Anteproyecto I

Roberto Sánchez Cárdenas - B77059
Gabriel Jiménez Amador - B73895

San José, 31 de agosto de 2021
Laboratorio de Circuitos Digitales

Índice

1. Introducción

Este proyecto tiene como fin obtener el conocimiento necesario para el correcto desarrollo de las actividades de laboratorio que serán propuestas más adelante en el curso de Laboratorio de Circuitos Digitales. En este caso se desarrolla una práctica introductoria sobre la síntesis de código HDL en un FPGA.

El estudiante debe luego de crear varios códigos, simularlos y justificar los resultados obtenidos de acuerdo con los conceptos de procesadores RISC.

2. Objetivos

- Familiarizar al estudiante con el ambiente integrado de desarrollo Vivado de Xilinx.
- Presentar al estudiante una implementación en HDL de un microprocesador de la arquitectura RISC-V.
- Presentar al estudiante un ejemplo sencillo en código C para ser ejecutado en el microprocesador.

3. Anteproyecto

3.1. Pipelining

Pipelining es un método por el cual las computadoras logran ejecutar múltiples instrucciones a la vez por medio de la superposición de instrucciones. Actualmente es un procedimiento casi universal en la computación, pues la mejora en el desempeño de los procesadores es significativa. Para lograr un pipelining eficiente, se divide el procesamiento de instrucciones en etapas. Es común que el proceso se divida en 5 etapas. [1]

3.1.1. Etapas clásicas de un pipeline en un procesador RISC

Al iniciar un ciclo de reloj, se busca una instrucción y comienza la ejecución de 5 ciclos de reloj. Al comparar con un procesador sin pipelining, se puede alcanzar hasta cinco veces el rendimiento del primero, con pipelining se inicia una instrucción cada ciclo de reloj, a diferencia de uno cada 5 ciclos. Las 5 etapas son se describen a manera de resumen a continuación [2]:

- IF - **Instrucion Fetch**

  Trae la instrucción de memoria (fetch) indicada por el Program Counter (PC). Se asume que la instrucción no es un salto y se incremente el PC por 1 para apuntar a la siguiente instrucción.

- ID - **Instruction Decode**

  Lee el registro, decodifica la instrucción y reconoce su tipo siguiendo el opcode respectivo.

- EX - **Execution**

  Ejecuta la instrucción, puede hacer varias cosas según el tipo de instrucción:

  - Para una instrucción aritmética, la ALU realiza la operación aritmética o lógica.
  - Para una instrucción load/store se calculan las direcciones en memoria.
  - Para un salto/branch se configura el PC con el valor correcto para la siguiente instrucción a procesar.

- MEM - **Memory Access**

  Accede en memoria según los operandos de memoria.

  - En el caso de un load, el contenido de Mem [addr] es obtenido desde algún nivel en la jerarquía de cache.
  - En el caso de un store, el contenido de la dirección es modificada.
  - Si la instrucción no es load/store, esta etapa es un NOP, nada ocurre.

- WB - **Write back**

  Escribe los resultados de la operación en los registros, si la instrucción no es un salto o un load/store.

El tiempo de ejecución de una etapa puede llegar a ser tan pequeño como el tiempo ejecución de la etapa más lenta. Esto se observa en la Figura 1.

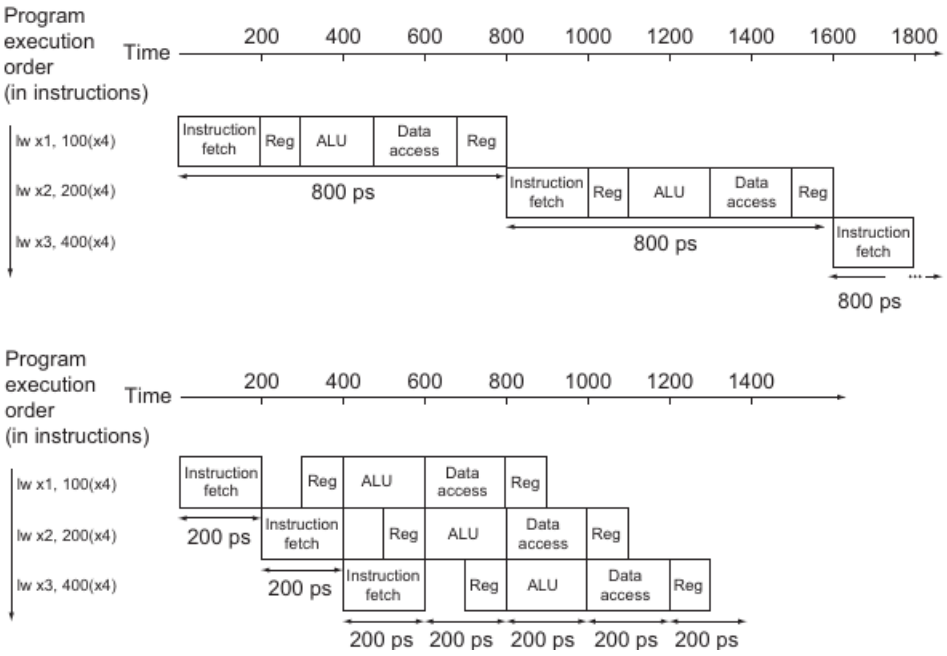


Figura 1: Ejecución con y sin pipeline [3]

3.1.2. Riesgos que atentan contra el pipeline

Cuando se utiliza este tipo de técnica, es importante considerar que existen instrucciones que dependen de datos calculados en una instrucción previa entonces el pipeline debe esperar hasta que este dato se genere, por esto los programas no se pueden paralelizar al 100 %. Para disminuir los efectos de esto, se utilizan técnicas como la predicción de saltos y la ejecución fuera de orden. [1]

Estos riesgos o *hazards* se dividen en 3 bloques principales [2]:

1. **Riesgos Estructurales**

   Son dados por conflictos de recursos del hardware, estos no son ilimitados y el riesgo ocurre cuando este no puede soportar todas las combinaciones de instrucciones superpuestas. Por ejemplo, sea el caso que se ocupara escribir a dos registros en etapas diferentes, pero solo se tiene un puerto de escritura. El hardware es insuficiente para realizar estas operaciones simultáneamente y se da un riesgo estructural. En la imagen de la Figura 2 se observa este riesgo en un pipeline.

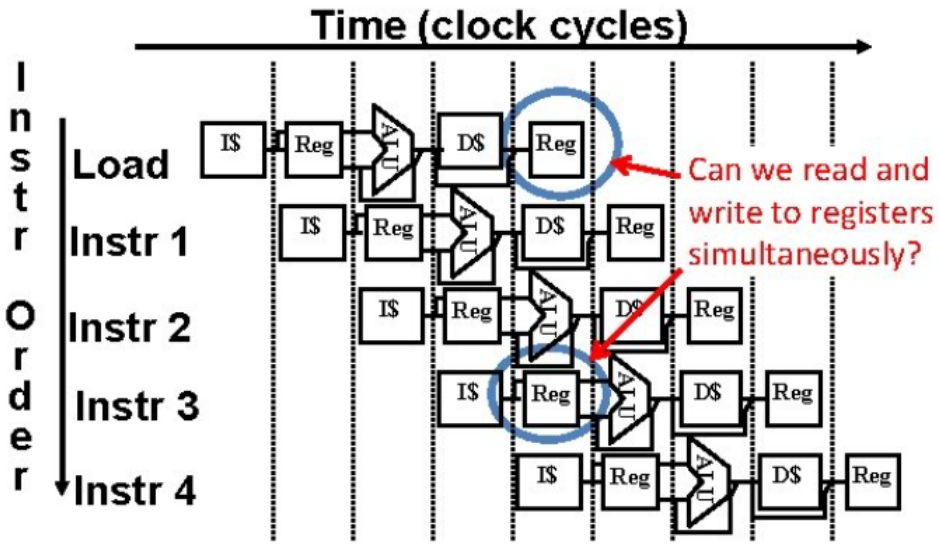


Figura 2: Riesgo estructural en un pipeline. [4]

2. **Riesgos de Datos**

   Se dan por dependencia de datos, puesto que las instrucciones son procesadas en paralelo y no secuencialmente. Se puede dar el caso de que una instrucción requiere un dato que aún no se ha terminado de procesar.

   Este tipo de riesgos se dividen en las siguientes categorías y se ejemplificarán con el proceso de dos instrucciones A y B dónde A es procesada primero.

   - RAW - **read after write**

     La instrucción B intenta leer antes de que A pueda escribir el valor correcto. A recibe el valor incorrecto.

     Por ejemplo, se puede considerar que las instrucciones son
     A: ADD R2, R1, R3
     B: ADD R4, R2, R3

     La primera instrucción calcula un valor a guardar en el registro R2 y la segunda utilizará este valor para calcular un resultado para el registro R4. Sin embargo, en un procesador con pipelining cuando se obtienen operandos para la segunda operación, los resultados de la primera aún no se han guardado y se da un riesgo de datos RAW.

   - WAW - **write after write**

     La instrucción B intenta escribir antes de que A escriba. El valor en el registro final será el de A en lugar de lo esperado de A.

   - WAR - **write after read**

La instrucción B intenta escribir a un registro antes de que la instrucción A pueda leerlo. La instrucción A usará el valor incorrecto.

3. **Riesgos de Control**

   Suceden al darse instrucciones de salto condicionales puesto que estas modifican el PC al darse una condición específica que no se sabe si se da o no. Por este riesgo es que el pipeline es retrasado hasta que la condición sea evaluada.

3.2. Arquitectura RISC-V

RISC-V es un conjunto de instrucciones de arquitectura (ISA, por sus siglas en inglés) que surge en 2010 en la Universidad de Berkeley y que además, es un proyecto abierto por lo que está su licencia puede ser utilizada por cualquier persona siempre y cuando se apegue a los lineamientos estipulados por la misma.

Esta arquitectura es del tipo computadora con conjunto de instrucciones reducido (RISC por sus siglas en inglés), este tipo de instrucciones realizan operaciones más básicas que su contra parte CISC, que posee operaciones complejas, sin embargo esto no significa que sea un modelo limitado. [3]

3.2.1. Diferencias entre RISC-V y otras arquitecturas RISC

A diferencia de otros ISA sea otros RISC como ARM o CISC como el x86 de Intel, RISC-V sigue un modelo que prioriza la modularidad y permite añadir instrucciones conforme se requieren para cierta aplicación. Este tipo de implementación permite implementaciones muy pequeñas y de muy bajo consumo energético. Además de su bajo consumo, esta arquitectura posee 7 ventajas claves [3]:

- Costo — RISC-V es *open source* no se requiere pagar por el IP, además de reducir el tamaño del silicio requerido del chip.
- Simplicidad — RISC-V es mucho más pequeño que otras ISAs comerciales.
- Rendimiento
- Aislamiento
- Espacio para crear
- Tamaño de programas
- Facilidad de programación

3.2.2. Set de instrucciones base RISC-V

Registro

- add: suma de registros
- sub: resta de registros
- xor: operación lógica o-exclusiva
- or: operación lógica o
- and: operación lógica y
- sll: desplazamiento lógico a la izquierda
- srl: desplazamiento lógico a la derecha
- sra: desplazamiento aritmético a la derecha
- slt: set menor que inmediato
- sltu: set menor que inmediato sin signo

Inmediatas

- addi: suma inmediata
- xori: operación lógica o-exclusiva inmediata
- ori: operación lógica o inmediata
- andi: operación lógica y inmediata
- slli: desplazamiento lógico a la izquierda inmediato
- srli: desplazamiento lógico a la derecha inmediato
- srai: desplazamiento a la derecha pero el bit más significativo se guarda
- slti: set menor que inmediato
- sltiu: set menor que inmediato sin signo
- lb: cargar byte
- lh: carga 16-bits de memoria
- lw: carga un word

- lbu: carga 8-bits de memoria pero extiende a 32 con 0s
- lhu: carga 16-bits de memoria pero extiende a 32 con 0s

Guardar

- sb: guarda 8 bits
- sh: guarda 16-bit
- sw: guarda un word

Salto/branch

- beq: salto condicional de igualdad
- bne: salto condicional de desiguadad
- blt: salto condicional menor que
- bge: salto condicional mayor que
- bltu: salto condicional menor que sin signo
- bgeu: salto condicional menor que sin signo

Saltos

- jal: salto incondicional
- jalr: salto incondicional de registro

Formato

- lui: cargar entero inmediato

Otras

- ecall: termina el programa

Como se menciona previamente, la arquitectura RISC-V es modular, por lo que las instrucciones presentadas previamente son únicamente las más básicas. Se puede extender para que tenga disponibles una mayor gama de instrucciones más complejas. El formato de registros de cada instrucción y algunos módulos adicionales se puede ver en el manual titulado: The RISC-V Instruction Set Manual. [5]

3.3. Compilación cruzada

La compilación cruzada es una forma mediante la cual se puede construir el archivo binario de un programa en cierta plataforma y que este pueda ser ejecutado en otra plataforma distinta. La importancia de esto es que dichas plataformas pueden tener distintos CPUs. Este método es muy conveniente pues en sistemas con poca capacidad, como lo son los sistemas embebidos, en muchas ocasiones no se puede programar y compilar el código directamente en la plataforma, sin embargo la compilación cruzada permite desarrollar programas en un sistema Linux que pueda ser corrido en un sistema embebido. Para el caso de RISC-V se puede aprovechar la compilación cruzada con compiladores como gcc o clang. [6]

3.4. Ejemplo en ensamblador RISC-V

A continuación se muestra un código en RISC-V ensamblador que calcula y guarda todos los números de la serie de Fibonacci hasta n = 10946.

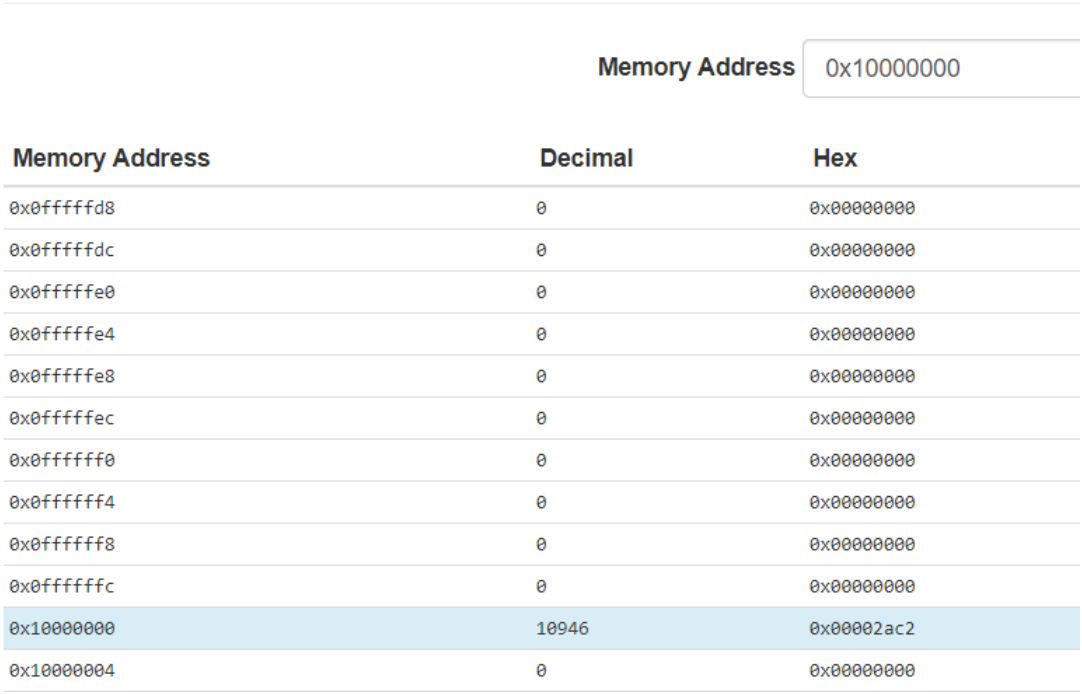
```
add t0,x0,x0            #x = 0
addi t1,x0,1            #y = 1
add t2,x0,x0            #z = 0
lui t4,10946            #n = 10946 es el limite
srli t4,t4,12           #Para contrarrestar donde lui lo posiciona
lui s0, 0x10000         #Posicion de memoria, a como lui lo posiciona queda 0x10000000
sw x0, 0(s0)            #0 es el primero en la serie de Fibonacci

Loop:
bge t2,t4,End           #Loop termina cuando llega a 10946
add t2,t0,t1            #z = x+y
addi t0,t1,0            #x = y
addi t1,t2,0            #y = z
andi t3,t2,1            #Mascara para ver si es par
beq t3,x0,Store         #Si el bit es 0 entonces es par y se guarda
jal x0, Loop

Store:
sw t2, 0(s0)            #Guarda z (par)
jal x0, Loop

End:
ecall                   #Termina el programa
```

Corriéndolo, a manera de demostración, en este interpretador RISC-V de Cornell University se ve que se guardan progresivamente (sobreescribiéndose) los números 0,2,8,34,144,610,2584 y 10946 y el programa termina. La dirección `0x10000000` queda como se observa en la Figura 3.



Memory Address 0x10000000

Memory Address	Decimal	Hex
0x0fffffd8	0	0x00000000
0x0fffffdc	0	0x00000000
0x0fffffe0	0	0x00000000
0x0fffffe4	0	0x00000000
0x0fffffe8	0	0x00000000
0x0fffffec	0	0x00000000
0x0ffffff0	0	0x00000000
0x0ffffff4	0	0x00000000
0x0ffffff8	0	0x00000000
0x0ffffffc	0	0x00000000
0x10000000	10946	0x00002ac2
0x10000004	0	0x00000000

Figura 3: Resultados del programa, utilizando el interpretador RISC-V de Cornell University.

Referencias

[1] David A. Patterson John L. Hennessy. *Computer Architecture, Sixth Edition: A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, 6 edition, 2017.

[2] J.L. Baer. *Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors*. Cambridge University Press, 2009.

[3] David A. Patterson John L. Hennessy. *Computer Organization and Design RISC-V Edition, Second Edition*. Morgan Kaufmann, 2 edition, 2021.

[4] Justin Hsia. Cs 61 c: Great ideas in computer architecture, pipelining hazards, Abril 2013.

[5] Andrew Waterman, Yunsup Lee, David A. Patterson, and Krste Asanović. The risc-v instruction set manual, volume i: User-level isa, version 2.0. Technical Report UCB/EECS-2014-54, EECS Department, University of California, Berkeley, May 2014.

[6] GNU. automake.