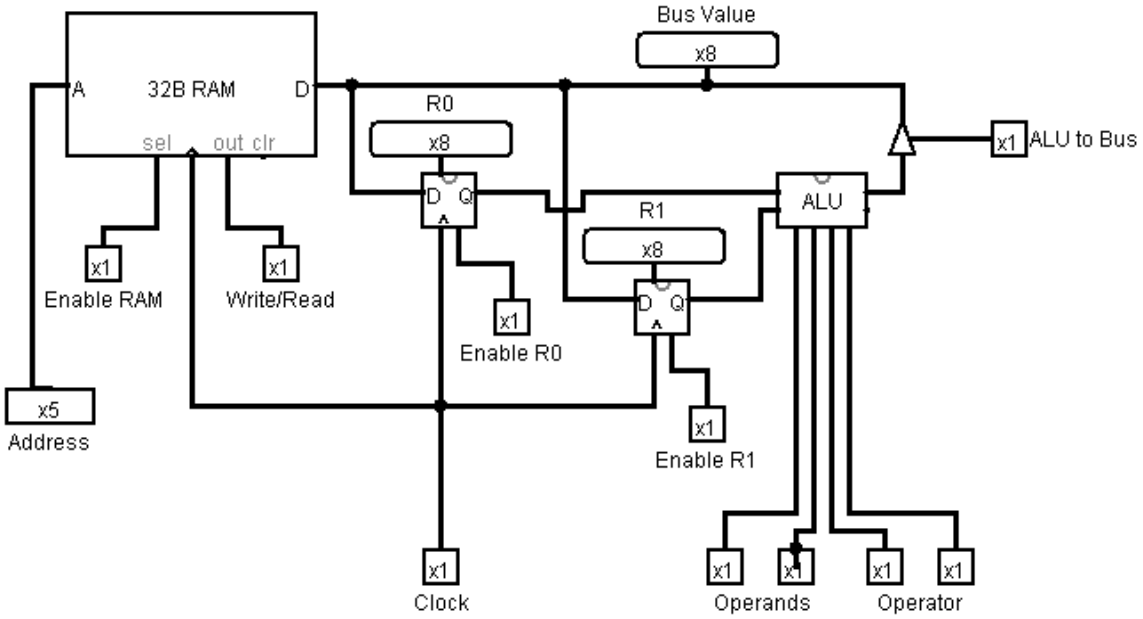
Notes on the Simple Computer Demonstration in LogiSim

While the "Simple Computer" is about as simple as one can get and still perform useful operations, it can also be of bewildering complexity at first glance. These notes will attempt to give a larger sense of how the components of the Simple Computer (abbreviated as SC) fit together, and to give some general directions for problem-solving while using it.

Main Components



The main components of the SC closely match that of a von Neumann architecture computer—there is a region to store data and a separate region to process it.

RAM: RAM, or Random Access Memory, holds binary values using logic gates. The RAM in the SC has 32 addresses, each of which holds one byte, or eight bits. Any memory location can be accessed in any order, which is why it is called random access. Writing to a memory location means to replace its current value with a new one, and reading from a memory location takes the value currently stored and copies it out to the rest of the computer. The memory location is specified using the 5-bit address box below the RAM. There are two switches attached to the RAM, 1) Enable RAM, which specifies whether the RAM is attached to the computer bus, described below, or disconnected, and 2) Write/Read which tells whether the RAM is taking values from the bus and writing to the specified memory location (when the switch is set to 0), or taking values from the memory location and putting them on the bus (switch is set to 1).

Bus: The bus is a set of wires which carries information around the computer. Information can go to and from the RAM, and also to the registers and from the ALU. Notice information does not go directly from the registers to RAM, it must first pass

through the ALU. Also notice that the output of the ALU is not always sent to the bus, passage is controlled by the "ALU to bus" switch, where values pass through if it is set to 1.

Registers and ALU: The registers and ALU should be thought of a connected unit, as they typically reside together on a CPU. Registers are small chunks of memory (a single byte), which are used to temporarily hold data. The ALU performs arithmetic and logical operations on values held in the registers. Note that the RAM is not directly connected to the ALU, so there are no commands to add two locations in RAM together. Instead, data is loaded from the RAM into registers, and then operations can be performed on them. The registers hold values, and the ALU is always seeing those values, even without the clock. The "Enable R0" (or R1) switch specifies whether the value on the bus will be copied into the register when the clock goes. It does not control whether the ALU sees the value—the ALU always does. So the "Enable" switch is more like a "Write" switch for the register.

Clock: The clock is what sets the pace of computations, and is critical for making sure operations occur in a consistent way. For example, values are only written from the bus to RAM when the clock transitions from 0 to 1. Without it, imagine the following scenario:

> *You want to write a value from the bus to the RAM. You set the Enable RAM switch and set the Write/Read to 0, for write. Then you start to set the address bits. But as you change one bit at a time, the value on the bus gets written to the incomplete, yet valid address you have partially specified. Rather than changing one value, you have changed a number of locations.*

Instead, the clock ensures that you have set all the switches to their appropriate values before initiating the command.

Data Flow: The types of problems you are being asked to solve typically involve moving values from RAM to the registers, through the ALU, then back to the registers and eventually back to RAM. This will typically take more than one set of instructions to accomplish. Looking at the diagram, data tends to flow in a counter-clockwise direction: RAM -> registers -> ALU -> RAM/registers. There is one main place of conflict – the bus. Sometimes you may want to read from RAM, which puts values on the bus, or send a value from the ALU to the bus, which obviously also uses the bus. So, one key is that you cannot do both at the same time. Another issue is that you may want to move a value from a register to the RAM. But the registers are not directly connected to send values to the RAM. Instead, they must send a value through the ALU, onto the bus, and from there into RAM. The problem is that the ALU does not have a copy operator, just ADD, SUB, AND, NOR. As discussed in class, any binary number AND'ed with itself produces itself (you should make sure you understand why), so you can do that to copy a register onto the bus.