Practica 7. Unidad de Control y ALU Control

Paul Vazquez
Ingenieria en Sistemas Digitales y Robotica
Tecnologico de Monterrey
Laboratorio de Arquitectura de computadoras A00819877

Rodolfo Cuan
Ingenieria en Sistemas Digitales y Robotica
Tecnologico de Monterrey
Laboratorio de Arquitectura de computadoras A01233155

1. Introducción

La unidad de control es uno de los bloques más importantes de la arquitectura MIPS, es responsable de enlazar y coordinar distintos bloques dentro del mismo sistema, por medio de lineas de control [1]. El bloque recibe instrucciones como entrada, y las interpreta para luego convertirlas a señales [2]. Si se recuerda en la arquitectura MIPS las instrucciones se manejan en unidades de 32 bits, donde la unidad de control utiliza los primeros 6 bits más significativos para generar las señales [3]. Estos bits tienen el nombre de Opcode y las 9 señales las cuales esta unidad es responsable de controlar son:

- Register Destiny
- Jump
- Branch
- Memory Read
- Memory to Register
- Memory Write
- ALU Source
- Register Write
- ALU operation

Las señales de control se conectan con la memoria de instrucciones, registros, y la memoria de datos, como se puede observar en la Figura 8 en el Apéndice A. Específicamente, la señal de ALU operation, determina que operación realizara la ALU [3]. Esta señal esta conectada directamente con el bloque ALU control, segundo bloque a realizar en esta practica, que procesa la señal para indicarle a la ALU que operación ejecutar junto con la sección de la instrucción (Fig. 1).

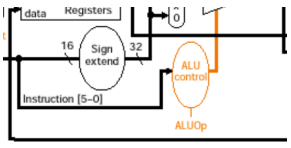

Figure 1: Esquemático del bloque ALU Control. Imagen de [4].

2. Desarrollo

Tomando como referencia el diagrama general del MIPS, empezamos con el diseño y construcción de las instrucciones de ensamblador.

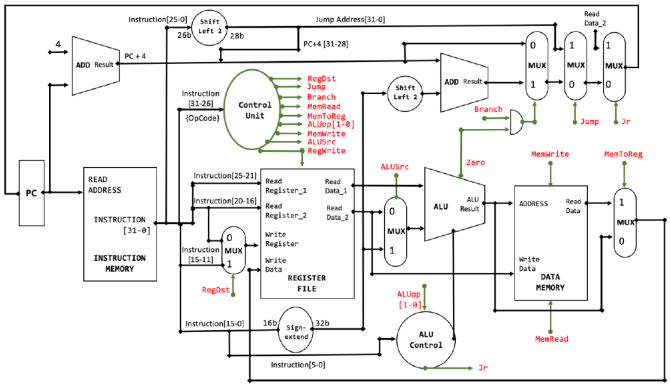

Figure 2: Esquemático del diagrama general del MIPS.

Tomando en cuenta que todas las instrucciones son de 32 bits y se leen en de la memoria de instrucciones, en al posición indicada por el Program Counter. Independientemente del tipo de instrucción R o I, los 6 bits mas significativos se introducen a la unidad de control para decodificarla y encender las lineas de control necesarias para ejecutarla. El set completo de instrucciones que tendrá nuestro procesador es el siguiente:

- ADD
- SUB
- AND
- OR
- SLT
- JR
- LW
- BEQ
- J
- ADDI
- ORI
- LUI

Las primeras 6 instrucciones de esta lista se denominan del formato 'R' y se describirán en la siguiente sección.

Como se menciono anteriormente, hay otro elemento que ayuda a decodificar las instrucciones llamado ALU CTRL. A este componente se le introducen los 6 bits menos significativos de la instrucción a ejecutar y evaluando ALUOP determina la operación a realizar.

2.1. Instrucciones tipo R

Las instrucciones ADD, SUB, AND, OR, SLT y JR, pertenecen a la categoría de instrucciones del tipo R porque todas tienen en común que utilizan dos registros de entrada(rs, rt) y un registro de destino (rd) con el resultado (con el caso especial de JR que mencionaremos mas adelante). Dado que su ejecución depende casi completamente de la ALU, todas comparten el OPCODE 000 000 y cuentan con un campo especial llamado ”function” de 6 bits que se introduce a la ALU para indicarle que operación ejecutar. El formato de este tipo de instrucciones es el siguiente:

Figure 3: Formato de instrucciones tipo R.

Donde op es Opcode, rt es el primer registro, rt el segundo registro, rd el registro destino, shamt la cantidad de bits a recorrer y funct el function code.

En resumen la lógica que sigue el procesador para ejecutar estas instrucciones es:

1) Determinar que el OPCODE es 000 000 por lo que la instrucción es del tipo R.
2) Decodificar los 6 bits del function code para determinar que operación debe realizar la ALU.
3) Generar el ALU CTRL code que le corresponde.
4) Encender las lineas de RGDEST y REGWRITE para guardar los resultados en el registro de destino.

Recordando la implementación del ALU de la Practica # 3, tenemos la siguiente relación entre ALU CTRL codes y operaciones en la ALU:

Tabla 2.1. Operaciones de la Unidad Aritmética Lógica

ALU Control Lines	Función	Operación
000	And	Result = a and b
001	Or	Result = a or b
010	Add	Result = a + b
011	Mov	Result = a
100	B upper	Result = b[15:0] & x”0000”
110	Substract	Result = a - b
111	Set less than	Consultar más adelante

Figure 4: Codigos de CTRL ALU.

Por lo tanto, la relación entre los function codes y ALU CTRL codes queda de la siguiente manera en nuestra implementación:

	FunctionCode	OPERACION EN ALU	ALU CTRL
ADD	100000	SUMA	010
SUB	100010	RESTA	110
AND	100100	AND	000
OR	100101	OR	001
SLT	101010	SLT	111
JR	1000	SUMA	010

Figure 5: Codigos de CTRL ALU para instrucciones tipo R.

2.1.1. Instruccion JR. Hay un caso interesante dentro del grupo de instrucciones tipo R, la instrucción JR. Dicha instrucción requiere encender la linea de control JR, que sale del componente ALU Control y carga el valor del registro 2 al Program Counter. Para este caso la ALU realiza una suma entre el Registro2 y el valor de ”0000_0000”, y se trata de guardar en el Registro 0, pero como este no es editable, solamente se carga el valor del Registro 1 en el Program Counter.

2.2. Instrucciones tipo I

El otro tipo de instrucciones se llaman instrucciones tipo I, porque requieren un field ”inmediato”. Este field inmediato contiene algún valor que se utiliza directamente en la ALU para calcular la ubicación de memoria de origen o destino.

Figure 6: Fields para instrucciones tipo I.

2.2.1. Instruccion SW y LW. Las dos instrucciones que interactúan con la memoria de datos (RAM) son las del tipo LOAD WORD y STORE WORD (LW, SW) y ambas requieren utilizar la linea de ALU SRC para que se sume el offset a la dirección base contenida en un registro. En el caso de SW, se enciende además la linea de memory write para que se escriba en memoria, y para el caso de LW memory read para que se lea la memoria.

Tomando en cuenta que el field de immediate es de 16 bits, se pueden acceder a ±2$\hat{1}$5 o 32,768 bytes de la memoria RAM.

2.2.2. Instruccion BEQ y J. Ambas instrucciones comparten la característica de que modifican el valor del Program Counter para saltar a distintas secciones del programa. Para el caso de Branch Equal (BEQ), con el opcode de 0x03, primero se realiza una resta en la ALU entre dos registros rs y rt. Si son iguales el resultado de la resta debe dar ’0’, por lo que se encendería la linea de Zero de la ALU. Se realiza una operación de AND entre esta linea y la linea de control Branch para que cuando sean ambas condiciones verdaderas, se cargue el valor de los 16 bits en el Program Counter.

La instrucción de Jump (J) es similar, con una linea de control especial llamada Jump, pero sin el paso intermedio

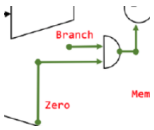

Figure 7: AND entre Branch y Zero.

de la comparación entre los registros, se carga de manera directa el valor del field.

2.2.3. Instruccion ADDI, ORI, LUI. Estas tres instrucciones son muy similares a las del tipo R, pues realiza operaciones lógicas y carga de bits. La diferencia principal es que en este caso en vez de realizarse entre dos registros, se realiza entre un registro y un valor inmediato de 16 bits. Por ello se requieren encender las lineas de ALU SRC y REG WRITE, para que se tome el valor del field en la ALU y para que se quede guardado en el registro de destino. Los opcodes son 0x08, 0x0D y 0x0F. Cada una tiene una ALUOP correspondiente a lo que tiene que realizar la ALU, suma para ADDI, OR para ORI y BUPPER para LUI. Cabe mencionar que LUI es la única instrucción de nuestra implementación que utiliza esta operación en la ALU.

Todas las instrucciones anteriores nos generaron la siguiente tabla:

3. Resultados y Discusión

El correcto funcionamiento de ambos bloques fue comprobado por medio de test benches y la comparación de los resultados con una tabla de verdad desarrollada.

3.1. Unidad de Control

En las indicaciones de la practica se especifican las instrucciones las cuales nuestro MIPS debe interpretar. Es por eso que nuestro test bench se diseño para simular la respuesta generada a partir de estas instrucciones.

En la Figura 9 del Apéndice B, se pueden observar la respuesta de 9 señales a partir de la entrada de las instrucciones. Al compararlas con nuestra tabla de verdad, se observa que el funcionamiento es correcto.

3.2. ALU Control

De manera similar se diseño el test bench para simular el comportamiento del bloque ALU Control. Concentrándose en las instrucciones de Tipo R, las cuales se accionan contemplando la instrucción. En la Figura 10 del Apéndice B, se puede ver como la señales generadas son las esperadas.

4. Conclusión

4.1. Paul Vazquez

Esta practica represento el primer paso para la implementación completa de nuestro MIPS. Fue un gran reto, ya que tuvimos que consultar muchas decisiones pasadas sobre detalles de la implementación de los módulos anteriores para la realización de la tabla con las lineas de control que le corresponde a cada instrucción. Una vez que se diseño la tabla, la implementación fue sencilla pero esperamos que tenga el resultado deseado cuando se conecte con los demás componentes que nos faltan.

4.2. Rodolfo Cuan

Lo que nos tomo más tiempo al realizar esta practica fue la creación de la tabla de verdad. Teniendo la tabla, el desarrollo de los módulos no fue gran problema, ya que programarlas no era muy diferente a lo ya realizado en practicas anteriores. Siento que es muy importante ir más allá del solo el desarrollo y también entender bien los conceptos y propósitos de los módulos, esto nos servirá en la siguiente practica cuando tengamos que integrar todo lo aprendido.

References

[1] Diego Fernando Valencia Martínez. *Manual del laboratorio*. Escuela de Ingeniería y Ciencias Departamento de Ciencias Computacionales, 2020.

[2] D. A. Patterson and J.L. Hennessy. *Fomputer Organization Design*. Morgan Kaufmann, 2012.

[3] Ruye Wang. "The Control Unit". In: (2005). URL: http://fourier.eng.hmc.edu/e85_old/lectures/processor/node5.html.

[4] University of Pittsburgh. *"The Control Unit"*. https://www.pitt.edu/ kmram/-CoE0147/lectures/datapath3.pdf, 2019.

5. Apéndices

5.1. A. Esquemático de La Unidad de Control y sus conexiones.

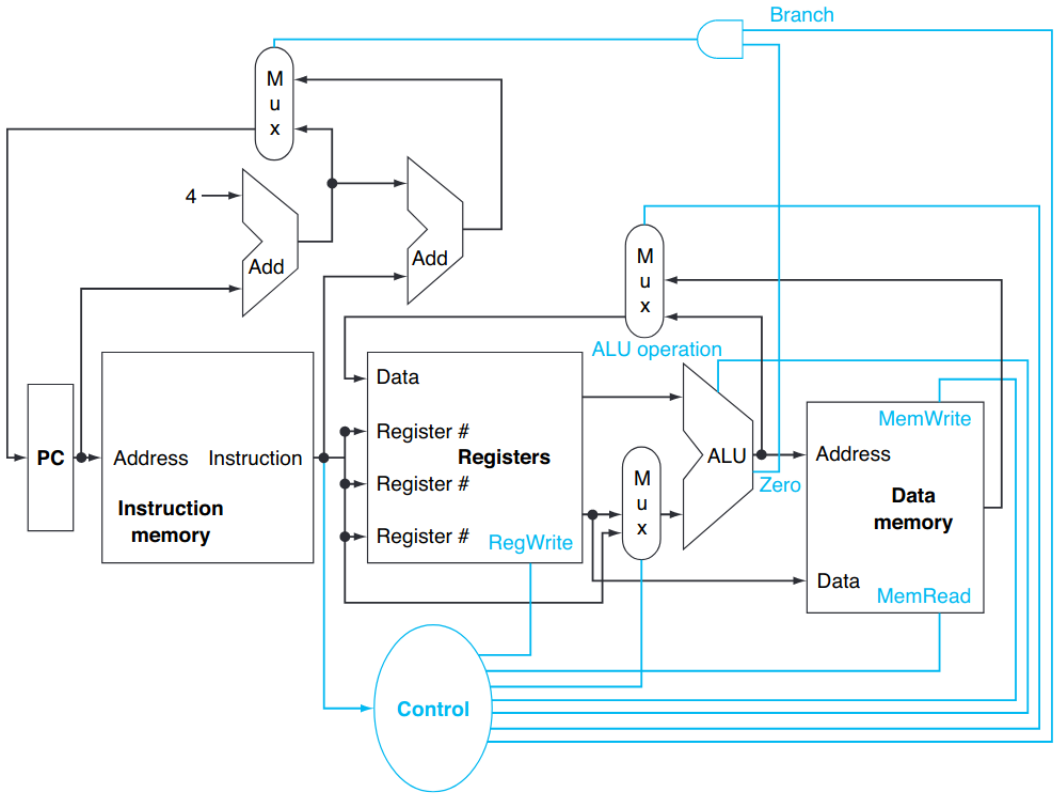

Figure 8: Imagen de [2].

5.2. B. Resultados de TestBenchs

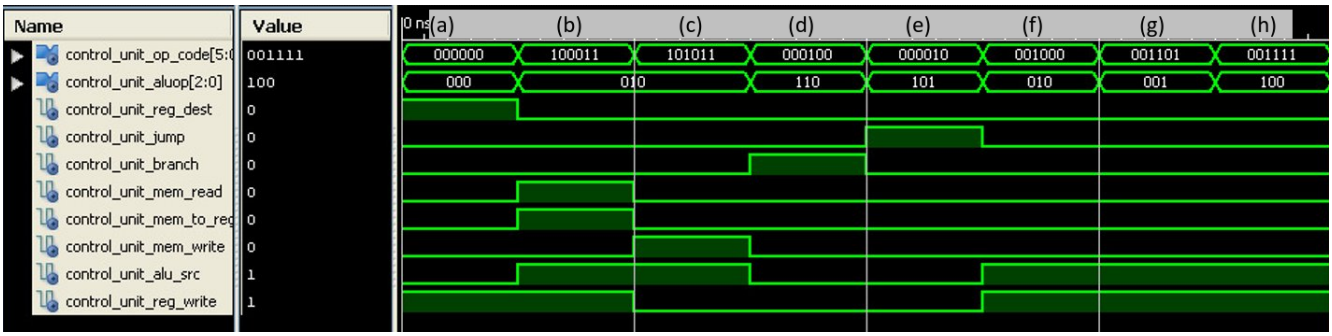

Figure 9: Resultados de TestBenchs de la Unidad de Control.

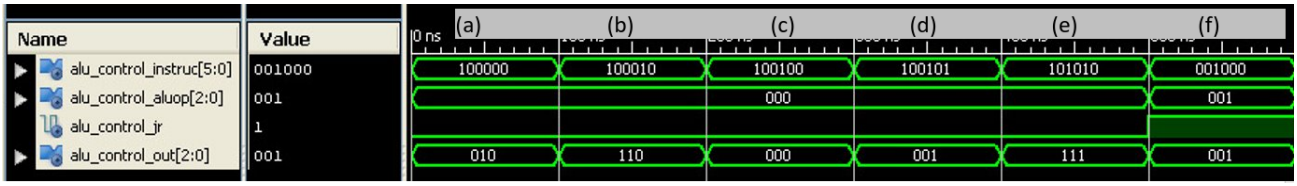

Figure 10: Resultados del TestBench del ALU Control.

5.3. B. Tabla de instrucciones

	Opcode	ALUOP	FunctionCode	REGDEST	JUMP	BRANCH	MEM READ	MEM TO REG	MEM WRITE	ALU SRC	REG WRITE	OPERACION EN ALU
ADD	000000	000	100000	1	0	0	0	0	0	0	1	SUMA
SUB	000000	000	100010	1	0	0	0	0	0	0	1	RESTA
AND	000000	000	100100	1	0	0	0	0	0	0	1	AND
OR	000000	000	100101	1	0	0	0	0	0	0	1	OR
SLT	000000	000	101010	1	0	0	0	0	0	0	1	SLT
JR	000000	000	1000	1 (ojo no importa porque 20-16 vale 0)	0	0	0	0	0	0	1	SUMA
LW	100011	010	xxxxxx	0	0	0	1	1	0	1	1	SUMA
SW	101011	010	xxxxxx	0	0	0	0	0	1	1	0	SUMA
BEQ	000100	110	xxxxxx	0	0	1	0	0	0	0	0	RESTA
J	000010	101	xxxxxx	0	1	0	0	0	0	0	0	NADA
ADDI	001000	010	xxxxxx	0	0	0	0	0	0	1	1	SUMA
ORI	001101	001	xxxxxx	0	0	0	0	0	0	1	1	OR
LUI	001111	100	xxxxxx	0	0	0	0	0	0	1	1	BUPPER

Figure 11: Tabla completa de instrucciones y lineas de control.