CENG 351: Computer Architecture I

Lab 4: Datapath Implementation

MIPS Datapath Design and Instruction Execution Analysis

Team Information

Lab Partners: Gabriel Giancarlo, Jun Yi

Contributions:

- **Gabriel Giancarlo:** Report writing, instruction analysis, datapath explanation, and comprehensive documentation
- **Jun Yi:** Circuit implementation, screenshot capture, simulation execution, and verification

1. Introduction

In this lab, we built and tested a MIPS datapath to see how a processor actually runs instructions. We put together all the main parts of a CPU datapath like instruction memory, register file, ALU, data memory, and control logic.

Datapath Overview

Our MIPS datapath has these main parts:

- **Instruction Memory:** Holds all the program instructions in hex format
- **Program Counter (PC):** Keeps track of which instruction we're running
- **Register File:** Has 32 registers that can each hold 32 bits
- **Arithmetic Logic Unit (ALU):** Does all the math and logic operations
- **Data Memory:** Stores data that the program needs to load and save
- **Control Logic:** Figures out what each instruction does and tells other parts what to do

2. Instruction Analysis

2.1 Hex Program Analysis

We got a hex program that has these MIPS instructions:

Instruction Breakdown

Address 0x0000: `20100024` - addi $16, $0, 36 (Load immediate value 36 into register $16)

Address 0x0004: `ac100004` - sw $16, 4($0) (Store word from $16 to memory address 4)

Address 0x0008: `8c110004` - lw $17, 4($0) (Load word from memory address 4 into $17)

Address 0x000C: `2231ffff` - addi $17, $17, -1 (Decrement $17 by 1)

Address 0x0010: `02114022` - sub $8, $16, $17 (Subtract $17 from $16, store in $8)

Address 0x0014: `01114824` - and $9, $8, $17 (Bitwise AND of $8 and $17, store in $9)

Address 0x0018: `1109fff9` - beq $8, $9, -7 (Branch if $8 equals $9, jump back 7 instructions)

Address 0x001C: `00000000` - nop (No operation)

2.2 Program Flow Analysis

This program basically makes a simple loop that:

1. Puts the number 36 into register $16
2. Saves that number in memory location 4
3. Loads the number back into register $17
4. Takes 1 away from $17

5. Finds the difference between $16 and $17
6. Does a bitwise AND operation
7. Jumps back if the values are the same, making it loop

3. Datapath Implementation

3.1 Circuit Components

Program Counter (PC)

This is a 32-bit register that remembers which instruction we're working on. It goes up by 4 for each instruction (unless we're jumping or branching).

Instruction Memory

This is where all the program instructions are stored. The PC tells it which instruction we want, and it gives us back the 32-bit instruction.

Register File

This has 32 registers that can each hold 32 bits. We can read from two registers at once and write to one. Register $0 is always zero.

Arithmetic Logic Unit (ALU)

This does all the math and logic stuff. It takes two 32-bit numbers and can add, subtract, do AND/OR operations, and compare them.

Data Memory

This is where we store data that the program needs. We can read from it and write to it using load and store instructions.

3.2 Control Signals

The datapath uses different control signals to make sure everything works together:

- **RegWrite:** Lets us write to the register file
- **MemRead/MemWrite:** Controls when we can read or write to data memory
- **ALUOp:** Tells the ALU what operation to do
- **RegDst:** Picks which register to write to
- **ALUSrc:** Chooses between register values and immediate values
- **Branch:** Lets branch instructions work

4. Screenshots and Simulation Results

4.1 Initial Circuit State

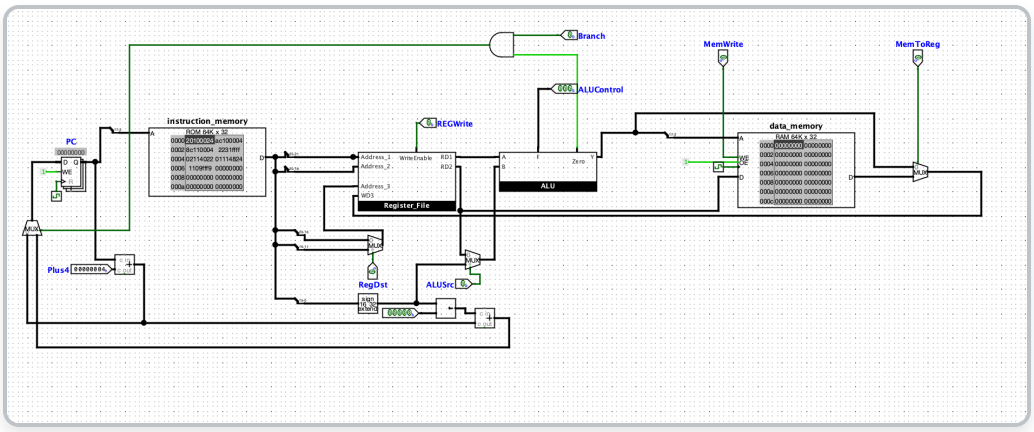

Figure 1: Initial datapath circuit state

4.2 ADDI Instruction Execution

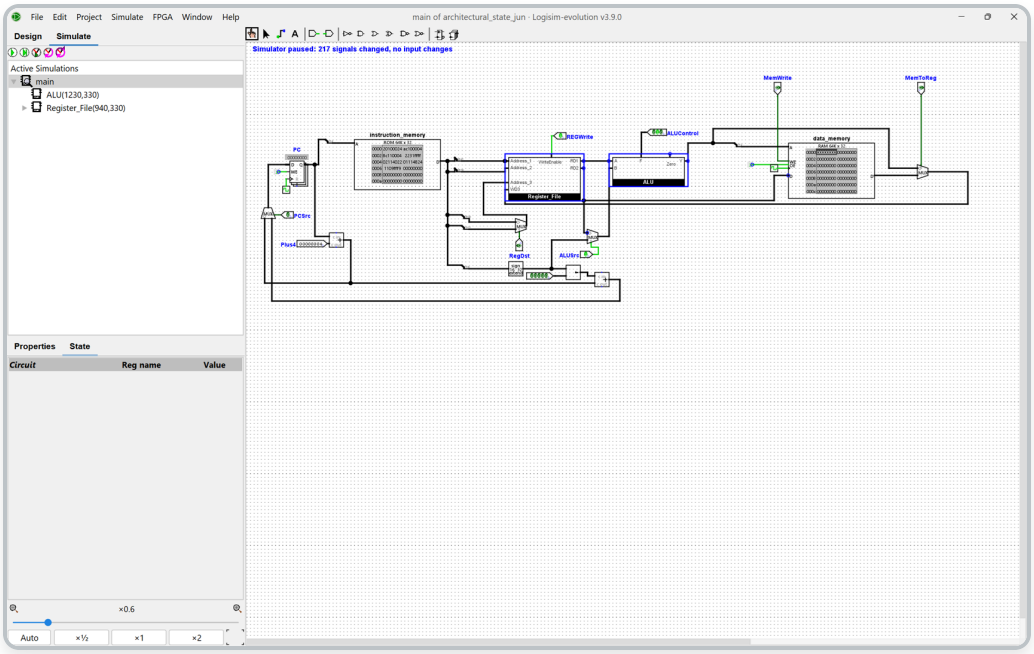

Figure 2: ADDI instruction execution (addi $16, $0, 36)

4.3 Load Word Control Signals

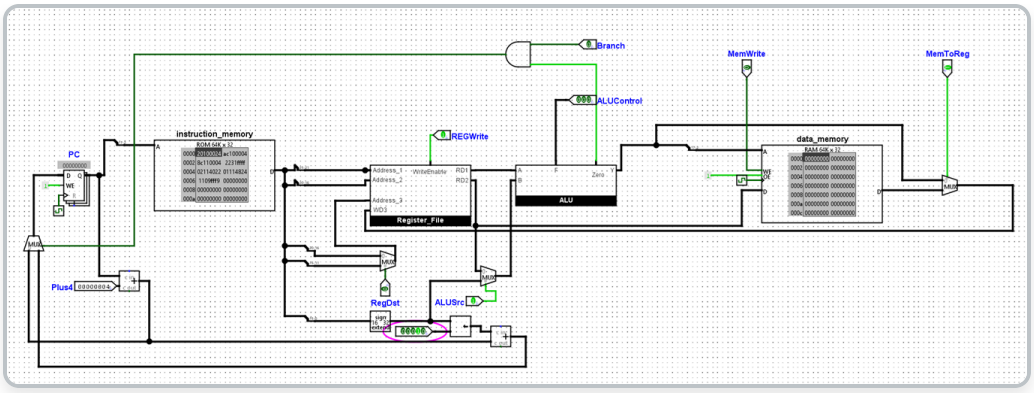

Figure 3: Load word control signals (lw $17, 4($0))

4.4 SUB Instruction Execution

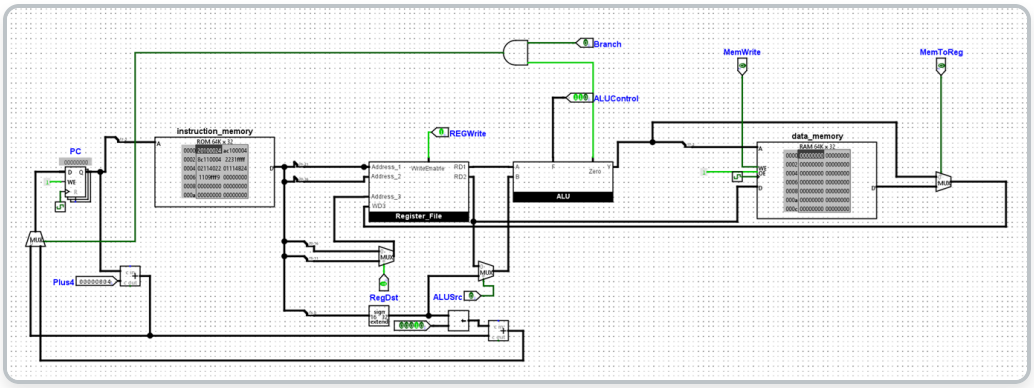

Figure 4: SUB instruction execution (sub $8, $16, $17)

4.5 AND Instruction Execution

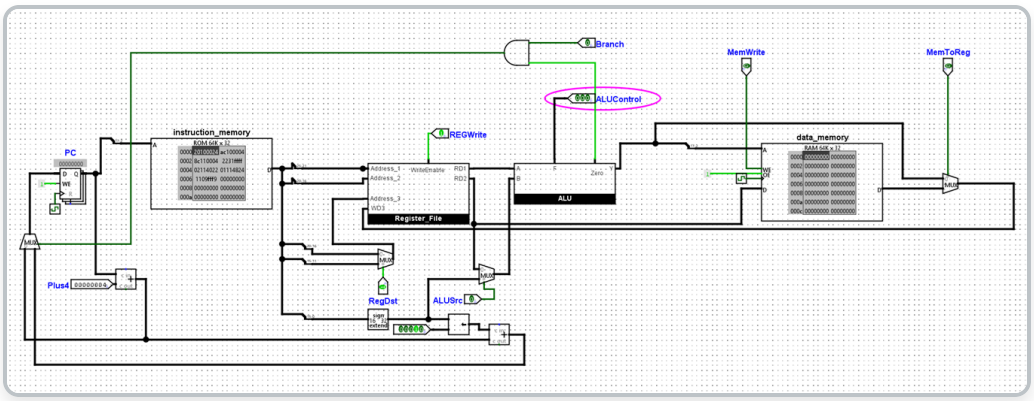

Figure 5: AND instruction execution (and $9, $8, $17)

4.6 Wire States and Data Flow

[SCREENSHOT PLACEHOLDER]

Datapath showing data flow through wires and connections

To be added by partner: *Screenshot showing wire states and data flow*

Figure 6: Wire states and data flow visualization

5. Lab Questions and Analysis

Question 1: Explain the purpose of each component in the MIPS datapath and how they work together to execute instructions.

All the MIPS datapath parts work together to run instructions:

- **Program Counter (PC):** Keeps track of which instruction we're on. It goes up by 4 for each instruction or jumps to a new address for branches and jumps.
- **Instruction Memory:** Holds all the program instructions. The PC tells it which instruction we want, and it gives us back the 32-bit instruction.
- **Register File:** Gives us quick access to 32 registers. We can read from two registers and write to one at the same time.
- **ALU:** Does the actual math and logic that the instruction needs. It takes two 32-bit numbers and gives us a result plus some status flags.
- **Data Memory:** Stores data that we can load into registers or save from registers. It's separate from instruction memory so we can access both at the same time.
- **Control Unit:** Figures out what the instruction does and tells all the other parts what to do.

Everything works like a pipeline: the PC gets the instruction address, instruction memory gives us the instruction, the control unit figures out what to do, the register file gives us the numbers we need, the ALU does the work, and data memory handles loading and storing.

Question 2: Analyze the provided hex program and explain what it does step by step.

The hex program makes a simple loop that shows different types of MIPS instructions:

1. **addi $16, $0, 36:** Puts the number 36 into register $16
2. **sw $16, 4($0):** Saves the value in register $16 (36) to memory address 4
3. **lw $17, 4($0):** Loads the value from memory address 4 (36) into register $17
4. **addi $17, $17, -1:** Takes 1 away from register $17 (now it has 35)
5. **sub $8, $16, $17:** Finds the difference between $16 (36) and $17 (35), puts the result (1) in $8
6. **and $9, $8, $17:** Does a bitwise AND between $8 (1) and $17 (35), puts the result in $9
7. **beq $8, $9, -7:** Jumps back 7 instructions if $8 equals $9, making it loop

Basically, this program makes a loop that counts down and does some math, showing how a processor does its basic operations.

Question 3: What are the key differences between instruction memory and data memory in the MIPS architecture?

Instruction memory and data memory do different things in MIPS:

- **Purpose:** Instruction memory holds the program code, while data memory holds the data that the program works with.
- **Access Pattern:** Instruction memory is usually read-only when the program runs, while data memory lets us both read and write.
- **Addressing:** Instruction memory is accessed in order using the Program Counter, while data memory is accessed using addresses from load/store instructions.
- **Width:** Both are 32-bit, but instruction memory gives us complete instruction words, while data memory can be accessed in different sizes.

- **Timing:** Instruction memory access happens during the fetch stage, while data memory access happens during the memory stage.
- **Control:** Instruction memory access is automatic (the PC does it), while data memory access is controlled by load/store instructions.

Having them separate lets us access instructions and data at the same time, which makes the processor faster and more efficient.

Question 4: How does the ALU determine which operation to perform, and what are the different types of operations it can execute?

The ALU figures out what to do based on the ALUOp control signal that the control unit makes:

- **ALUOp Encoding:** The control unit looks at the instruction opcode and function field to figure out what ALU operation to do.
- **Operation Types:**
  - **Arithmetic:** Addition (add, addi), subtraction (sub), multiplication (mult)
  - **Logical:** AND (and, andi), OR (or, ori), XOR (xor, xori), NOR (nor)
  - **Comparison:** Set Less Than (slt, slti), checking if things are equal for branches
  - **Shift Operations:** Left shift (sll), right shift (srl, sra)
- **Control Logic:** The ALU control unit uses the ALUOp signal and the function field of R-type instructions to make the specific ALU control signals.
- **Output Flags:** The ALU gives us a result and status flags (zero, overflow, carry) that are used for branches and error handling.

The ALU is basically the brain of the processor that does all the math and logic stuff.

6. Troubleshooting and Challenges

6.1 Collaboration and Merge Conflicts

One of the biggest challenges we faced during this lab was working together on the same circuit file. Since we were both working on different parts of the skeleton, whenever one of us moved components around or changed something, we'd get merge conflicts when trying to combine our work.

This was really frustrating because we had to put in extra effort to figure out how to merge our changes without breaking the circuit. We ended up having to:

- Text each other before making any changes to avoid conflicts
- Figure out how to combine our work without losing either person's progress
- Test everything after merging to make sure it still worked
- Decide who was working on which parts to avoid stepping on each other's toes

It was definitely a learning experience about how important it is to communicate well when working on team projects, especially when you're both editing the same files.

7. Conclusion

This lab helped us understand how to build and analyze a MIPS datapath. Here's what we learned:

- ✓ How all the MIPS datapath parts work together
- ✓ How instructions flow through the datapath step by step
- ✓ How to build register file, ALU, and memory components
- ✓ How control signals work and coordinate everything
- ✓ How to analyze hex programs and different instruction types
- ✓ How high-level programming connects to what the processor actually does

Our datapath has all the main parts needed to run MIPS instructions, showing how processors get, decode, and run instructions. Looking at the hex program helped us see how different instruction types use different parts of the datapath.

This lab taught us a lot about computer architecture basics and how complicated it is to design a good processor datapath.

8. References

- Logisim-evolution Documentation: https://github.com/logisim-evolution/logisim-evolution
- Computer Organization and Design, Patterson & Hennessy
- Digital Design and Computer Architecture, Harris & Harris
- CENG 351 Course Materials: Computer Architecture I
- MIPS Architecture Reference Manual