Project I
Processor Design

CS 2200: Summer 2013

In this project you will design the Instruction Set Architecture (ISA) for a processor, then implement a datapath to support this ISA and a control system to make it all work.

Phase I: ISA Design

For this portion of the project, you are to design an ISA. Its architecture will be based on ideas you were exposed to in CS2110 with its LC-3 processor and CS2200 with its LC-2200 processor.

Specifications

Your processor's memory must conform to the following criteria.

- 32 bit addressability with byte addressing.
- Each discrete memory location should have an address and contain a byte of data.
- The maximum address size is 24 bits (This is a Logisim limitation).
- The ALU must be 32 bits and can implement whatever function you think you will need. If you can adapt and modify anything you made in CS2110 go for it.
- Instructions must be 32 bits long.
- You should design your memory system to allow fetching of 32 bits of data (4 consecutive bytes) at a time.
- Op codes should be six bits.
- Your processor should not have condition codes. You are to implement a BEQ style conditional instruction.

Your processor should at least support the following instructions.
Operate Instructions

- ADD RD RS1 RS2
- NAND RD RS1 RS2
- ADDI RD RS IMM
- LDADR RD OFF

Data Movement Instructions

- LW RD RB OFF (Load Word)

- LB RD RB OFF (Load Byte)
- SW RS RB OFF (Store Word)
- SB RS RB OFF (Store Byte)

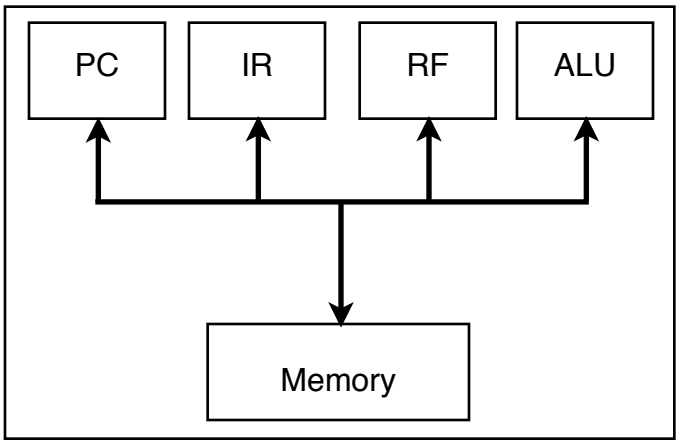
Control Instructions
- BEQ RT1 RT2 OFF
- JALR RT RL

In addition, you are to implement 3 additional instructions of your choosing (e.g. NOT; RSHIFT; LOAD RD, RB, RO).

Phase II: Datapath

You will need both a PC and an IR register to implement the BEQ logic. You will also need an MDR and an MAR to facilitate memory access. You should have a register file (RF) with at least 16 registers (you can have more if you want) The register file should be dual ported. Register 0 should always contain 0, regardless of what is written to it.

We strongly advise you layout your processor cleanly and linearly (as opposed to all wrapped up in a ball like the LC-3).



Control Logic

You may use one or more buses.

You may use all the multiplexors, latches, and registers that you wish. Anything that Logisim provides is fair game.

Instead of having to load the PC with a particular starting address (e.g. x3000), your processor can begin executing at address 0.

BEQ

The BEQ instruction contains two register numbers and an offset. The instruction will subtract the registers from one another, and, if the result is 0, the offset will be added to the incremented PC and that value will be put into the PC. If the difference is not equal to zero then the PC will retain its incremented value.

You may choose to implement other types of branches (BNE, BGT, BGE, etc.)

Phase III: Control System

Here, you will design and implement a Finite State Machine (FSM) to control the flow of your processor using Read-Only Memory (ROM) and a state register. A simplified version of the state machine would look like this:

Current State	<----------Contents---------->	
	Control Bits	Next State
aaa	c1	bbb
bbb	c2	ccc
ccc	c3	ddd
ddd	c4	aaa

Suppose we are in state aaa (i.e. the state register contains aaa). The value in the state register is used as an address to get a value from the ROM. This value consists of two parts: the bits that are used to control the datapath (i.e. the control lines for the RF, MAR, etc.), and the next state.

When run, the state machine should:
1. Use the state number as the address to pull a value from ROM.
2. Use the bits designated for the controls lines to set their values (for the duration of the clock cycle)
3. Use the other bits to latch the next state number into the state register, then repeat.

Naturally, sometimes the next state depends on the opcode or the BEQ comparison result. This can be handled with another ROM. Basically, if your state could transition into more than one next state, use whatever the choice depends on to index into a ROM where you store the next state to go to and load that value into the state register using multiplexors.

This will be presented in detail in class.

Suggested Approach

Decide what you want your ISA to be.

Construct a Finite State Machine that shows how the datapath should operate.

Implement the datapath, but do it in such a way that the datapath can be operated manually. In other words, have inputs that will allow you to toggle the control lines by hand.

Have a connection for the clock that goes to an input or a pushbutton so you can manually pulse the clock.

Do not start on the control system until you can demonstrate that you can manually operate all instructions. This means you operate the datapath per the Finite State Diagram.

Good luck! Get to work!