Trabajo Práctico 2: Infraestructura básica

Jimenez, Ruben, *Padrón Nro. 92.402*
rbnm.jimenez@gmail.com

Reyero, Felix, *Padrón Nro. 92.979*
felixcarp@gmail.com

Suárez, Emiliano, *Padrón Nro. 78.372*
emilianosuarez@gmail.com

Primera Entrega: *30/04/2015*

1er. Cuatrimestre de 2015
66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Se tuvo que modificar el conjunto de instrucciones de distintos diseños de datapath.

1. Objetivo

El objetivo de este trabajo es familiarizarse con la arquitectura de una CPU MIPS, específicamente con el datapath y la implementación de instrucciones. Para ello, se deberá agregar instrucciones a diversas configuraciones de CPU provistas por el simulador *DrMIPS*.

2. Introducción

El programa *DrMIPS* nos permite evaluar distintos diseños de datapath para procesadores *MIPS32*, al darnos la posibilidad de organizarlo como queramos. Podemos poner sumadores, multiplexores, extensores de signo y conexiones arbitrariamente.

También es posible modificar el conjunto de instrucciones. Además de la estructura lógica del DP, *DrMIPS* nos permite escribir programas simples y simular su ejecución en el DP, mostrando los valores que toman las diversas entradas y salidas de cada elemento.

3. Implementación

Se implementaron las siguientes funciones, sobre distintos datapaths:

– Instrucción *j* en el DP *pipeline.cpu*
– Instrucción *jr* (Jump Register) en el DP *unicycle.cpu*
– Instrucción *jr* en el DP *pipeline.cpu*
– Instrucción *jalr* (Jump and Link Register) en el DP *unicycle.cpu*
– Instrucción *jalr* en el DP *pipeline.cpu*]

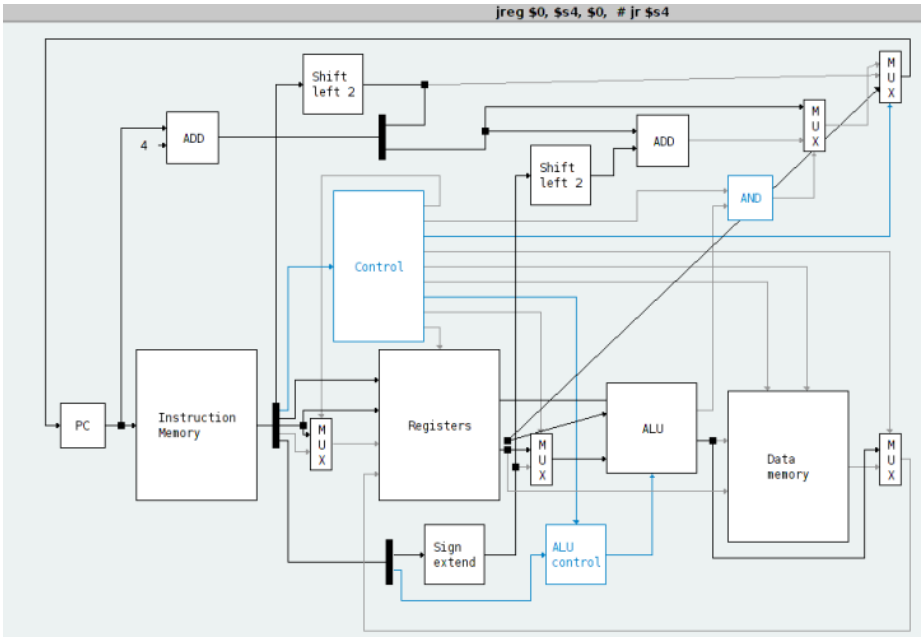

Figura 1: jr en DP uniciclo

Para poder ejecutar el jr (JumpRegister) en el CPU uniciclo, se agrego al MuxJump una entrada más “2”, a su vez el cable de control Jump también puede tomar 3 valores (0,1,2) para poder elegir la acción del jr a través del seteo en “2”. Además se agregó un componente mas: ForkRegJ que deriva los 32 bits del registro que se está leyendo hacia la tercera entrada (generada) en el MuxJump para asi poder ser redirigida al PC en caso de ser jr.

En el .set se agrego jreg como como instruccion y jr como pseudoinstruccion utilizando la anterior y seteandola de manera tal que cumpla con la especificacion de jr. En la seccion de control se activa el cable de Jump en el valor “2”.

Utilizando el .CPU del jr como base, se agrego al MuxDst una entrada más “2”, a su vez el cable de control RegDst también puede tomar 3 valores (0,1,2) para poder elegir la acción del jalr a través del seteo en “2” y una constante de 31 para seleccionar el registro $ra a escribir.

Tambien se modificó el ForkBranch y el MuxMem para poder escribir a

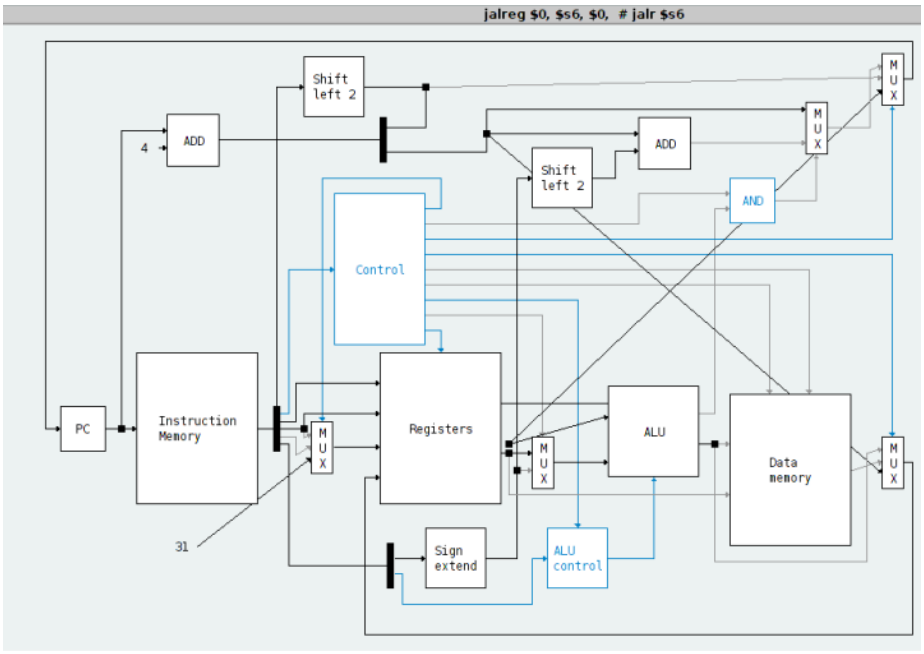

Figura 2: jr unicycle 1

través del MuxMem los datos del registro que debía seguir antes del salto que vienen del ForkBranch.

En el .set se agrego jalreg como como instruccion y jalr como pseudoinstruccion utilizando la anterior y seteandola de manera tal que cumpla con la especificacion de jr. En la sección de control se activa el cable de Jump, RegDst, MemToReg en el valor "2" y RegWrite en el valor "1" para permitir la escritura.

Para demostrar el correcto funcionamiento del jarl se muestra un ejemplo sencillo:

```
0 - addi $s1,$zero,16
4 - jalr $s1
```

Antes de la linea 0.

Una vez efectuada la suma y por generar el salto y linkear $ra a 8 (que sería la siguiente a 4):

Salto a 16, $ra = 8. Correcto.

Para la elaboracion de las instrucciones en pipeline se tuvo que considerar unicamente los hazards de control.

Debido que las instrucciones eran jump, jump register y jump and link register.

Para la elaboracion de la instruccion jump se uso el modelo de referencia que se encuentra en unicycle.cpu. Y para el control de hazards se hace un flush en el registro IF/ID, para ello el mutiplexor (muxjump) que tiene las entradas:

– 0: branch o PC+4

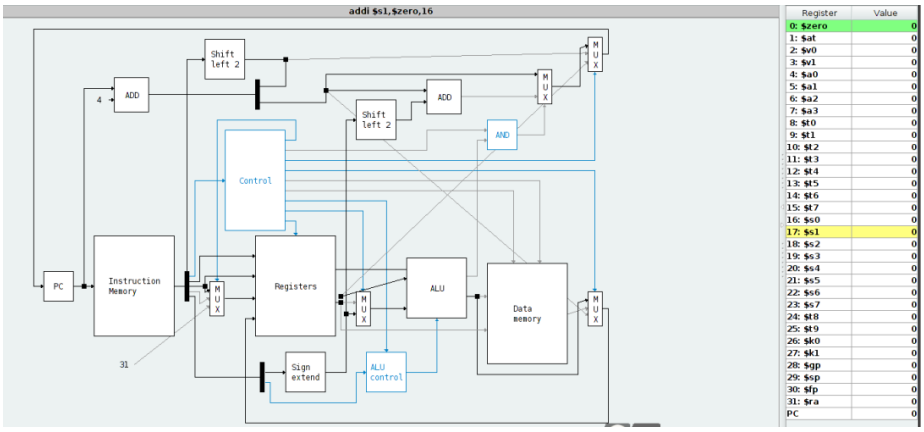

Figura 3: jr unicycle 2

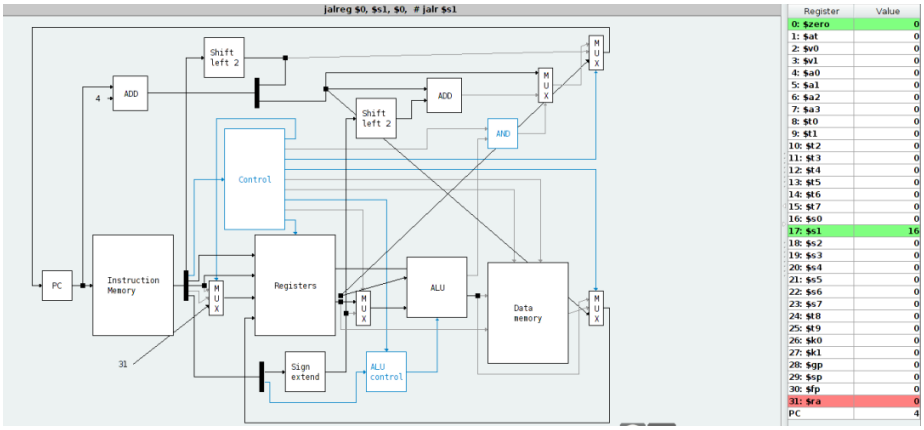

Figura 4: jr unicycle 3

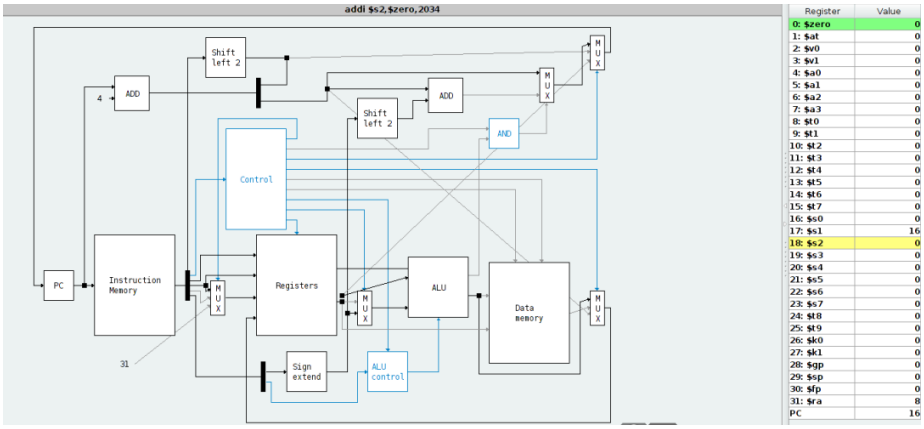

Figura 5: jr unicycle 4

– 1: jump
– 2: jump register y jump and link

La instruccion j,jr y jalr se realiza en el bloque de decodificacion ID, donde el valor del nuevo PC va hacia el multiplexor de saltos (muxjump) y ésta es habilitada mediante el la señal de control jump.

Como el salto se realiza en el bloque ID, se necesita descartar la siguiente instruccion Fetchada.

Dado que la instrucción branch hace un flush en los registros IF/ID,ID/EX,EX/MEM.

Para las instrucciones jump sólo es necesario hacer flush en el bloque IF, osea en el registro IF/ID. Para eso se necesito hacer un or entre la entrada de control jump que va hacia muxjump y la respectiva señal de flust de las instrucciones branch, dado que la entrada de control jump es de dos bit, entonces se usa un distribuidor de la señal de manera que:

– dos bits para el multiplexor de saltos (muxjump)
– bit numero 0 para la entrada 1 del or (orJump)
– bit numero 1 para la entrada 2 del or (orJump)

Para que se cumpla la condicion de flush sobre el registro IF/ID alguno de los bits tiene que tener valor 1, osea que sea 01 jump o 10 para que la salida del orJump jr y jalr.

Mientras que el branch o PC+4 que toma el valor 00, no produce el flush en el registro IF/ID

Los archivos *.set* y *.cpu*, pueden verse en dentro de la carpeta *dp/* del CD.

4. Conclusiones

Es importante destacar que es indispensable el dibujo previo del camino de datos anterior para el planeamiento cuando se va a modificar el datapath para poder entender el funcionamiento y generar el cambio necesario.

El simulador *DrMips* nos ayudó a entender mejor temas como la composición y el funcionamiento de un camino de datos, segmentación, codificación de la instrucción y el proceso de cada una de ellas.

5. Bibliografía

Referencias

[1] *MIPS-Light Instruction Set Summary*
http://web.stanford.edu/class/ee282h/projects/info/isa.html

[2] *Conditional and Unconditional Jumps*
http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/jump.html

[3] *MIPS Instructions*
http://web.cse.ohio-state.edu/~crawfis/cse675-02/Slides/MIPS%20Instruction%20Set.pdf

[4] *MIPS_ Arch2.ppt*
http://faculty.washington.edu/lcrum/Archives/TCSS372AF08/MIPS_Arch2.ppt

[5] *Tool to Support Computer Architecture Teaching and Learning*
https://bytebucket.org/brunonova/drmips/wiki/papers/cispee13_24.pdf