

Trabajo práctico #2: datapath

Santiago Fernandez, *Padrón Nro. 94.489*
fernandezsantid@gmail.com

Pablo Rodrigo Ciruzzi, *Padrón Nro. 95.748*
p.ciruzzi@hotmail.com

Horacio Martinez, *Padrón Nro. 94.926*
hmk142@hotmail.com

2do. Cuatrimestre de 2015
66.20 Organización de Computadoras – Práctica Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

26 de Noviembre, 2015

Resumen

En este trabajo práctico se verán modificaciones a distintos datapaths de una arquitectura MIPS, con el fin de agregar algunas instrucciones que no han sido implementadas en el mismo, y así poder familiarizarse con dicho concepto. La herramienta utilizada fue el DrMIPS version 2.0 [1][2].

Índice

1. Introducción	3
2. Desarrollo	3
2.1. Punto 1	3
2.1.1. Set de instrucciones modificado	3
2.1.2. Pruebas de sll y srl	4
2.2. Punto 2	5
2.3. Punto 3	5
2.4. Punto 4	5
3. Conclusiones	5
4. Referencias	6

1. Introducción

DrMIPS nos permite modificar el datapath de la arquitectura MIPS32 y el conjunto de instrucciones. Lo usaremos para agregar las siguientes instrucciones, tres de ellas, *sll*, *srl*, *jr* al datapath monociclo y las otras dos, *j* y *blt* al datapath pipeline.

Para lograrlo, agregaremos nuevos componentes al datapath y modificaremos el set de instrucciones, según sea necesario.

2. Desarrollo

2.1. Punto 1

En este ítem, se agregaron las instrucciones *sll* y *srl* al datapath monociclo. Para ello, no fue necesario modificar el datapath, simplemente bastó con agregar las instrucciones al set de instrucciones.

2.1.1. Set de instrucciones modificado

Se agregaron las siguientes líneas al campo *instructions* del set:

```
"sll": {
  "type": "R", "args": ["reg", "reg", "reg"],
  "fields": {"op": 0, "rs": "#2", "rt": "#3",
             "rd": "#1", "shamt": "#3", "func": 0},
  "desc": "$t1 = $t2 << $t3 = $t2 * 2^$t3"
},
"srl": {
  "type": "R", "args": ["reg", "reg", "reg"],
  "fields": {"op": 0, "rs": "#2", "rt": "#3",
             "rd": "#1", "shamt": "#3", "func": 2},
  "desc": "$t1 = $t2 >> $t3 = $t2 / 2^$t3"
},
```

Lo que esto hace, es definir dos instrucciones nuevas del tipo R, que reciben como argumento tres registros. Luego, el campo *func*, junto con el *aluop*, será el que referencie a estas instrucciones en la sección de control de la ALU, por lo tanto, al ejecutarlas, la entrada de la ALU será la especificada. Por último, debemos asociar esta entrada con la operación a realizarse, esto lo hacemos en la configuración de la ALU de la siguiente manera:

```
"alu": {
  ...
  "control": [
    ...
    {"aluop": 2, "func": 0, "out": {"0operation": 8}},
    {"aluop": 2, "func": 2, "out": {"0operation": 9}},
    ...
  ],
  "operations": {
    ...
```

```

        "8": "sll",
        "9": "srl",
        ...
    }
}

```

Esto le indica a la sección de control de la ALU, que ante el *aluop*=2 y *func*=0, debe ejecutar la operación 8, en este caso definida como *sll*, análogamente ocurre con *srl*.

2.1.2. Pruebas de sll y srl

Prueba 1:

```

li $t1,2
li $t2,6
sll $t3,$t1,$t2
li $t2,3
srl $t3,$t3,$t2
# 2 << 6 = 128, luego 128 >> 3 = 16

```

Prueba 2:

```

li $t1,3
li $t2,4
sll $t3,$t1,$t2
li $t2,5
srl $t3,$t3,$t2
# 3 << 4 = 48, luego 48 >> 5 = 1

```

Prueba 3:

```

li $t1,9
li $t2,1
srl $t3,$t1,$t2
li $t2,0
sll $t3,$t3,$t2
# 9 >> 1 = 4, luego 4 << 0 = 4

```

Prueba 4:

```

li $t1,16385
li $t2,18
sll $t3,$t1,$t2
li $t2,19
srl $t3,$t3,$t2
# 16385 << 17 = 262144, luego 262144 >> 19 = 0

```

Prueba 5:

```

li $t1,32767
li $t2,17
sll $t3,$t1,$t2
li $t2,17
srl $t3,$t3,$t2
# 16385 << 17 = -131072 (Recordar complemento a la base),
# luego -131072 >> 17 = 16385 (No hace extension de signo)

```

2.2. Punto 2

2.3. Punto 3

2.4. Punto 4

3. Conclusiones

Realizar este trabajo práctico nos permitió familiarizarnos con la arquitectura de CPU MIPS y con el simulador DrMiIPS. Nos sirvió para comprender como funciona el datapath monociclo y el pipeline, también para darnos cuenta que hay distintas formas de implementar una instrucción.

En algunos casos es posible agregar instrucciones sin modificar el datapath, como en el caso de las instrucciones *sll* y *srl*, pero generalmente requiere el agregado de nuevos componentes, lo que implica más costos, un aumento en la complejidad del datapath y posiblemente una disminución en la velocidad de ejecución. Por lo cual, a la hora de elegir si agregar una nueva instrucción, deben considerarse estas desventajas.

Por último, pudimos observar que en determinados casos, como en el de *blt*, no es posible agregar una nueva instrucción al datapath, sino que es necesario implementarla como una combinación de instrucciones ya existentes.

4. Referencias

- [1] DrMIPS, <https://bitbucket.org/brunonova/drmips/wiki/Home>.
- [2] DrMIPS, <https://github.com/brunonova/drmips>.

66:20 Organización de computadoras

Trabajo práctico 2: Data Path.

1. Objetivos

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 deberán agregar instrucciones a diversas configuraciones de CPU provistas por el simulador DrMIPS [1]

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 8), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo¹, y se valorarán aquellos escritos usando la herramienta \TeX / \LaTeX .

4. Recursos

Usaremos el programa DrMIPS [1] para configurar y simular el data path de un procesador MIPS [4], tanto unicycle como multiciclo.

5. Descripción.

5.1. Introducción

El programa DrMIPS nos permite evaluar distintos diseños de datapath para procesadores MIPS32, al darnos la posibilidad de organizarlo como queramos. Si bien sólo puede haber uno de algunos de los componentes del DP (como el registro de PC o la unidad de control), podemos poner sumadores, multiplexores, extensores de signo y conexiones arbitrariamente. También es

¹<http://groups.yahoo.com/group/orga6620>

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. El programa se puede conseguir en <https://bitbucket.org/brunonova/drmips/wiki/Home>, o se puede descargar para Ubuntu, ya sea desde el repositorio de Ubuntu (aunque la versión está desactualizada) o autorizando un repositorio externo (ver [2]).

5.2. Datapaths

El programa viene con algunos DP ya implementados, a saber:
Uniciclo:

- `unycycle.cpu`: El DP uniciclo por defecto.
- `unycycle-no-jump.cpu`: Variante más simple del DP uniciclo que no soporta la instrucción `j`.
- `unycycle-no-jump-branch.cpu`: Una variante aún más simple que no soporta `jump` ni `branch`.
- `unycycle-extended.cpu`: Una variante que soporta instrucciones adicionales, como multiplicación y división.

Multiciclo:

- `pipeline.cpu`: El DP de pipeline por defecto, implementa detección de hazards. Los DP de pipeline no soportan la instrucción `j` (salto).
- `pipeline-only-forwarding.cpu`: Variante del DP de pipeline que implementa forwarding pero no genera stalls (genera resultados incorrectos).
- `pipeline-no-hazard-detection.cpu`: Otra variante que no hace hazard detection de ninguna manera (genera resultados incorrectos).
- `pipeline-extended.cpu`: Una variante que soporta instrucciones adicionales, como multiplicación y división, como `unycycle-extended.cpu`.

5.3. Instrucciones a implementar

1. Implementar las instrucciones `sll` y `srl` (Shift Left Logical y Shift Right Logical) en el DP `unycycle.cpu`.
2. Implementar la instrucción `j` en el DP `pipeline.cpu`.
3. Implementar la instrucción `jr` (Jump Register) en el DP `unycycle.cpu`.
4. Implementar la instrucción `blt` (Branch if Less Than) en el DP `pipeline.cpu`. ¿Se puede hacer nativamente, o hay que agregar pseudoinstrucciones? ¿Por qué?

Para todos los casos, verificar que no se produzcan hazards de control o de datos.

6. Implementación.

Los archivos antes mencionados, así como los archivos `.set` que contienen los datos del conjunto de instrucciones, están en formato JSON [3], y se pueden modificar con un editor de texto. Se sugiere uno que pueda hacer *color syntax highlighting*, como el `gedit` que viene con el Ubuntu. La explicación de los formatos se encuentra en el archivo `configuration-en.pdf` que se distribuye con el programa.

7. Pruebas

En todos los casos debe verificarse que la instrucción se ejecute correctamente. Esto implica que el PC tome el valor deseado, y además que en el caso del DP multiciclo no se produzcan hazards, como ser la ejecución de la instrucción siguiente al salto, o en el caso de utilizar el valor de un registro, que éste tenga el valor correcto.

8. Informe.

Se debe entregar:

- Informe describiendo el desarrollo del trabajo práctico.
- Capturas de pantalla de los DP modificados.
- Programas de prueba.
- CD conteniendo los DP, los programas de prueba y los conjuntos de instrucciones usados en cada caso.
- Este enunciado.

9. Fechas de entrega.

- Primera entrega: Jueves 19 de Noviembre.
- Revisión: Jueves 26 de Noviembre.
- Vencimiento: Jueves 3 de Diciembre.

Referencias

- [1] DrMIPS, <https://bitbucket.org/brunonova/drmips/wiki/Home>.
- [2] PPA de Bruno Nova, <https://launchpad.net/~brunonova/+archive/ubuntu/ppa>.
- [3] ECMA-404 The JSON Data Interchange Standard, <http://www.json.org/>.
- [4] “Computer organization and design: the hardware-software interface”, John Hennessy, David Patterson. Capítulo 5.