**Lab 6 – MIPS Datapath for R-Type, I-Type, and J-Type Instructions**

CECS 341 – Computer Architecture & Organization

Kenry Yu, 028210726

Olena Bilinska, 028897191

Garret Towner, 028303091

Professor: Mandy He



California State University, Long Beach

College of Engineering

1250 Bellflower Blvd, Long Beach, CA 90840

May 4, 2022

**Goal/Objective:**

The objective of this lab is to modify the existing MIPS datapath and add the capabilities to execute the jump instruction.

**Technical Description/Steps:**

The source codes of MIPS datapath from the last lab are used in this lab. The MIPS datapath would need an additional wire and multiplexer to support J-type instructions. On top of that, the control unit needs a new case to support the opcode of the jump instruction. We would use Xilinx Vivado to design and edit the existing MIPS datapath.

During the project, we:

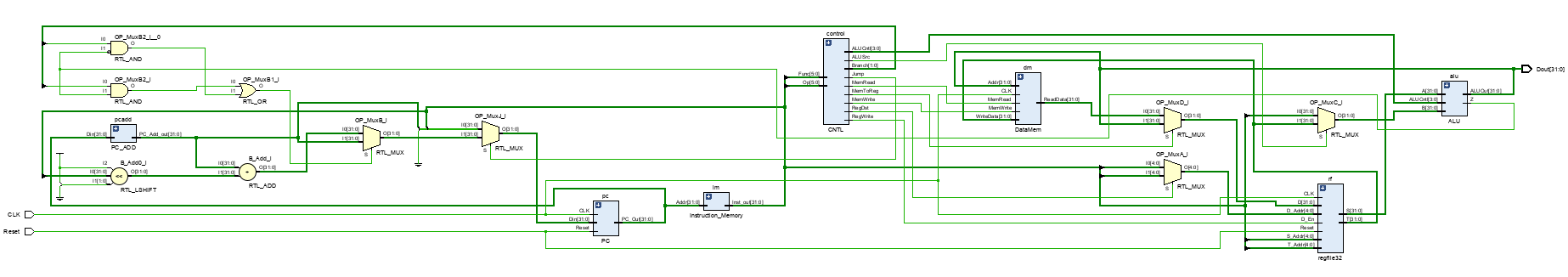
1. Added a 1-bit wire named “Jump” in both the control unit and the datapath to indicate J-type instruction.
2. Added a new case in the control unit to output the correct control signals.
3. Added a new 32-bit wire named “Jump\_Add” to determine the target address to jump for the jump instruction.
4. Added a new multiplexer named “OP\_MuxJ” to determine the target address between the result from “Jump\_Add” and the result from the multiplexer for the branch instruction.

After we finished designing the MIPS datapath and confirmed the design by comparing the schematic, we moved on to translating the assembly language instruction to hexadecimal codes.

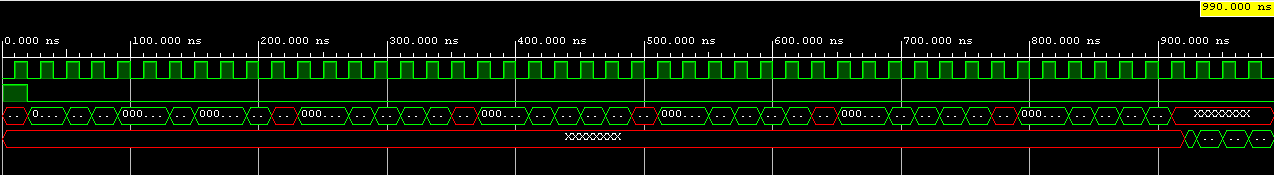
To translate the instruction to hexadecimal codes:

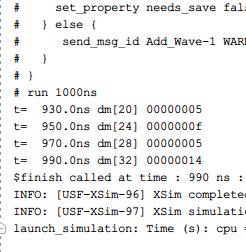
1. Determine the opcode, funct number, and register numbers in hexadecimal form by checking the MIPS reference card.
2. Translate the hexadecimal code to binary form.
3. Combine the binary codes, then split the entire instruction into groups of four.
4. Translate each group to hexadecimal form.
5. Convert the code to little-endian order by reversing the order of the instruction.
6. Store converted instructions into an “imem.dat” file.

**Results:**



The image above is the schematic of the datapath module, it shows that our modules are connected together with the new multiplexers we added. In addition, the “DataMem.v” file provided by the instructor is also connected in the schematic.  
The image below is the waveform of the simulation of the MIPS datapath.





The image above is the console output of the program, it displays the result of the instructions that were stored in the specified registers. Our handwritten calculation matches the operation performed.This calculations attached to group zip file named “Result\_Calculations”.

**Conclusion:**

In this lab, we learned how to modify the project from the last lab to support J-type instructions in the datapath. We also learned how to use the syntax for multiplexers to determine the input data that we want. Over the project, we encountered problem to determine end value for our “beq” operation, which have not provided us the satisfy result and continue running after reaching the end value. After examining the behavior of “beq” instruction, we changed our codes in ALU and it began to output correct results.