1. Booting Computer

To begin testing the functionality of your CPU, follow these steps:

1. Load [illegible]struction and data memory into the ROM and RAM [illegible]ely.
2. Set t[illegible]ial low state by poking it.
3. Trig[illegible] on and off.
4. If yo[illegible]input to the keyboard, do so at this point.
5. Use either the manual "Tick Once" function (Ctrl+T) or the automatic "Ticks Enabled" function (Ctrl+K) to start the program execution.

2. Automated Testing

For the self-test tool to work, your circuit must meet the following requirements:

1. Name your circuit file "cpu.circ" and store it in the same directory as the tool and data.
2. Use a component called "RegisterFile" (capitalization matters) as your register file.
3. Use an input named "reset" (capitalization matters) as your reset signal.
4. Place a Probe on each register in your register file and name them "r0", "r1", "r2", etc.
5. Set the default state of all D flip-flops to 0 to avoid issues with tests that toggle the reset signal.
6. You can use Probes for your own purposes, but leave their label blank. The tester ignores unlabeled probes, but labeled probes other than "r0", "r1", etc. will affect the results.
7. Configure the TTY with 13 rows, 80 columns, and rising edge.
8. Use a ROM component only for instruction memory, as the tool will overwrite every ROM component in your circuit with instruction data.
9. Make sure you have Python and Java 1.6/1.7/1.8 installed to run the tester. The tool has been tested on Course Linux, but it should work on other environments as long as the required Java Runtime Environment is available.
10. Do not use the "Project | Add Library" feature, as it will break your submission online。Instead, use the "File | Merge" feature to bring in circuits from other files. You may need to rename circuits to avoid conflicts.

To view a usage message, run the command " ./hwtest.py". The command will generate files named "actual.txt" and "diff.txt" to show your output and the differences between your output and the expected output. The assembly language source code for the CPU 250/16 tests is located in the "programs/" directory. Refer to the assembler and simulator section in this document for more information.

If you want to manually run a specific command line test, you can use the Logisim command-line version directly. Check the "settings.json" file for the test configuration, especially the arguments. For example, the "simple" test has the following arguments:

```
"args": [
"-c", "10",
"-ic", "0,reset=1:1,reset=0",
"-lo", "tests/simple.imem.lgsim",
"-la", "tests/simple.dmem.lgsim"
]
```

To execute this test ma ents on the command line as follows (note that this is all one command):

```
java -jar logisim 10 -ic 1,reset=1:2,reset=0 -lo tests/simple.imem.lgsim -la tests/simple.dm
```

3. Assembler & Simulator

We are providing an assembler and a simulator for you to generate test programs and to verify your program's behavior. The assembler and simulator are included in the folder

There are two pseudo-instructions available for use in your programs:

1. `la $rd, label   # load address`
2. `halt`

The `la` pseudo-instruction is converted into multiple actual machine instructions that have the effect of loading a 16-bit address into the specified register (specifically, a series of `addi` and `sll` instructions). Specifically, the transformation is that:


```
la $rd, ADDR
```

Will become the following, where the bracket notation indicates bits within ADDR:


```
addi $rd, $r0, ADDR[15..11]
sll  $rd, $rd, 5
addi $rd, $rd, ADDR[10..6]
sll  $rd, $rd, 5
addi $rd, $rd, ADDR[5..1]
sll  $rd, $rd, 1
addi $rd, $rd, ADDR[0]
```

The `halt` instruction is actually a branch that simply branches back to itself, creating an infinite loop (though when run with the simulator, this special branch is detected and causes the simulator to terminate). The `halt` instruction is actually assembled into:

```
beq $r0, $r0, -1
```

For information on using these tools, see the readme.txt included with it!

4. *Final Tests*

After passing all the tests, you can run the demo programs that come included:

- demo-fib-pri…uses an iterative approach instead of recursive and prints th…s to the TTY. The recurse.s test performed by the tester only re…egister.
- demo-prime-…m computes prime numbers, which is a challenging …out a divide instruction!

To run these programs, you need to use the included assembler tool to produce imem and dmem files as they do not come pre-assembled. Running these programs at a high clock speed will cause your CPU to pulse and generate numbers after numbers to the console.