Trabajo Práctico 3

[6620] Organización de computadoras
Curso 2
Segundo cuatrimestre de 2020

Nombre y apellido	Padrón	E-mail	User Slack
Maria Sol Fontenla	103870	msfontenla@fi.uba.ar	Maria Sol Fontenla
Stephanie Izquierdo	104196	sizquierdo@fi.uba.ar	Stephanie Izquierdo
Agustina Segura	104222	asegura@fi.uba.ar	Agustina Segura

1 Introducción

El objetivo del trabajo practico es el de implementar algunas de las instrucciones de MIPS tanto en una arquitecura de cpu uniciclo y una arquitectura con implementacion de pipeline.

2 Criterios de diseño

- Dado que un jump se puede considerar como un branch que se toma, en las distintas implementaciones de pipeline se utilizo la salida de control de branch para tomar el salto y poder hacer un flush de los registros de pipeline.
- Cada instruccion a implementar se implemento en un .cpu aparte.
- A las instrucciones jr y jalr se las declaro del tipo R, con un op = 5 para poder asi definir las salidas de control necesarias para ejecutar la instruccion.

3 Comportamiento deseado

Instruccion j target: el comportamiento esperado para esta instruccion es que esta realice un salto incondicional de "target". Para su implementacion, primero se concatenaron los 26 bits mas bajos con los 4 bits mas altos del registro PC. Luego, a esta direccion de 28 bits se le concatenaron los dos bits mas bajos con 2 ceros. De esta manera, obtenemos la proxima direccion que debe cargarse en el program counter. Una vez que se obtuvo esta, con un multiplexor se elige entre esta direccion o el valor que se obtiene de las instrucciones de valor inmediato, y se elige la salida mediante la salida de control "Jump". Por ultimo, con la misma salida de control, decidimos si el proximo valor en el program counter sera PC + 4 o la direccion que obtuvimos de la instruccion j. Tambien se utilizo la salida de control "Branch" para poder realizar un flush de los registros de pipeline, los cuales almacenabanlas instrucciones siguientes a la instruccion j, y que no debian ejecutarse dado que el flujo del programa cambia.
Instruccion jr rs: el comportamiento esperado para esta instruccion es que se realice un salto incondicional hacia la direccion almacenada en rs.

- implementacion en uniciclo: para la implementacion de esta instruccion tambien se utilizo una salida de control ("JumpReg") para conectar con un multiplexor que decide si el proximo valor del program counter es PC + 4 (o en caso de haberse ejecutado un branch, la direccion de salto del mismo) o la direccion almacenada en rs. En caso de saltar, dado que se necesita el valor almacenado en el registro rs, se copia la salida del register bank que contiene el valor leido de este registro, y se lo conecta en la salida 1 del multiplexor mencionado previamente.

- implementacion en pipeline: en este caso, se decidio que el valor del registro se sume al valor del registro cero, dado que la instruccion se va ejecutando en distintas etapas. Al llegar al final de la ejecucion (etapa 5), se realiza un flush al igual que con la instruccion j, y se carga el pc con la direccion obtenida de la suma de rs + zero.

instruccion jarl rs,rd: el comportamiento esperado para esta instruccion es que se realice un salto incondicional hacia el registro rs, y almacenar la direccion en la cual se ejecuto el jarl + 4, la cual seria la siguiente instruccion a ejecutarse luego del jarl.

- Implementacion en uniciclo: para su implementacion se realizo lo mismo que para jr en unicilo, con la diferencia de que el resultado de PC + 4 se conecta con un multiplexor, y

en caso de estar ante un jarl, se activa una salida de control "PCCopy", y esta selecciona si el primer operando que ingresa a la ALU es lo que contiene el registro rs o el valor del pc. Para esta instruccion, PCCopy = 1 y este valor se suma con el registro cero, y se almacena su resultado en el registro rd.

- Implementacion en pipeline: para su implementacion se realizaron los flush correspondientes como en jr de pipeline, pero sin la utilizacion del flag branch. En este caso, seleccionamos con un multiplexor si el primer operando en la ALU cero o e registro que viene en la instruccion, con un selector "PCCopy" que indica si debemos copiar el PC + 4 o no, y este esta conectado con el PC. Ademas, con otro multiplexor seleccionamos si el primer operando que utiliza la AlU es el cero, o el registro rs. En caso de estar ante un jalr, la salida de este es cero (PCCopy = 1 y el mux no esta conectado a nada, obteniendo asi la salida cero), y con la salida de control "regWrite" copiamos en rd.

Deteccion de hazards: Mediante el uso de flush, pudimos detectar los hazards de control, limpiando las instrucciones que no debian ser ejecutadas debido a que se realizaba un salto incondicional. Ademas, los hazards de datos ya estaban siendo controlados por la unidad de forwarding ya implementada en pipeline.cpu

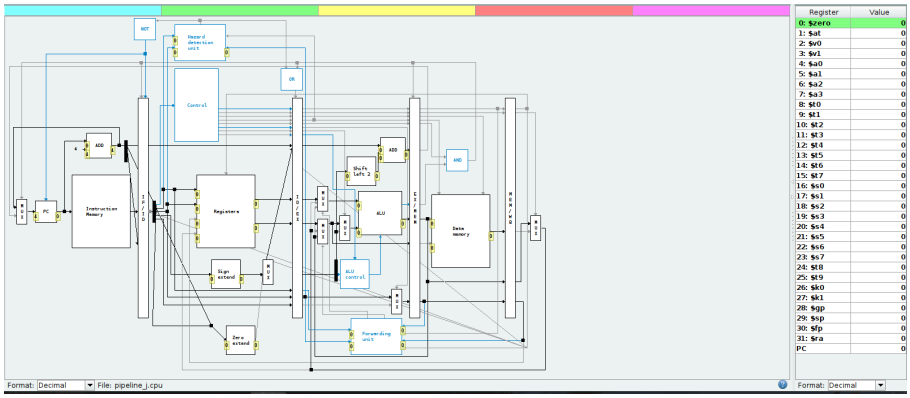

Figure 1: arquitectura pipeline con la implementacion de la instruccion jump

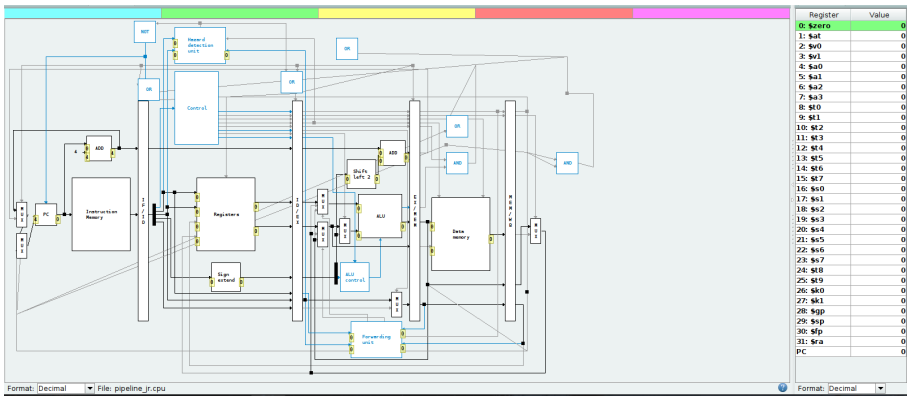

Figure 2: arquitectura pipeline con la implementacion de la instruccion jump register

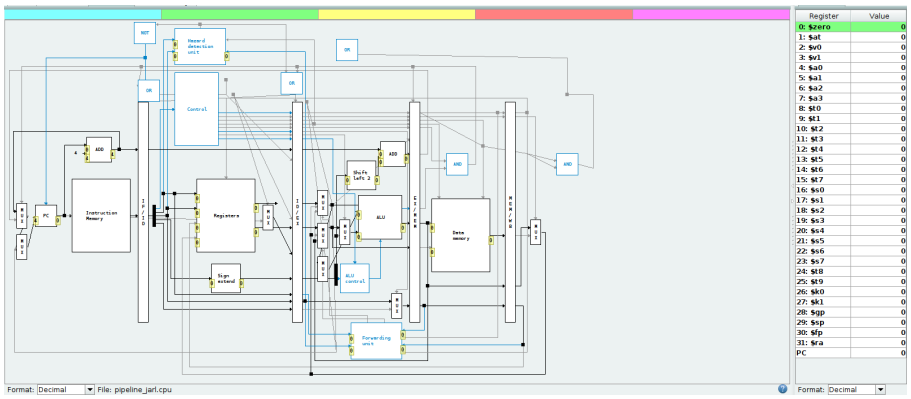

Figure 3: arquitectura pipeline con la implementacion de la instruccion jumpa and link register

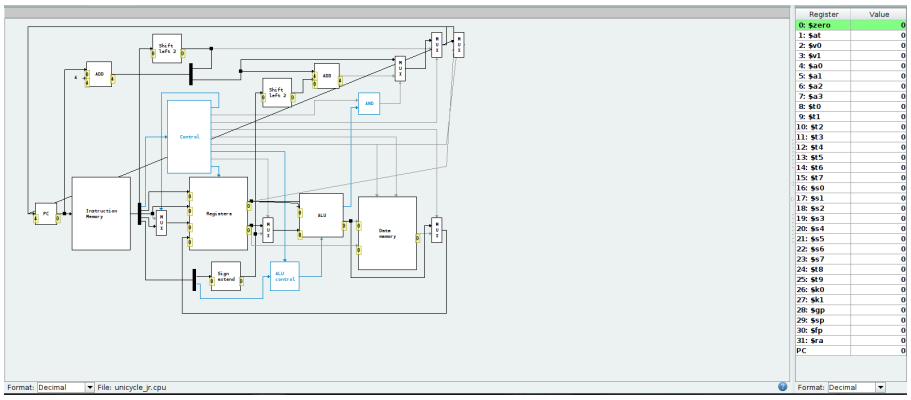

Figure 4: arquitectura uniciclo con la implementacion de la instruccion jump register

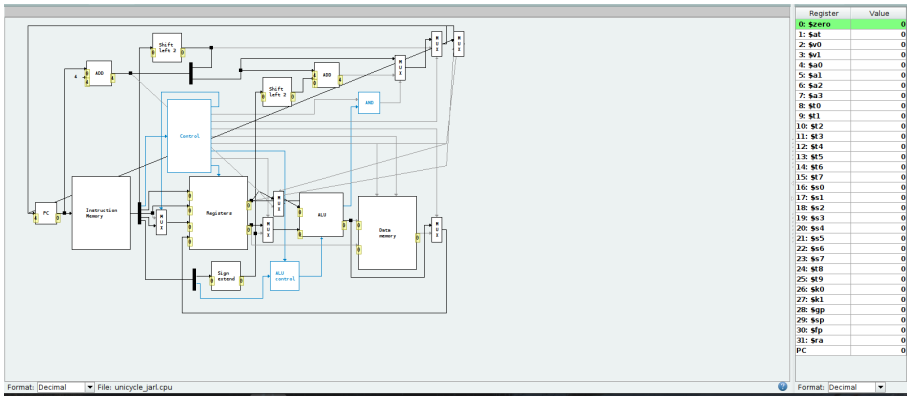

Figure 5: arquitectura uniciclo con la implementacion de la instruccion jump and link register

4 Conclusiones

Pudimos familiarizarnos con las arquitecturas uniciclo y pipeline y pudimos comprender mejor como funcionan las distintas partes de las mismas, pudiendo asi realizar cambios para agregar nuevas instrucciones al set que ya venia con estas arquitecturas.

Ademas, pudimos familiarizarnos con el programa DrMips para poder comprender mejor cada etapa de ejecucion en las distintas arquitecturas.