Universidad Nacional de Córdoba

Facultad de Matemática, Astronomía, Física y Computación

Facultad de Matemática, Astronomía, Física y Computación

Laboratorio 1:

MIPS con Pipeline

Autores:
Mario Ferreyra
Marco Moresi
Juan Ignacio Farias
Alan Bracco

Docentes:
Pablo Ferreyra
Delfina Velez

Asignatura:
Arquitectura de Computadoras

23 de Septiembre de 2015

Índice

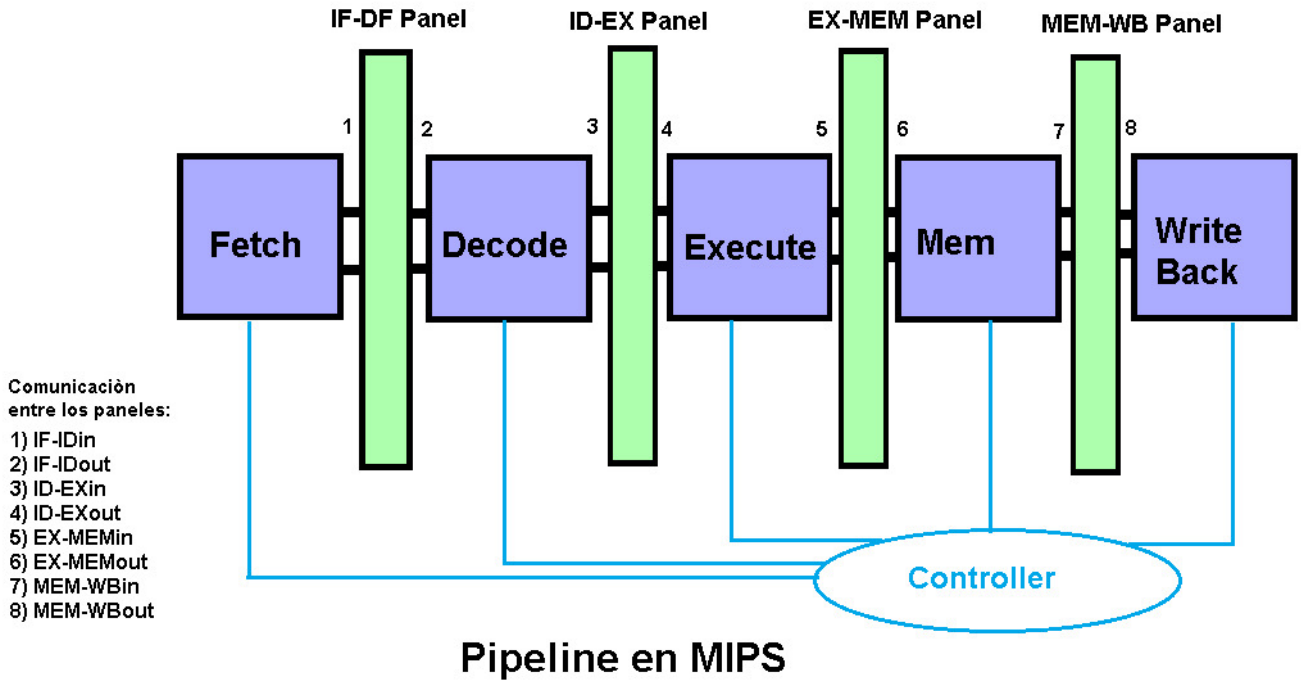
1. Estructuración del código

La estructuración del código se organiza de la siguiente manera: Principalmente, se debieron desarrollar los módulos otorgados por la cátedra. Estos módulos son las 5 etapas de un microprocesador MIPS con Pipeline, ellas son:

- Fetch
- Decode
- Execute
- Memory
- Writeback

Los módulos están formados por los pequeños componentes desarrollados durante los prácticos de tal forma que respete el funcionamiento de cada modulo uniendo estos componentes para conformar el microprocesador.

Los módulos se conectan mediante paneles, que sirven para comunicar las distintas etapas del pipeline para ello tuvimos que desarrollar señales especiales, (especificadas en la imagen adjuntada) con el objetivo de encapsular las salidas y poder manejarlas de un modo mas cómodo, y logrando así un código un poco mas legible en ciertos aspectos.



Pipeline en MIPS

2. Dificultades Encontradas

El hecho de encontrar la manera correcta de conectar los módulos fue la parte mas difícil; junto con esto se sumaron algunas complicaciones como no comprender de donde salían algunas señales y que señal iba con cada una.

La forma de resolverlas fue terminar de comprender el flujo de los datos a través del MIPS, me refiero como datos, a las instrucciones.

3. Como probar el Proyecto

Para facilitar la prueba del proyecto, se realizo un *Makefile*, el cual tiene los siguientes comandos:

$ make

Este comando se encargar de compilar todos los archivos con la extensión "*vhd*"se forma correcta (es decir, en el orden adecuado) hasta llegar a compilar el archivo que representaría el microprocesador, dicho archivo se llama "*mips.vhd*".

Este comando también se encarga de generar un archivo llamado "*DUMP_FILE.txt*", el cual contiene el contenido de la memoria durante la ejecución del *Testbench*.

$ make clean

Este comando se encarga de borrar todo lo creado por *make*, es decir: archivos con la extensión "*vcd*"del *Testbench*, códigos objetos y el archivo "*DUMP_FILE.txt*"mencionado anteriormente.

$ make signal

Este comando se encarga de ejecutar el programa "*GTKWave*"(recordar tener instalado) con el cual se podrá ver las señales generadas por el programa a partir del *Testbench*, de esta formase podrá chequear que el MIPS trabajada de la forma esperada.

Aclaración

Para la prueba del proyecto, se recomienda ejecutar la siguiente secuencia de comandos:

1. $ make
2. $ less DUMP_FILE.txt
3. $ make signal
4. $ make clean

Observación 1: El paso 2 es para ver el contenido del archivo *DUMP_FILE.txt* desde la consola, si necesidad de abrirlo con algún editor de texto.

Observación 2: Ante cualquier problema de compilación, ejecutar la siguiente secuencia de comandos:

1. $ make clean
2. $ make

4. Elaboración 1

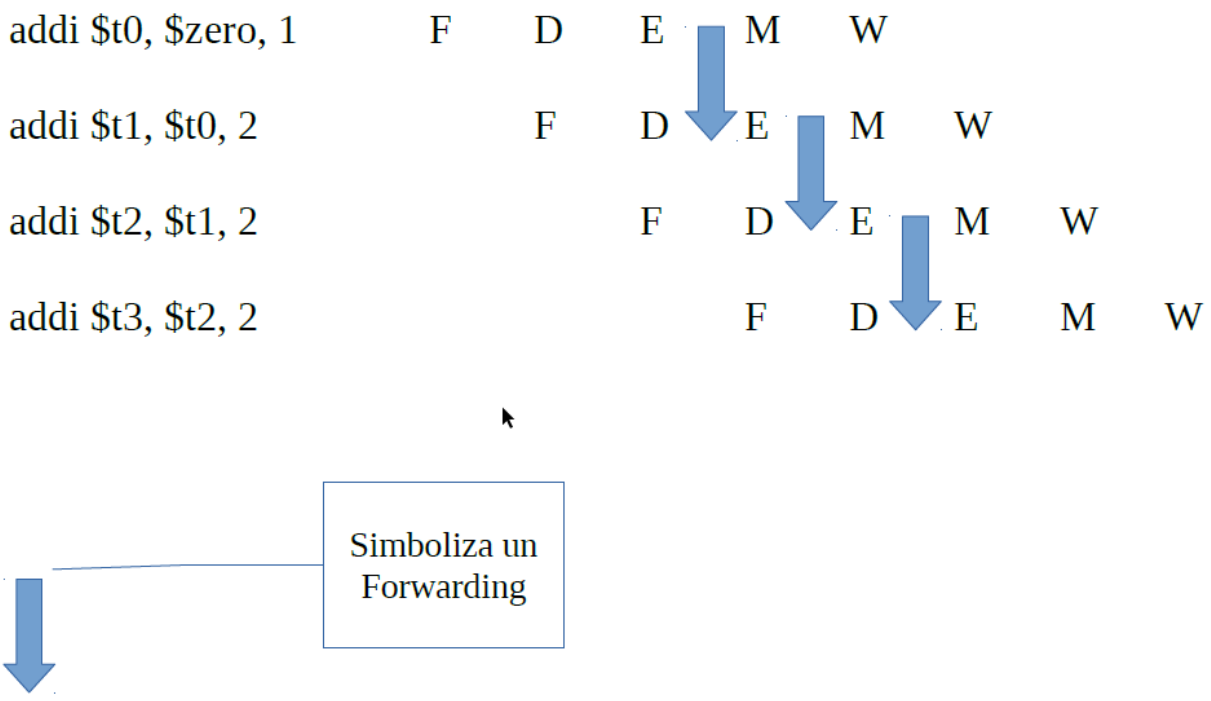
```
addi $t0, $zero, 1
addi $t1, $t0, 2
addi $t2, $t1, 2
addi $t3, $t2, 2

sw $t0, 0($zero)
sw $t1, 4($zero)
sw $t2, 8($zero)
sw $t3, 12($zero)

exit: beq $zero $zero exit
```

Para este código no es posible hacer un reordenamiento estático, ya que a partir de la segunda instrucción "*addi*"toman como argumento el resultado de la operación inmediatamente anterior, con lo cual se debe esperar hasta que el resultado sea escrito en el registro de destino. En cambio con las instrucciones "*sw*"no hay problema alguno de reordenamiento ya que estas solo se encargan de guardar el contenido de los registros el "$\$zero + CONSTANT$".

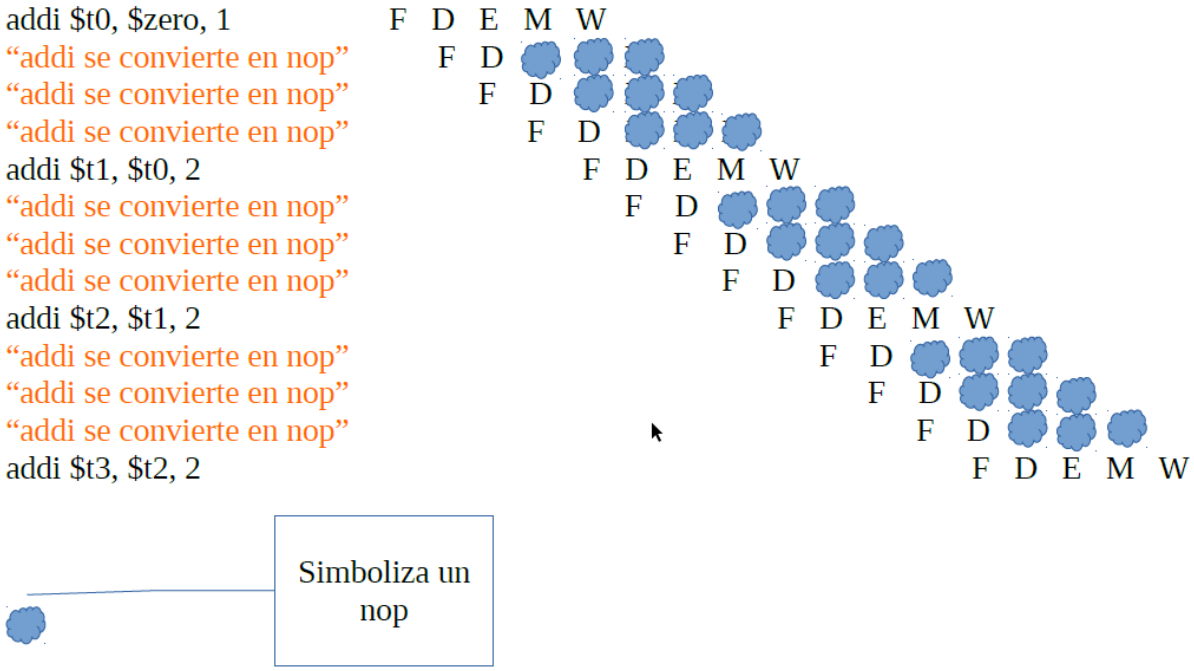
Hay dos formas de resolver el problema planteado, la primera es usando la técnica de anticipación de datos (forwarding) y la segunda es usando instrucciones *nops*.

Observación: Se analizara la parte de las instrucciones "*addi*", ya que en la parte de las instrucciones "*sw*"no hay problema alguno.

4.1. Forwarding



4.2. NOPS


addi $t0, $zero, 1
“addi se convierte en nop”
“addi se convierte en nop”
“addi se convierte en nop”
addi $t1, $t0, 2
“addi se convierte en nop”
“addi se convierte en nop”
“addi se convierte en nop”
addi $t2, $t1, 2
“addi se convierte en nop”
“addi se convierte en nop”
“addi se convierte en nop”
addi $t3, $t2, 2
F D E M W
Simboliza un nop

5. Elaboración 2

Retardos

- Datapath (sin Pipeline): 280 ns
- Fetch: 50 ns
- Decode: 45 ns
- Execute: 75 ns
- Memory: 90 ns
- Writeback: 20 ns

¿Cuál es el periodo del clock de la versión encausada (con Pipeline)?

El período del clock de la versión con Pipeline tendría que ser de 90 ns, ya que se trata de la etapa con mayor retardo de todas las etapas del Pipeline.

¿Cuál es la latencia de las instrucciones?

- Versión sin Pipeline: 1 pulso de reloj $\rightarrow$ 280 ns
- Versión con Pipeline: 5 pulsos de reloj $\rightarrow$ 450 ns

¿Cuál es la ganancia de velocidad teórica con respecto al micro no encauzado (sin Pipeline)?

280 ns / 90 ns = 3.11

Por lo tanto la versión con Pipeline es 3.11 veces mas rápida que la versión sin Pipeline, esto es para una cantidad considerable de instrucciones.