Assignment 8 - Building a Controller

50 Points Possible

11/4/2024

Attempt 1

10/31/2024
NEXT UP: Review Feedback

Attempt 1 Score: N/A

Add Comment

Unlimited Attempts Allowed

Details

8 Building a Controller

In this assignment, we will finish our implementation of a fully-functioning (albeit limited) CPU by adding a Controller component.

Objectives

1. Understand the various control signals throughout the architecture
2. Combine previous skills in combinatorial logic to create the controller

Resources

Assignment8.circ (https://usu.instructure.com/courses/759913/files/92989942?wrap=1)
(https://usu.instructure.com/courses/759913/files/92989942/download?download_frd=1)

ControlSimple.txt (https://usu.instructure.com/courses/759913/files/92989945?wrap=1)
(https://usu.instructure.com/courses/759913/files/92989945/download?download_frd=1)

ControlAdvanced.txt (https://usu.instructure.com/courses/759913/files/92989947?wrap=1)
(https://usu.instructure.com/courses/759913/files/92989947/download?download_frd=1)

Task 1 - Create the Controller

Overview: In this task you will be using combinatorial logic to create a controller component. Use the following resources to help complete this controller. You should implement it as a subcircuit named Controller.

Instruction Formats:

Name	Bit Fields					Notes
	bits 15-12	bits 11-9	bits 8-6	bits 5-3	bits 2-0	(16 bits total)
R-Format	op	rs	rt	rd	funct	Arithmetic, logic
I-Format	op	rs	rt	**address/ immediate (6 bits)**		Load/store, branch, immediate
J-Format	op	**target address (12 bits)**				Jump

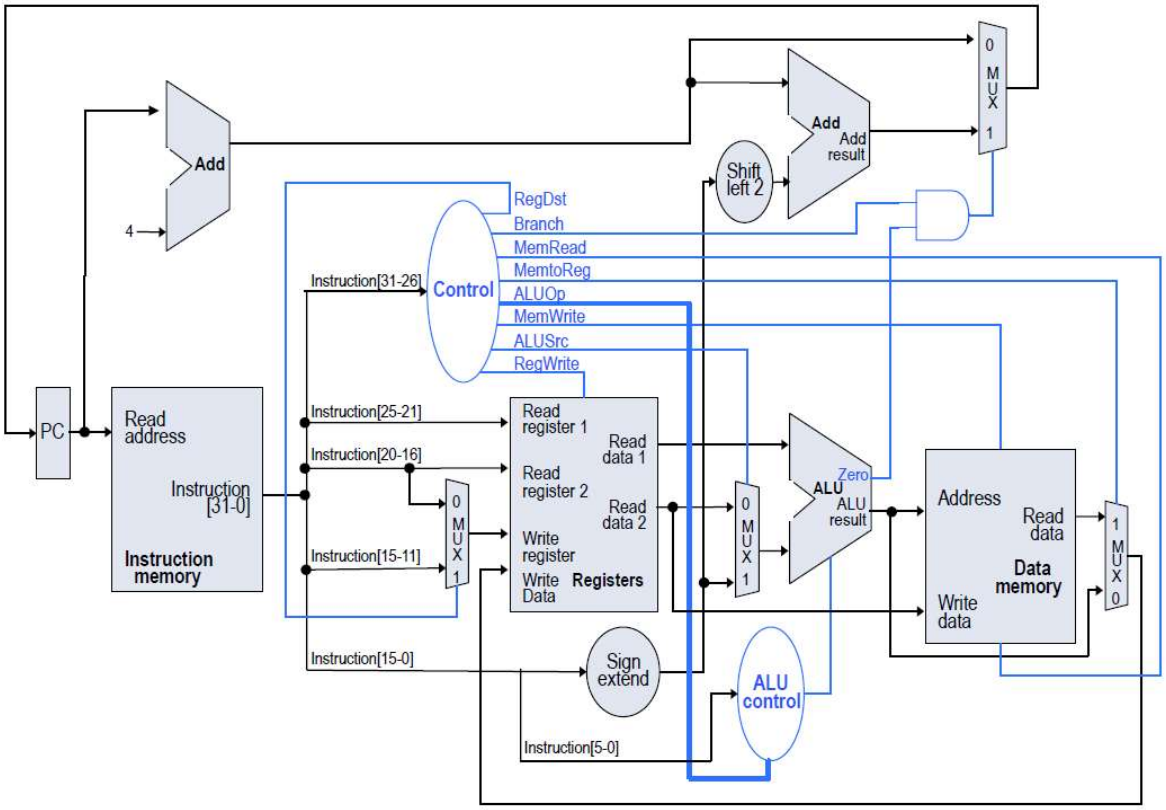
Opcodes:

Core Instruction Set	Syntax	Format	Op	funct
add	add	R	0000	010
subtract	sub	R	0000	110
and	and	R	0000	000
or	or	R	0000	001
set less than	slt	R	0000	111
load word	lw	I	0001	xxx
store word	sw	I	0010	xxx
branch on equal	beq	I	0011	xxx
add immediate	addi	I	0101	xxx

Control Signals:

Signal Name	Effect when deasserted (0)	Effect when asserted(1)
RegWrite	Register does not change	Register is written with data from **Write Data**
ALUSrc	Operand 2 comes from Register File	Operand 2 comes from immediate
ALU Operation	What operation the ALU performs	
MemWrite	We do not write to data memory	We do write to data memory
MemtoReg	ALU Result flows back to register file	Memory value flows back to register file
Branch	0 indicates we do not have a beq instruction	1 indicates we do have a beq instruction
RegDst	Write Register comes from the **rt** field	Write Register comes from the **rd** field

CPU Diagram (for 32-bit MIPS):



Inputs:

- **4-bit Opcode (this should come from the instruction)**
- **3-bit function code (this should come from the funct field of the instruction)**

Outputs:

- **1-bit RegDst**
- **1-bit Branch**
- **1-bit MemRead**
- **1-bit MemtoReg**
- **3-bit ALUOp**
- **1-bit MemWrite**
- **1-bit ALUSrc**
- **1-bit RegWrite**

Tips:

- Start by creating 4-way AND gates that match the Opcode for each instruction
  - These AND gates should output 1 if the Opcode matches the one for each instruction
  - For example, you should have an **AND** gate labeled "lw", that only activates if the Opcode is 0001
  - Really helps to use the **Negate Input** attribute of the AND gates for this
- Once you have each of these gates setup, you can begin connecting their output to the output pins
- For example, **MemWrite** should be 1 if you are writing to memory, i.e. a **sw** instruction. Note that some outputs (**RegWrite** for example), is activated by several instructions. You will need to use an additional gate in this case.
- You can handle all of the outputs except for **ALUOp** by using the above process.
- For **ALUOp**, it should follow the following process:
  - If the instruction is an **R-type** instruction, **ALUOp** should just be whatever **funct** is

- Otherwise, there are two other options for the **ALUOp**
  - The opcode for **add**: 010
  - The opcode for **sub**: 110
  - You will need to figure out which instructions send the code for add vs. subtract to **ALUOp**
- Use multiplexers to enable all of the above behavior

- Take this task one control signal at a time
  - For each control signal, ask yourself the following:
    - What does it mean when the control signal is 1?
    - Which instructions set the control signal to 1?
    - What opcodes do these control signals have?
    - How can I match those opcodes with combinational logic (AND gates and OR gates)?
- You will only use the **funct** field when determining the 3-bit **ALUOp** output.

Testing

The following test vector files can be used to test your implementation of the Controller component

ControlSimple.txt: This file will test all of the outputs **except** for **ALUOp**.

ControlAdvanced.txt: This file will test all of the outputs.

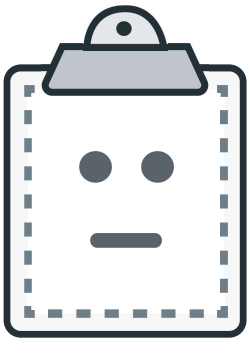
Submission

Submit a file named **Assignment8.circ** containing your Controller implementation.

View Rubric

Assignment 10

Criteria	Ratings		Pts
Task 1 - Controller view longer description	**50 pts** **Full Marks**	**0 pts** **No Marks**	/ 50 pts
			Total Points: 0



Preview Unavailable

Assignment8.circ

Download
(https://usu.instructure.com/files/93047155/download?download_frd=1&verifier=5lS0KmeHZZcgN4bfvDxISJkzb6Rg99CN6v5iCuHy)

(https://usu.instructure.com/courses/759913/modules/items/6509937)
(https://usu.instructure.com/courses/759913/modules/items/65