

## Testing the MIPS Processor

This document, along with the files contained in `progmem.zip` which you should download from the course web page, will allow you to test your MIPS processor once you have finished building it.

### Step 1. Unzip `progmem.zip`

The file `progmem.zip` contains four files as follows:

- `prog1.txt`: Contains MIPS machine code for the first test program that your MIPS processor will run.
- `mem1.txt`: Contains the initial data memory configuration for the first test program.
- `prog2.txt`: Contains MIPS machine code for the second test program.
- `mem2.txt`: Contains the initial data memory configuration for the second test program.

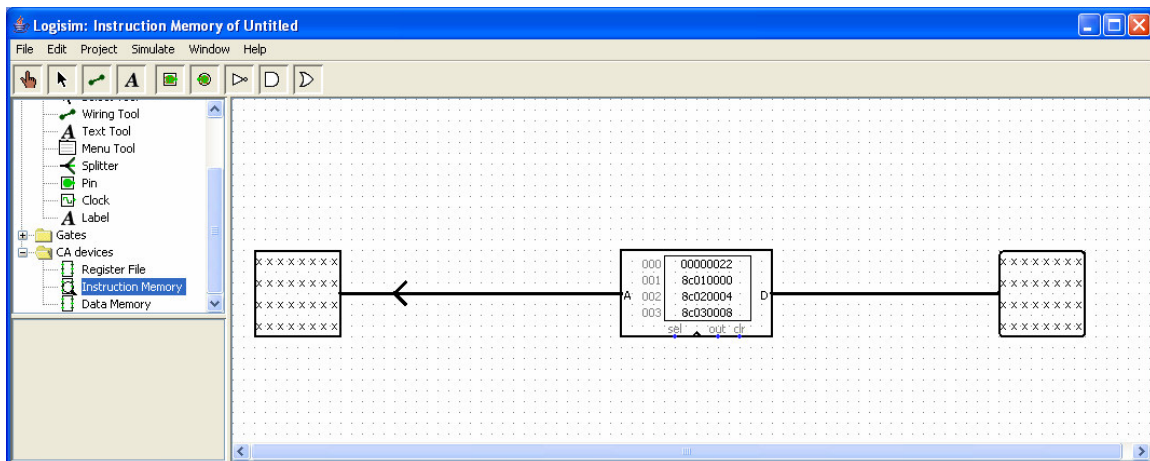
The two test programs are described at the end of this document.

### Step 2. Start Logisim and load your MIPS processor

Your MIPS processor should have a single input, a one-bit clock input. Hook a clock (found in the “Base” folder of devices) up to the clock input. At this point, the clock should not be running (make sure “Simulate>Ticks Enabled” is not checked).

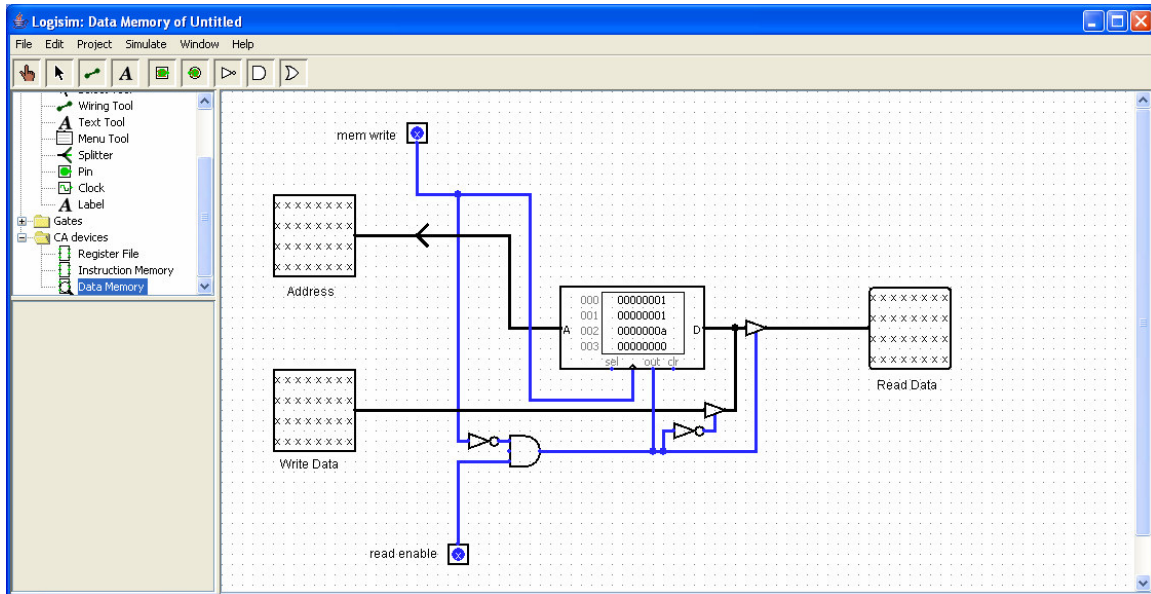
### Step 3. Load the first program into the instruction memory of your processor

Right click on the instruction memory device in your MIPS processor and choose “View Instruction Memory”. In the middle of the device, you will see the actual RAM. Right click on the RAM and choose “Load Image...”. Then, specify the file `prog1.txt` that you saved in a folder after extracting from `progmem.zip`. At this point, you should see something like:



#### **Step 4. Load the initial memory configuration for the first test program**

Go back to your MIPS processor (by right clicking on the name of your MIPS circuit in the upper left-hand window – it might be called “main” or some other name you gave it – and choosing “View Circuit”). Now, right click on the data memory of your processor and choose “View Data Memory”. In the middle of the device, you will see the actual RAM. Right click on the RAM and choose “Load Image...”. Then, specify the file `mem1.txt` that you saved in a folder after extracting from `progmem.zip`. At this point, you should see something like the picture below. Then, go back to your MIPS circuit.



#### **Step 5. Read the description of the first test program at the end of this document**

Make sure you understand the description of the test programs. Note that since there is no halt instruction to end the program, I simply put a loop at the end of each program that does nothing.

#### **Step 6. Run the program on your MIPS processor**

Assuming that your PC register starts off containing 0, the program will start to run as soon as you start the clock ticking. Choose a relatively slow frequency (1Hz or less) by clicking “Simulate>Tick Frequency”, and then start the clock running by clicking “Simulate>Ticks Enabled” (to put a check mark next to it). Once you have let the program run for a little while, you will see that the PC keeps getting the same value over and over again because it has encountered the final loop. At this point, disable the clock by clicking on “Simulate>Ticks Enabled” again.

### **Step 7. Check the result**

The result of the program is written to data memory, as described below. View the RAM in your data memory (see step 4 above) to see if the correct result has been put into the data memory. To examine the RAM, choose the poke tool and click on a location of the RAM. Then, hit enter to scroll down in the RAM or backspace to scroll up in the RAM.

### **Step 8. Repeat for the second test program**

Restart Logisim (or click “Simulate>Reset Simulation”) to clear out the old program and data. Then, repeat steps 2 through 7 above on the second test program, using the files `prog2.txt` and `mem2.txt` (in the same way that you previously used `prog1.txt` and `mem1.txt`, respectively).

(turn to next page)

## A Description of the Test Programs

### Program 1: The Sum Program

This program adds the numbers between A and B, inclusive, where A originally resides at address 4 in data memory and B resides in address 8 in data memory. The result is placed at data memory location 0. Since there is no halt instruction on our MIPS processor, the program, after computing the result, will just loop (until you turn off the clock).

The program is:

### Assembly Code

```
sub r0,r0,r0      ; set reg[0] to 0, use as base
lw  r1,0(r0)     ; reg[1] <- mem[0] (= 1)
lw  r2,4(r0)     ; reg[2] <- mem[4] (= A)
lw  r3,8(r0)     ; reg[3] <- mem[8] (= B)
sub r4,r4,r4     ; reg[4] <- 0, running total
add r4,r2,r4     ; reg[4]+ = A
slt r5,r2,r3     ; reg[5] <- A < B
beq r5,r0,2      ; if reg[5] = FALSE, go forward 2 instructions
add r2,r1,r2     ; A++
beq r0,r0,-5     ; go back 5 instructions
sw  r4,0(r0)     ; mem[0] <- reg[4]
beq r0,r0,-1     ; program is over, keep looping back to here
```

### Machine Code

```
00000022
8c010000
8c020004
8c030008
00842022
00822020
0043282a
10a00002
00411020
1000ffffb
ac040000
1000ffff
```

In the initial data memory configuration, we will put the following values:

- location 0: 1
- location 4: 1 (the variable A)
- location 8: a hex (the variable B, note: a hex = 10 decimal)

Thus, the program will compute the sum of the numbers from 1 to 10, inclusive. At the end of the program (when it is just looping) the result value, 55, should end up in data memory location 0.

### **Initial Data Memory Configuration**

```
00000001
00000001
0000000a
```

---

### **Program 2: The AND/OR Program**

This program tests the processor's AND and OR operations, as well as testing addressing using offsets from a base register. The program starts with two variables, A and B, and does the following:

- computes A AND B and places the result in memory location 4
- computes A OR B and places the result in memory location 8

Initially, data memory is configured as follows:

```
location 0:  14 hex (= 20 decimal)
location 20: the variable A
location 24: the variable B
```

For this example, we'll set the values of A and B as follows:

```
A: 430a1f9b hex
B: 728cd2e3 hex
```

Here is the program:

### **Assembly Code**

```
sub r0,r0,r0      ; set reg[0] to 0
lw  r1,0(r0)     ; reg[1] <- mem[0] (= 20)
lw  r2,0(r1)     ; reg[2] <- mem[20]
lw  r3,4(r1)     ; reg[3] <- mem[24]
and r4,r2,r3     ; reg[4] <- reg[2] AND reg[3]
or  r5,r2,r3     ; reg[5] <- reg[2] OR reg[3]
sw  r4,4(r0)     ; mem[4] <- reg[4]
sw  r5,8(r0)     ; mem[8] <- reg[5]
beq r0,r0,-1     ; program is over, loop back here
```

### **Machine Code**

00000022  
8c010000  
8c220000  
8c230004  
00432024  
00432825  
ac040004  
ac050008  
1000ffff

### **Initial Data Memory**

00000014  
00000000  
00000000  
00000000  
00000000  
430a1f9b  
728cd2e3