Getting Ready for the TEPID Implementation Project.

As I have explained in class, for this semester's final project you will design a digital circuit to implement a minimal, hypothetical instruction level architecture called TEPID. Your circuit will be constructed and tested using Logisim.

This week, to prepare for this project we want you to complete two tasks:

1. Extend a Logisim circuit we will provide that implements a small subset of the MIPS architecture so that it can execute the jump, jump and link, and jump register instructions. The purpose of this task is to familiarize you with how to work with larger Logisim projects and how to use them in conjunction with an assembler to create and load machine language programs that test your circuit.

2. Write a small program in the assembly language for the TEPID architecture and assemble and test it using the WARM assembler and interpreter. The goal of this task is to introduce you to the details of TEPID and the software tools Duane Bailey created for WARM.

Part I: Implementing Jump Instructions

The MIPS architecture includes four jump instructions: jump, jump and link, jump register, and jump and link register. The book contains a section that outlines how to implement jump. You should definitely look at that section before starting this assignment. What we would really like you to add is jump and link. Implementing jump and link, however, requires doing most of the work to implement jump. In particular, both instructions are encoded as J-Type instructions, a format not yet supported by the circuit we have discussed in class. If you do jump and link, you might as well do jump too! Worse yet, jump and link leaves the return address in a register, $ra. Without the jump register instruction, you cannot use this return point information. Therefore, to make this assignment meaningful, you should implement three of the four jump instructions (you don't need to implement the fourth, jump and link register, but it should be pretty easy if you want to try it).

To get you started, a copy of a slightly improved version of the Logisim project we have seen in class will be made available through the labs page of the course website. This project contains several improvements:

- We moved the components of the data flow diagram around a bit to try to make room where we think you will need it. Consider this a hint. If you want to put a component somewhere that there is no room for it, maybe you should try to think of another way.

- We added lots of comments explaining the circuits. We are trying to set a good example! The projects you submit for this lab should include comments explaining what you did. You should also take care to keep your circuits as simple as possible and to take the time to lay everything out neatly so that your circuit is easy to understand. Failure to do so will reduce your grade.

- We added three outputs labeled "is jump", "is jal", and "is jump register" to the CONTROL sub circuit. Right now they are all wired to always output 0.

  Consider these outputs a hint. You could use other control signals if you think they are necessary, but we believe that these three control signals should be sufficient to complete the desired circuit. You will, of course, have to add circuitry to set them based on the op code and funct fields of the current instruction.

- We modified the instruction memory so that it only pays attention to the last 20 bits of the program counter. This addresses a problem with the way MARS positions code in memory. The MARS text segment begins at address 0x00400000. It is very hard to load code from a file into a Logisim random access memory at this address. It is much easier to load the code at address 0x00000000. This is what we have been doing in class and it works fine for programs that do not contain jumps. The addresses in jumps, however, are encoded in a way that depends on the actual location of the code. By only looking at the last 24 bits of instruction addresses, 0x00400000 becomes equivalent to 0x00000000.
- We added the PC + 4 value as an input to the instruction decoder and modified the decoder so that it uses the low order bits in a jump or jump and link instruction correctly to determine the target address.

Despite these efforts on our part. There is still work for you to do.

- You will need to add several multiplexers to the circuit and connect them to the appropriate data and control inputs. For example, the new value for the PC can now come from four (!) sources:
  1. The PC + 4 line,
  2. the output of the adder that combines the updated PC value with the sign-extended offset from a branch instruction,
  3. the ADDR output of the instruction decoder for a jump or jump and link, or
  4. the RD1 output of the register bank for a jump register instruction.
- You will need to make the outputs we added to the CONTROL circuit work and connect them to your new multiplexers or other components. Some of the existing outputs of the control circuit will also need to be modified to account for the jump instructions.

Plan your approach carefully before you start editing the circuit!

Using MARS with Logisim

To test your circuit, you will want to write small MIPS assembly language programs, debug them with MARS, save the machine code MARS produces to a file, and then load that file into the instruction memory of your Logisim circuit.

You should keep the test programs you create simple. In particular, do not put anything in the data segment. Just write code for the text segment.

After your code is correct, select "Dump Memory" from the MARS file menu. In the dialog box that appears:

- Select the .text segment for output.
- Select hexadecimal text as the output format.
- Click "Dump to File"

Navigate to put the file in the folder with your Logisim project. I tend to name these binary files with a .o suffix even though this is not completely appropriate.

Next edit the file with a text editor (emacs or TextEdit will do). Add a first line of the form "v2.0 raw". Save the update file.

Now, switch to Logisim.

- Double-click to make the "MIPS Single Cycle Dataflow" sub circuit the active circuit.
- Click on the pointing finger icon in the upper left corner of the Logisim window.
- Click on the "instruction memory" component of the circuit. It should become highlighted and a magnifying glass icon should appear in the middle of its icon.
- Double click on the magnifying glass to see the details of the instruction memory.
- Control click on the ROM icon within the instruction memory sub-circuit.
- Select "Edit contents" from the menu that appears.
- Click on the "Open" button at the bottom of the window showing the memory values.
- Navigate to and open the MARS memory dump file you created and edited.
- Make sure your code is now displayed at the beginning of memory.
- Close the memory contents window.

If you need any initial values in the data memory to test your program, you can enter them directly using the mouse from the "Edit Contents" window for the data memory.

There are lots of tricks you can use when testing your circuit. Once it is working fairly well, you can run at a reasonable speed by selecting the clock speed using the "Tick frequency" item in the Logisim "Simulate" menu and then enabling ticks in the same menu. If you need to watch the circuit step by step, selecting the "Tick once" item may make more sense.

Part II: Getting Warm

To introduce you to Tepid and the tools for Warm that you can use to write and test sample Tepid programs, we would like you to translate the following simple C program into Warm assembly language using only the instructions included in Tepid.

```
#include <stdio.h>

int main( int argc, char *argv ) {
  int count;
  int values[20];

  scanf( "%d", &count );

  int p;
  for ( p = 0; p < count; p++ ) {
    scanf( "%d", &values[ p ]);
  }

  for ( p = count - 1; p >= 0; p-- ) {
    printf( "%d\n", values[ p ]);
  }

  return 0;
}
```

Your completed TEPID program should be stored in a file named `reverse.s`.

You should consult "The WARM Assembler and Interpreter Guide" handout together with the handout describing TEPID to complete this part of the lab. In addition, to make the tools described in the assembler handout work, you should type the command

```
source /usr/cs-local/bin/237
```

in your terminal window.

Submitting Your Work

As we did for labs 3 and 5, we want you to submit both the .circ file for your work on this lab and a file containing drawings of the sub-circuits you modified so that we can print these drawings easily. To produce this drawing file:

- Select the "Print" item from the Logisim File menu. A dialog box should appear.
- Shift click or drag to select the names of the CONTROL and data flow sub-circuits in the list in this window (i.e. select all of the sub-circuits you have modified).
- Edit the "Header" field in the window by adding your name. Click OK. A new dialog box should appear. Make sure the Print to File check box is selected. Then click Print.
- Use the next dialog box to navigate into the folder where your .circ file is saved and save the file containing drawings of your circuits as out.ps in this folder.

For this lab, we would also like you to submit at least one of the MIPS assembly code files for the test programs you used with your Logisim circuit. To make it easy to find this file and your .circ file:

- Rename the test file MIPStest.s.
- Rename the .circ file MIPSplusJUMPS.circ

If necessary, move a copy of your WARM assembly code file, reverse.s, to the same folder as your Logisim files.

Within a terminal window, use the cd command to make the directory containing all your files your current working directory. Then type the command:

```
tar -cf lab8.tar out.ps MIPSplusJUMPS.circ MIPStest.s reverse.s
```

Next type the command:

```
turnin -c 237 lab8.tar
```

Respond to the prompts appropriately and your work should be submitted.