resolución de riesgos de datos?
c) ¿Qué diferencias presenta el pipeline de 5 etapas de ARM respecto del clásico RISC?

4) Realice una descripción de los principales aspectos de la arquitectura de los procesadores ARM.

a) ¿Qué aspectos preserva del RISC teórico?
b) ¿Qué aspectos característicos del RISC teórico fueron descartados en este diseño? ¿Por qué?
c) ¿Qué prestaciones novedosas se agregaron?
d) ¿Qué sectores del mercado son dominados por esta arquitectura y cuáles no?

5) Describa la utilización de ventanas de registros en los procesadores SPARC como alternativa a la utilización de una pila. ¿Qué ventajas presenta?

6) En muchos casos los austeros repertorios de instrucciones RISC carecen de instrucciones nativas *explícitas* que realicen operaciones frecuentemente utilizadas por los programadores, como por ejemplo la copia de valores entre registros. Para simplificar la tarea del programador los compiladores proveen las llamadas pseudo-instrucciones, que son instrucciones "ficticias" que son reconocidas por el compilador y reemplazadas por una o más instrucciones nativas del procesador que realizan una función equivalente. Partiendo de los ejemplos presentes en la sección 4.8.1 de la ref. [4] muestre como se podrían sintetizar en instrucciones nativas de MIPS R2000 las siguientes pseudo-instrucciones: MOVE, NOT, NEG, BGE y SEQ. Tenga en cuenta que el código generado por el compilador para reemplazar pseudo-instrucciones no puede alterar ningún registro que pueda ser utilizado por el usuario; para salvar esta

limitación el registro *R1* del MIPS R2000 está reservado para su utilización por parte del compilador.

7) Segmentación RISC con instrucciones multiciclo: Considere un procesador RISC segmentado en 5 etapas al que se desea incorporar funcionalidad de punto flotante de doble precisión. Para ello se incorpora una unidad de multiplicación y suma segmentada en 4 etapas (de duración equivalente) y un banco de registros de 64 bits dedicado. Muestre en un diagrama de tiempo, utilizando un programa simple, cómo conviven en el cauce del pipeline operaciones de enteros y de punto flotante.

Para los siguientes ejercicios vamos a trabajar con el procesador RISC R32. El código fuente en VHDL de dicho procesador, así como el archivo de proyecto para Modelsim y la documentación asociada al proyecto se encuentran disponibles en el sitio web de la cátedra junto a esta práctica.

Para realizar las simulaciones es necesario contar con el software **Modelsim PE Student Edition** que puede ser descargado gratuitamente desde el sitio de la firma **Mentor Graphics** (link al sitio de descarga). Antes de finalizar la instalación del software se deberá crear e instalar la licencia gratuita de uso; para esto siga las instrucciones que presenta el instalador.

Una vez instalado el software, abrir el archivo de proyecto "*r32.mpf*" que se encuentra en la subcarpeta "*modelsim*" del contenido del proyecto R32.

Tener en cuenta al usar el Modelsim:

- Puede editar los archivos del código fuente haciendo doble click sobre ellos en la página "*project*".
- Siempre que modifique el código VHDL, guarde sus cambios y recompile antes de iniciar una nueva simulación. De esa forma sus cambios entrarán en efecto.
- Para simular, haga doble click sobre el nombre del entidad base que encontrará dentro de la librería "*work*" en la página "*library*" del Modelsim. Eso abrirá la ventana "*objects*", donde podrá seleccionar las señales que quiere observar haciendo click derecho y ejecutando la acción "*Add wave*"). Una vez elegidas las señales puede comenzar la simulación con el menú "*Simulate/Run/Run-next*".
- Las entidades base que utilizaremos son:
  - *e_r32_simple* – Procesador R32 con organización no segmentada.
  - *e_r32_segmented* – Procesador R32 con organización con segmentación de 4 etapas.

Los programas Fibonacci y Hailstone que corre R32 son prácticamente idénticos a los utilizados en el simulador WinMIPS64 en la práctica anterior, por lo que puede recurrir a los códigos fuentes provistos con dicha práctica.

Para más información recurrir al documento de diseño del procesador, el cual encontrará en la subcarpeta "*doc*" del proyecto.

---

8) Determine a partir de la documentación que del procesador R32:

a) ¿Qué tipos de instrucción hay disponibles? ¿Qué modos de direccionamiento?
b) ¿Qué y cuántos formatos de instrucción tiene? ¿Qué utilidad tienen los campos principales? ¿Cuántos registros hay? ¿Cuáles tienen funciones especiales?
c) ¿Qué relación guardan los formatos de la palabra de instrucciones con las clases de instrucciones disponibles?
d) ¿Qué etapas tiene el pipeline de la versión segmentada? ¿qué relación guardan las 4 etapas del R32 con el pipeline RISC clásico de 5 etapas?
e) ¿Qué riesgos pueden aparecer en este pipeline? ¿RAW, WAW, WAR?
f) ¿Hay dependencias de datos que no puedan resolverse mediante adelantamientos?

g) ¿En qué etapa se deciden los saltos? ¿qué estrategia de control de riesgos de control utiliza?

9) Entrada en calor con el simulador. En este ejercicio utilizaremos el sistema descripto por la entidad *e_r32_simple*. Esta entidad está compuesta de una versión no segmentada , con un reloj de 1 MHz, una memoria de datos y una memoria de programa.

a) Cargue el proyecto en Modelsim y compílelo para asegurarse de que no hay problemas con la licencia de uso.
b) El programa que correrá R32 se modifica reemplazando la arquitectura que implementa la entidad memoria de programa (*e_program_memory*). La arquitectura de la memoria se elige al instanciar un componente a partir de la entidad. Confirme que el procesador R32 tiene cargado el programa Fibonacci. Lo puede ver examinando qué arquitectura está siendo seleccionada en el archivo "*r32_simple.vhdl*", en las proximidades de la línea 59.
c) Familiarícese con la forma en que está expresado el programa. Para ello abra el archivo "*program_memory.vhdl*" y examine las arquitecturas *a_program_memory_** disponibles.
d) Las funciones **assembleALURR**, **assembleALURI**, **assembleLOAD, assembleSTORE** y **assembleJUMP**, **assembleNOP** están definidas en el archivo "*assembler.vhdl*". Las últimas están incompletas; utilizando el ejemplo de la función **assembleALURR** y la documentación de los formatos de instrucciones utilizados por el procesador complete las funciones restantes.
e) Compile y simule la entidad *e_r32_simple*, graficando los siguientes puertos de la entidad: reloj, contador de programa, código de instrucción, valores de los primeros 8 registros del procesador y valores de las primeras 8 posiciones de memoria.
f) Cuando Modelsim termine de realizar la simulación en la ventana "*wave*" verá el diagrama de tiempos del sistema. En el diagrama podrá ver el valor del contador de programa, el código de la instrucción ejecutado, y el valor de los primeros 8 registros y las primeras 8 posiciones de memoria, todos estos valores correspondientes a la posición apuntada por el cursor. Verifique que esta versión no segmentada ejecuta una instrucción cada microsegundo.
g) Seleccione la línea "*MONITOR_CLOCK*" y desplácese por ella verificando que efectivamente el programa almacena en memoria los primeros números de la secuencia Fibonacci (ver señales "*MONITOR_M**"). Si los resultados no son correctos verifique que sean correctas las funciones **assemble*** que modificó en el inciso (d).
h) *En* "*r32_simple.vhdl*" cambie el programa cargado en la memoria de programa de R32 para que ahora ejecute el programa Hailstone. Compile, simule y verifique que la secuencia que almacena el programa en memoria es efectivamente la secuencia de máximos correspondientes a las secuencias Hailstone de los primeros números naturales.

10) Ahora podemos ir más rápido. Simule la entidad *e_r32_segmented*. Ésta es idéntica a *e_r32_simple* (compuesta de memorias de datos y programa, la cpu y un reloj de 1Mhz) salvo porque la cpu utilizada es una versión segmentada de la arquitectura R32. La entidad *e_r32_segmented* tiene puertos que permiten observar qué instrucción se encuentra en cada etapa del pipeline (a partir del contador de programa que corresponde a dicha instrucción), las etapas que serán detenidas durante el ciclo de reloj siguiente ("*stalled*"), los operandos que requieren adelantamiento ("*forward_operand*"), y si la condición de los saltos condicionales evalúa a verdadero ("*branch_taken*").

1. Verifique que los programas Fibonacci y Hailstone funcionan exactamente igual que en la versión no segmentada; la resolución de riesgos del pipeline de este procesador hace que la diferencia en la organización de las CPUs de *e_r32_simple* y *e_r32_segmented* sea transparente a la ISA.
2. Verifique que las instrucciones recién actualizan el estado de los registros del procesador cuando pasan por WRITEBACK. Compare esto con el comportamiento de la versión no segmentada del procesador.
3. Busque en el programa las instrucciones en las que los riesgos de datos requieren de adelantamiento para ser resueltas, y verifique que efectivamente el procesador realiza adelantamientos al ejecutar esas instrucciones.

4. Busque en el programa las instrucciones en las que los riesgos de datos requieren detenciones del pipeline para ser resueltas, y verifique en la simulación que dichas paradas del pipeline efectivamente ocurren.
5. Verifique la naturaleza *predict-untaken* de los saltos: las instrucciones inmediatamente a continuación de un salto entran al pipeline, pero son anuladas si la condición de salto indica que el programa debe bifurcarse.

11) Rompiendo el juguete.

1. Analice el funcionamiento del control de riesgos del pipeline de la CPU segmentada. Puede encontrar el código que decide los adelantamientos, las detenciones y el control de saltos en el archivo "*pipeline_control.vhd*" (aproximadamente alrededor de la línea 114).
2. Elimine la generación de adelantamientos en dicho control y verifique que los programas Fibonacci y Hailstone dejan de ser correctos al simular *e_r32_segmented*. ¿Cómo sería necesario modificar los programas para que funcionen con esta nueva organización sin adelantamientos? ¿Es transparente a la ISA esta organización del procesador? ¿funciona de la misma manera un programa ejecutado en la versión segmentada y no segmentada del procesador?
3. Vuelva atrás los cambios anteriores y ahora modifique el control del pipeline para que la CPU segmentada tenga un único *delay slot* después de los saltos. ¿Qué ocurre con la ejecución de nuestros programas de prueba, Fibonacci y Hailstone?