

Trabajo Práctico 2: Data path y pipeline

José Ignacio Sbruzzi, *Padrón Nro. 97.452*

jose_sbruzzi@hotmail.com

Leandro Huemul Desuque, *Padrón Nro. 95.836*

desuqueleandro@gmail.com

2do. Cuatrimestre de 2016

66.20 Organización de Computadoras

Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Se introdujeron modificaciones sobre datapaths para agregar ciertas funcionalidades a los mismos. El objetivo del trabajo es la práctica y el aprendizaje del funcionamiento del datapath.

1. Introducción

El objetivo del presente trabajo es introducir una serie de modificaciones a datapaths que vienen con el software DrMips. De esta forma, nos familiarizamos con los conceptos fundamentales de la Organización de Computadoras.

2. Desarrollo

2.1. Modificación del DP monociclo

La modificación introducida fue una nueva instrucción denominada "load byte unsigned" (mnemónico lbu), que en vez de cargar un word como lw, carga un byte, y no hace extensión de signo: los bits a la izquierda del último son siempre 0.

2.1.1. Procedimiento de la modificación

Luego de notar que incluso al recibir direcciones desalineadas la memoria carga una palabra alineada (es decir, por ejemplo, las direcciones 0, 1, 2 y 3 hacen que se cargue la palabra en la dirección 0), se decidió hacer que lbu funcionara exactamente igual que lw, pero removiendo 3 bytes, haciendo parecer de esta manera que se carga un solo byte.

Para esto, se agregó una señal a la unidad de control llamada `çortar`. La señal impacta el sistema por medio de un multiplexor que define si se guardará en el register file la palabra obtenida de memoria o una versión modificada que mantiene un sólo byte. El word modificado se calcula incluso si la señal `çortar`.^{es} 0.

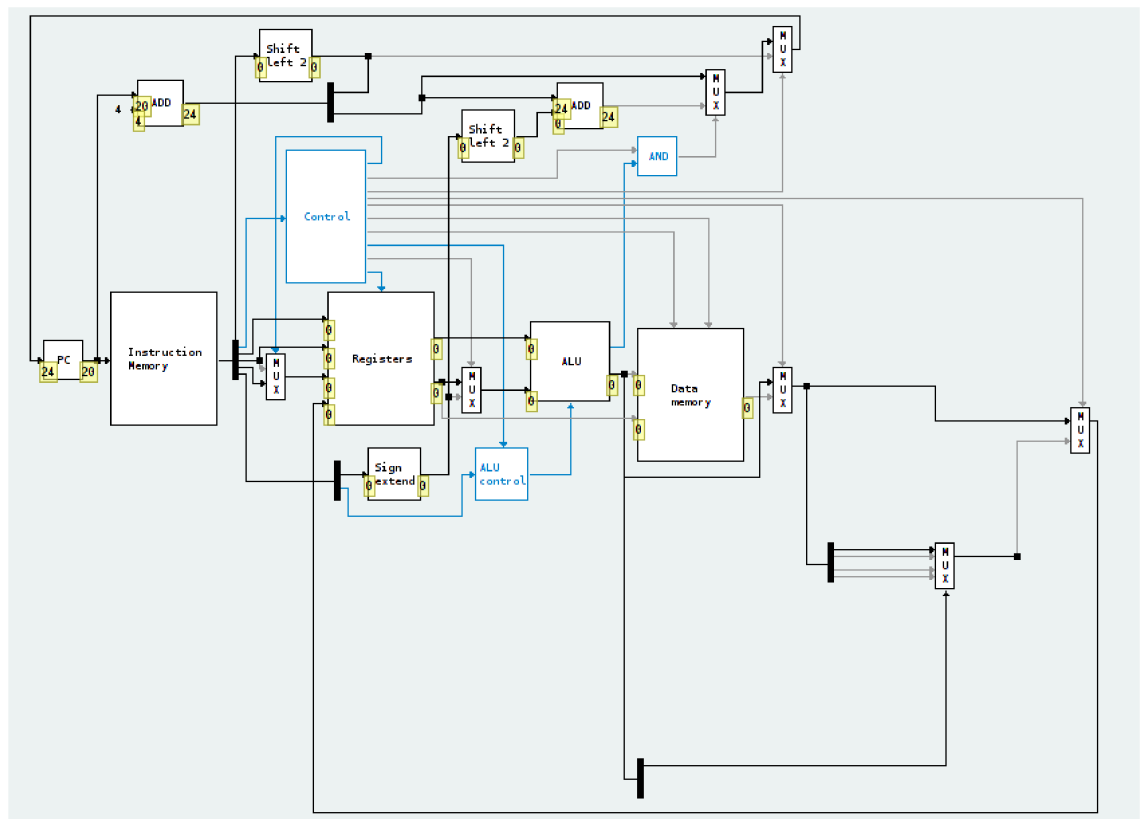


Figura 1: Estado final del datapath monociclo

La modificación del word es llevada a cabo por medio de un *Distributor* que divide el word en sus 4 bytes. Luego, por medio de un multiplexor se realiza la elección de uno de esos 4 bytes. Finalmente, el byte seleccionado se concatena a 24 ceros, quedando así el byte seleccionado en la parte menos significativa del word.

2.1.2. Pruebas

A continuación se muestra el código assembler utilizado como prueba. El CPU pasa la prueba si en el registro t0 queda el valor 0x000000aa, el registro t2 queda en 0x000000bb, el t3 en 0x000000cc y el t4 en 0x000000dd.

```

1 .data
2 prueba: .word 0xAABBCCDD, 0x11223344, 0x22222222, 0x33333333
3
4 .text
5
6 mostrar:
7     la $t5, prueba
8     lbu $t0, 0($t5)
9     lbu $t1, 1($t5)
10    lbu $t2, 2($t5)
11    lbu $t3, 3($t5)

```

Figura 2: Prueba utilizada para verificar el correcto funcionamiento de la modificación

2.2. Modificación del DP pipeline: jal

Se decidió agregar la instrucción jal antes de la bgezal para aprovechar el hardware añadido. Inicialmente se notó que los branches se resuelven en la etapa MEM: es allí donde se modifica el PC. El guardado de la dirección de retorno en el registro ra puede realizarse en la etapa WB, ya que jal -tal como hace beq- "flushea" todos los registros interetapa, con lo cual al ejecutarse el WB del jal, la instrucción que sigue estará en la etapa IF.

Con el objetivo de hacer la menor cantidad de modificaciones posibles, no se cambia la condición del salto: se siguen comparando dos registros, solo que esta vez siempre se comparan los registros zero. Así, *jal offset* equivale a *beq zero, zero, offset* con la única diferencia de que se guarda PC+4 en RA.

Entonces, se agregó una señal "GuardarPC" a la unidad de control, la cual se guarda en los registros interetapa siguientes a ID. Además se agregó NewPC (presente en ID/EX) a los registros EX/MEM y MEM/WB.

Se Agregan también dos multiplexores en la etapa WB controlados por la señal "GuardarPC": uno permite fijar la señal WriteAddress en 31 (correspondiente a RA) y otro que permite fijar WriteData en NewPC.

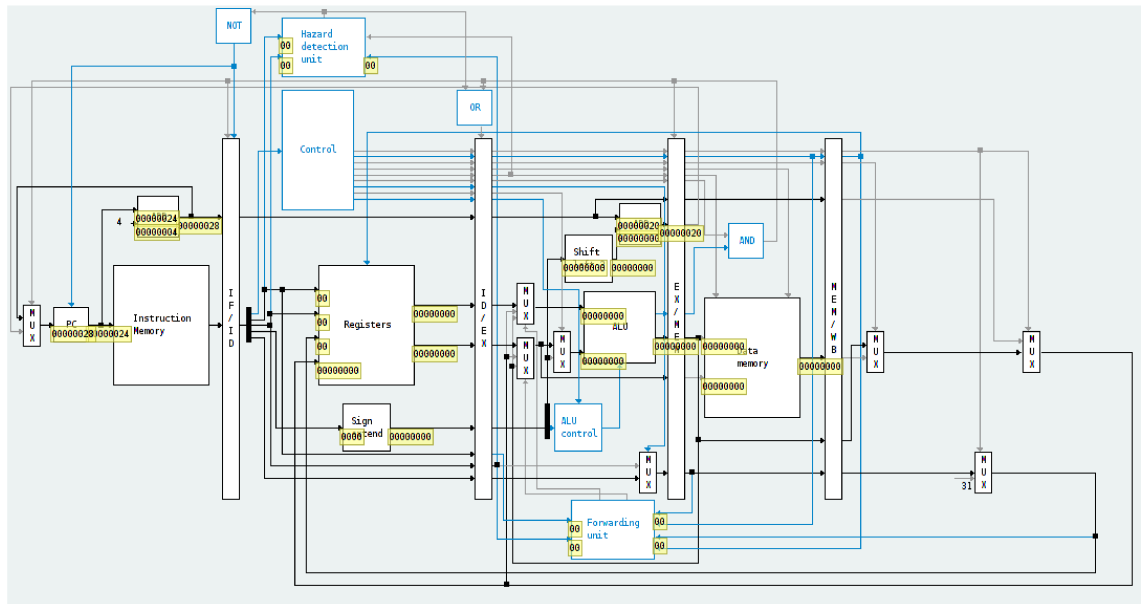


Figura 3: Datapath pipeline con jal

2.2.1. pruebas

La prueba utilizada se expone a continuación. Si es exitosa, t1 termina en 0, t0 termina en 1 y ra en 0c.

```

1 .data
2 prueba: .word 0xAABBCCDD, 0x11223344, 0x22222222, 0x33333333
3
4 .text
5
6 mostrar:
7     add $t1, $zero, $ra
8     addi $t0, $zero, 1
9     jal funcioncita
10    addi $t0, $zero, 2
11 funcioncita:
12    add $t2, $zero, $ra
13

```

Figura 4: Datapath pipeline con jal

2.3. Modificación del DP pipeline: bgeal y bgezal

Una tarea solicitada fue la implementación de una instrucción BGEZAL. En vez de implementar bgezal, se decidió implementar bgeal, y hacer que bgezal sea una pseudoinstrucción. La pseudoinstrucción se traduce como una única instrucción (se cambia uno de los argumentos por el registro zero), con lo que no hay pérdidas de velocidad y se gana versatilidad.

La compuerta AND de la etapa MEM reúne la información proveniente de las señales Branch (proveniente de la unidad de control, fijada en la instrucción) y Zero (proveniente de la ALU, es 1 si el resultado de la ALU es 0). La ALU implementa la operación slt, con lo cual, por medio de modificaciones de software a partir de la instrucción JAL se puede obtener BGEAL. Se aprovechó que slt deja el bit Zero en 1 si la condición "mayor o igual" se cumple (pero no "menor estricto").

2.3.1. modificación bgeal

Las modificaciones respecto de JAL, así, no se hicieron sobre el archivo que describe el hardware (.cpu) sino sobre el que describe el software (.set). Se tomó la línea de la sección "instructions" que describe beq y sólo se reemplazaron el opcode (por uno nuevo, distinto del de JAL) y el nombre mismo de la instrucción.

En la sección "control" se copió la línea que corresponde al beq, pero se cambió el valor de RegWrite y GuardarPC a 1 (tal como se había hecho para JAL) y se asignó un nuevo ALUOp. El nuevo ALUOp se hizo corresponder con la operación slt en la sección ".alu" del mismo archivo.

Luego de hacer pruebas se detectó que el registro RA se guardaba siempre, se dé o no el salto, con lo que se hizo necesario almacenar el valor de salida del componente AndBranch (que registra si realmente se llevó a cabo el branch) en el registro interetapa MEM/WB, y luego usar una compuerta AND y un multiplexor para filtrar el valor de la señal RegWrite.

2.3.2. modificación bgezal

Se agregó la pseudoinstrucción bgezal, que es bgeal pero con su segundo argumento en el registro 0.

2.3.3. pruebas

Se utilizaron tres pruebas, una de las cuales está representada a continuación. Las otras dos son variantes en las que se cambia el -5 por un número positivo y por 0. De ser un número positivo o 0, al final de la ejecución RA es 0c y t0 es aaa, de lo contrario, RA es 0 y t0 es bbb.

2.3.4. mejoras posibles

Una vez implementada bgeal, se podría implementar jal como una pseudoinstrucción y de esta manera se reduciría el número de instrucciones que implementa el datapath.

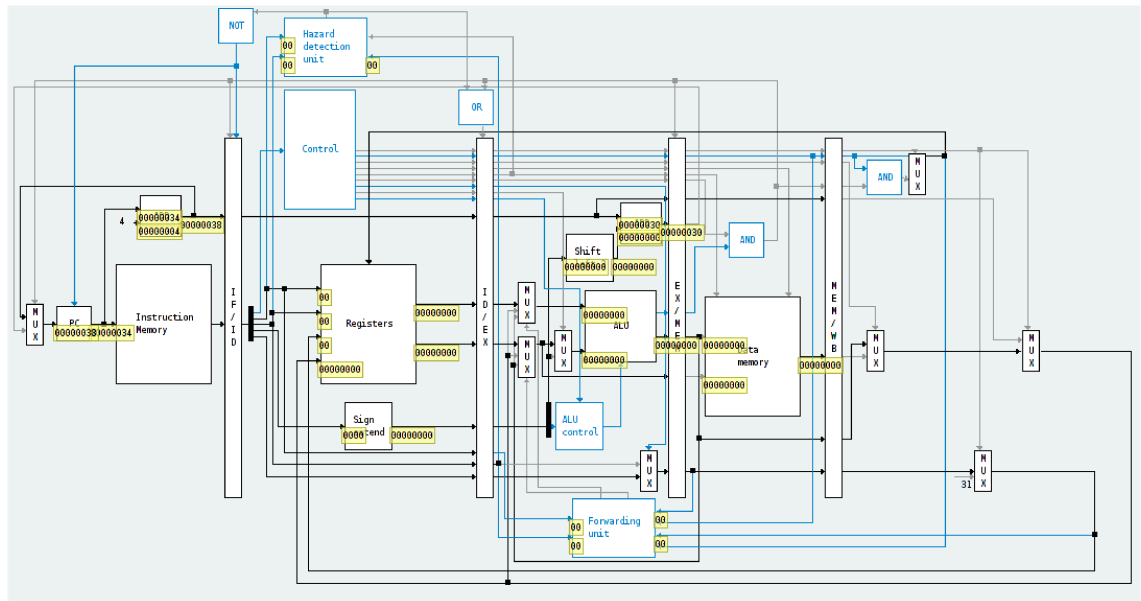


Figura 5: Datapath pipeline con las modificaciones necesarias para bgeal

```

1 .data
2 prueba: .word 0xAABBCCDD, 0x11223344, 0x22222222, 0x33333333
3
4 .text
5
6 mostrar:
7     addi $t0, $zero, 0x0AAA
8
9     addi $t1, $zero, -5
10    bgeal $t1, funcioncita
11    addi $t0, $zero, 0x0BBB
12 funcioncita:
13    add $t2, $zero, $ra
14

```

Figura 6: Prueba de bgeal.

2.4. Modificación del DP pipeline: sb

La instrucción sb (store byte) almacena el byte más bajo del registro pasado en memoria. El problema más importante que presenta esta instrucción es el hecho de que DrMips maneja la memoria de a words: es imposible guardar o cargar un único byte. Así, la instrucción debería cargar un word, superponerle el byte a guardar, y luego guardar ese word a memoria. Otro problema es que es imposible guardar y cargar datos a memoria en el mismo ciclo, con lo que la instrucción debería ocupar la etapa MEM durante dos ciclos. Esto va en contra de la idea central del pipeline, con lo que se hace necesario implementar sb como una pseudoinstrucción que tenga al menos dos instrucciones: una de carga y otra de almacenamiento. Las operaciones intermedias en las que se transforma el word también son complicadas, ya que requieren la aplicación de varias máscaras de bits.

```
#define SB(Rs,Imm,Rt) \
    addi at, zero, 3      \      //máscara del desplazamiento (en bytes)
    addi Rt, Rt, Imm      \      //formar dirección final
    and at, at, Rt        \      //en at están los últimos 2 bits, valores 0,1,2,3
    add at, at, at         \      //multiplicación por 2, at tiene 0,2,4,6
    add at, at, at         \      //multiplicación por 2, at tiene 0,4,8,12
    add at, at, at         \      //multiplicación por 2, at tiene 0,8,16,24
    sllv Rs, Rs, at        \      //desplazar el byte a guardar
    addi at,Imm            \      //carga inmediato al registro para hacer la resta
    sub Rs,Rs,at           \      //resta del inmediato, Rs vuelve al estado anterior

    lwob at, Imm(Rt)       \      //carga s/mascara
    or at, at, Rs           \      //formar el word final
    sw at, Imm(Rt)         \      //guardado del word

    //deshacer las modificaciones sobre Rs

    addi at, zero, 3      \      //máscara del desplazamiento (en bytes)
    addi Rt, Rt, Imm      \      //formar dirección final
    and at, at, Rt        \      //en at están los últimos 2 bits, valores 0,1,2,3
    add at, at, at         \      //multiplicación por 2, at tiene 0,2,4,6
    add at, at, at         \      //multiplicación por 2, at tiene 0,4,8,12
    add at, at, at         \      //multiplicación por 2, at tiene 0,8,16,24
    srlv Rs, Rs, at        \      //deshacer el desplazamiento
    addi at,Imm            \      //carga inmediato al registro para hacer la resta
    sub Rs,Rs,at           \      //resta del inmediato, Rs vuelve al estado anterior
```

Figura 7: Implementación de sb como macro

La idea es remover el enmascarado de la palabra cargada: construir una instrucción que cargue la palabra entera, pero que ponga en 0 el byte que luego se va a escribir. La instrucción se llamaría lwob (load without byte).

Se hace también necesario implementar sllv y srlv. Disponer de un solo registro auxiliar supone un problema muy grave, ya que se hace necesario casi duplicar la cantidad de instrucciones necesarias.

Es imposible generar la pseudoinstrucción sin lwob porque todos los cálculos de la máscara requerirían al menos un registro auxiliar más.

Una mejora posible a la implementación propuesta es agregar sll, con lo que se reduciría en 4 el número de instrucciones.

2.4.1. Implementación de slv y srlv

La ALU ya soporta las operaciones slr y sll, con lo que sólo es necesario modificar los archivos .set. Se agregaron las instrucciones en la sección "instructions", agregando nuevas líneas a partir de las de add. Se les asignaron campos "func" nuevos, y se hizo la relación entre esos campos y las operaciones correspondientes en la sección .alu".

2.4.2. Implementación de lwob

Se agregó a la unidad de control la señal lwob, que se almacena en los tres registros interetapa siguientes. En la etapa de memoria se agregó un circuito combinacional que a partir de los dos bits más bajos del campo Result”, que indica la dirección de donde provienen los datos, calcula el word final removiéndole el byte correspondiente. Finalmente, mediante un multiplexor al que ingresa la señal lwob se elige si usar el word calculado o el usual.

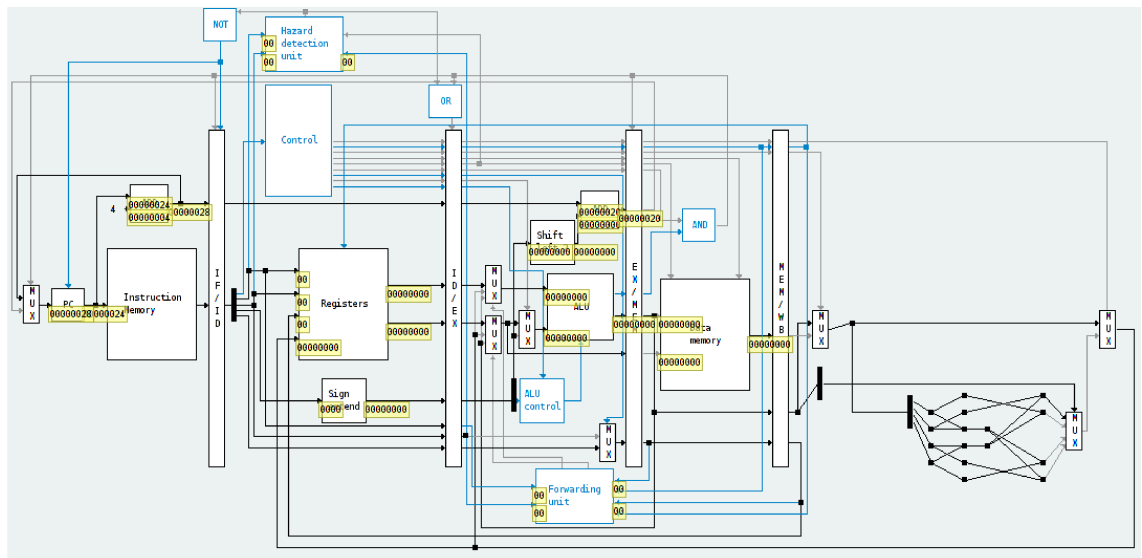


Figura 8: Implementación de lwob en el datapath

Prueba de lwob

La prueba, de ser exitosa, deja t1 en 0x11223300, t2 en 0x11220044, t3 en 0x11003344, y t4 en 0x00223344. Nótese que en este caso se utilizó un endianness distinto del correspondiente al lbu. Se considera esto válido porque, al ser distintos datapath, no hay conflictos con el endianness de las instrucciones.

2.4.3. Implementación de sb

Se modificó la macro mostrada más arriba y se incluyó la pseudoinstrucción en el archivo .set correspondiente. Como el DrMips no está preparado para pseudoinstrucciones de la forma *sb rs, Imm(rt)*, se tuvo que usar la forma *sb rs, rt, Imm*.


```

1 .data
2 prueba: .word 0xAABBCCDD, 0x11223344, 0x22222222, 0x33333333
3
4 .text
5
6 mostrar:
7     la $t0, prueba
8     addi $t0, $t0, 4
9     lwob $t4, 3($t0)
10    lwob $t3, 2($t0)
11    lwob $t2, 1($t0)
12    lwob $t1, 0($t0)
13

```

Figura 9: Prueba de lwob

2.4.4. Prueba de sb

El resultado de la prueba es: t1 debe terminar en 0x88 t2 en 55223344 t3 en 55663344 t4 en 55667744 t5 en 55667788.

Se probó cada caso por separado para encontrar todos los errores de la pseudoinstrucción. Una limitación que se encontró es que la pseudoinstrucción no funciona si se le pasa el registro zero como uno de sus argumentos, ya que es necesario escribir sobre él.

3. Resultados

Además de las modificaciones y los resultados de las pruebas, ambos ya expuestos, en todos los casos puede notarse que implementar la instrucción en el datapath disminuye muchísimo la cantidad de instrucciones en contraste con la opción de crear una pseudoinstrucción.

4. Conclusiones

Al manejar el datapath y modificarlo, se conoció en detalle su funcionamiento, y se produjo una familiarización con el mismo que no hubiera sido posible de haberse estudiado únicamente como algo teórico.

Una dificultad muy grande que se tuvo en el desarrollo del trabajo fue el hecho de que el software no tiene una interfaz gráfica para modificar los circuitos. Esto incluso influyó en cierta medida algunas decisiones de diseño.

Resulta compulsorio el desarrollo de una herramienta que permita editar los archivos .cpu más fácilmente.

```

1 .data
2 prueba: .word 0xAABBCCDD, 0x11223344, 0x22222222, 0x33333333
3
4 .text
5
6 mostrar:
7     la $t0, prueba
8     addi $t0,$t0,4
9
10
11
12     addi $t1, $zero, 0x55
13     sb $t1, $t0, 3
14     lw $t2, 3($t0)
15
16     addi $t1, $zero, 0x66
17     sb $t1, $t0, 2
18     lw $t3, 2($t0)
19
20     addi $t1, $zero, 0x77
21     sb $t1, $t0, 1
22     lw $t4, 1($t0)
23
24     addi $t1, $zero, 0x88
25     sb $t1, $t0, 0
26     lw $t5, 0($t0)
27

```

Figura 10: Prueba de sb

Referencias

- [1] Gardner, Martin. “Mathematical Games - The fantastic combinations of John Conway’s new solitaire game ‘life’ ” Scientific America, 223. pp. 120-123. ISBN 0-89454-001-7. Archivado del original en: https://ddi.cs.uni-potsdam.de/HyFISCH/Produzieren/lis_projekt/proj_gamelife/ConwayScientificAmerican.htm

Consultado en septiembre 2016.

- [2] Sitio web de GXemul <http://gxemul.sourceforge.net/>

Consultado en septiembre 2016.