Rawisara Chairat

U65053623

Lab Report

Introduction

The goal of this lab was to expand upon a basic single-cycle MIPS CPU design by adding and testing new instructions. Each modification to the CPU involves adjusting the control logic and datapath, followed by detailed testing to ensure functionality and correctness. Tasks include implementing instructions like SLT, ADDI, J, BNE, and LUI, as well as testing for functional accuracy.

Instructions Implementation

1. SLT Instruction (Set on Less Than)

- The SLT instruction was implemented to set the destination register to 1 if the first source register's value is less than the second source register's value. Otherwise, it would set the destination register to 0. This operation was added by modifying the ALU and ALU Control logic as follows:

1.1. ALU Control Modification: The ALU Control was modified to identify the SLT instruction based on its function code. According to the MIPS Instruction Decoding, the function code of the SLT is 42, so an if-else statement was added to recognize when this function code is inputted and set ALU to perform the SLT operation.

1.2. ALU Modification: In ALU, I added an if-else statement that would handle the SLT operation.

2. ADDI Instruction (Add Immediate)

- The ADDI Instruction was implemented to allow adding the constant value to the content of a register, then stored the result in the destination register. The implementation involved setting the control signals in the Control Logic to support immediate addition without modifying the datapath.

2.1. Control Logic Modification:

- **ALUOp** is set to 2'b10 to indicate add operation.
- **MemRead, MemtoReg. MemWrite** is set to 0 as no memory access in required.
- **RegDst** is set to 0 to direct the result to be written to the register specified in the I-type instruction format.
- **ALUSrc** is set to 1 for the ALU to accept the immediate as input instead of the value from the register**.**
- **RegWrite** is set to 1 to enable the result to be written back to the destination register.
- **Branch, Jump, BranchNot, Upper** is set to 0 since it is not necessary for this operation.

3. J Instruction (Jump)

- The Jump Instruction was implemented to change the program counter by jumping to a specified address. The implementation of the Jump instruction in the single-cycle CPU required modifications to the datapath to allow the new jump address calculation.

3.1. Datapath Modification

- **Jump Address Calculation:** The jump instruction gets the last 26-bit of the instruction, then shifted left by 2, resulting in a 28-bit value. This value is then combined with the upper 4 bits of the program counter after being increment by 4, forming a 32-bit new jump address.

- **MUX for PC Selection:** A new control signal, Jump, is created to control what values is being select as the next program counter. When Jump signal is equal to 1, the MUX selects the jump address. This allows the jump address to being load into the program counter.

3.2. Control Logic Modification:

- **Jump** is set to be 1 when the Opcode field of the instruction is corresponding to the jump operation.

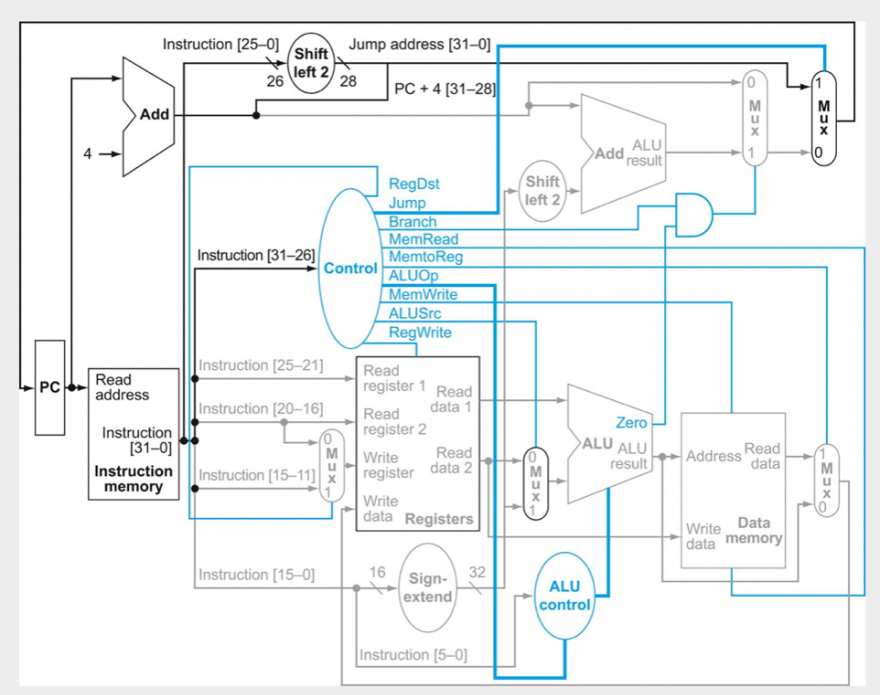


Figure 1: The simple control and datapath extended to handle the jump instruction.

4. BNE Instruction (Branch Not Equal)

- The BNE Instruction allows the program to branch to the specified location if the value of the two source registers is not equal. This was achieved by modifying the control logic and datapath to include the BranchNot signal that would control the branching based on the inequality between register values.

4.1. Datapath Modification:

- **BranchNot Control Signal:** A new control line labeled BranchNot, was added to handle the branching when the source registers' values are not equal. When the BNE operation is being executed, the BranchNot is set to 1, allowing the program counter to update with the branch addres.
- **Non-Zero Flag Check:** The ALU performs a subtraction between the two source registers to determine equality. The Zero Flag in the ALU is normally used to detect when the result of this subtraction is zero. For BNE, a NOT gate was added to the Zero Flag output to create a Non-Zero Flag, allowing the control logic to detect when the registers are unequal.
- **MUX for Branch Control:** A new mux is created to decide whether to take the branch or not. When the instruction is BNE and the BranchNot signal is set to 1, the MUX selects the Non-Zero Flag as the condition for branching, updating the PC with the branch address if the registers are not equal.

4.2. Control Logic Modification:

- **BranchNot** is set to be 1 when the Opcode field of the instruction is corresponding to the BNE operation.

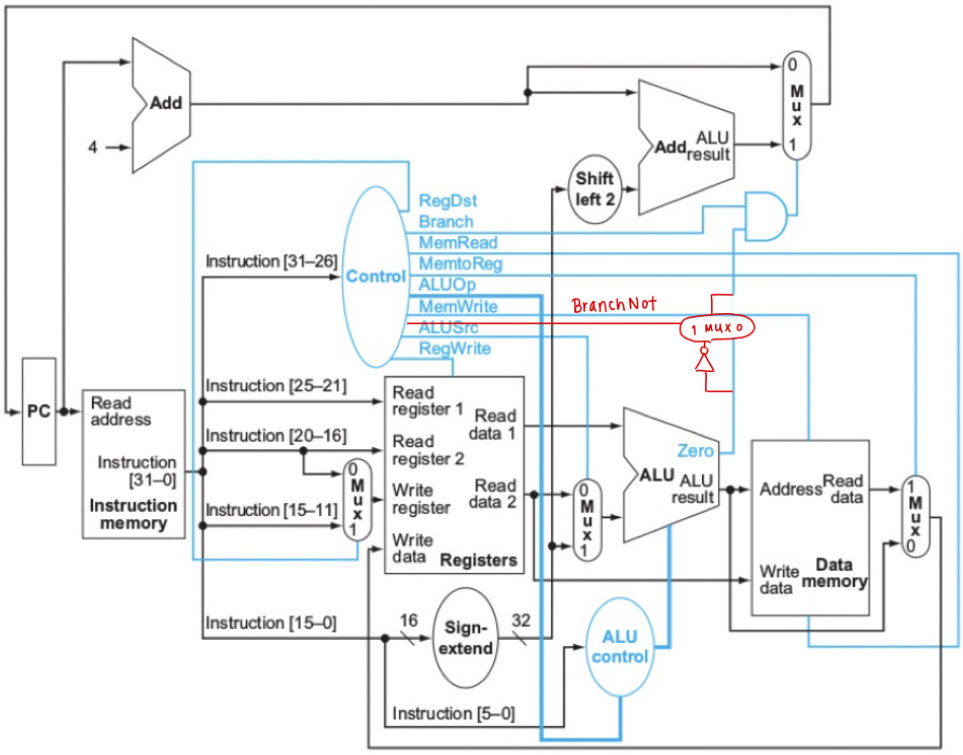


Figure 2: The simple control and datapath extended to handle the branch not equal instruction.

5. LUI Instruction (Load Upper Immediate)

- The LUI Instruction loads a 16-bit immediate value into the upper 16 bits of a register, with the lower 16 bits set to zero. Implementating LUI required changes to the datapath to shift the immediate value and add a new control signal, Upper.

5.1. Datapath Modification:

- **Shift Left 16 Operation:** The 16-bit immediate value of the instruction is shifted left by 16 bits to get the upper 16 bits of a 32-bit register.
- **MUX for Upper Immediate:** A new mux is added to allow the shifeted 16-bit immediate value to be selected and loaded into the register destination, The control signal, Upper, is set to 1 when the LUI instruction is executed, causing the MUX to pass the shifted immediate value to ALU, rather than using the regular immediate.
- **MUX for Zero Input:** Another mux is created to provide a zero value as one of the ALU's inputs when executing the LUI instruction. This MUX passes zero to the ALU, allowing it to perform an addition operation where the shifted immediate value (from the previous MUX) is added to zero. This results in the ALU outputting the shifted immediate value directly, which is then stored in the register file as the final result.

5.2. Control Logic Modification:

- **Upper** is set to be 1 when the Opcode field of the instruction is corresponding to the LUI operation.

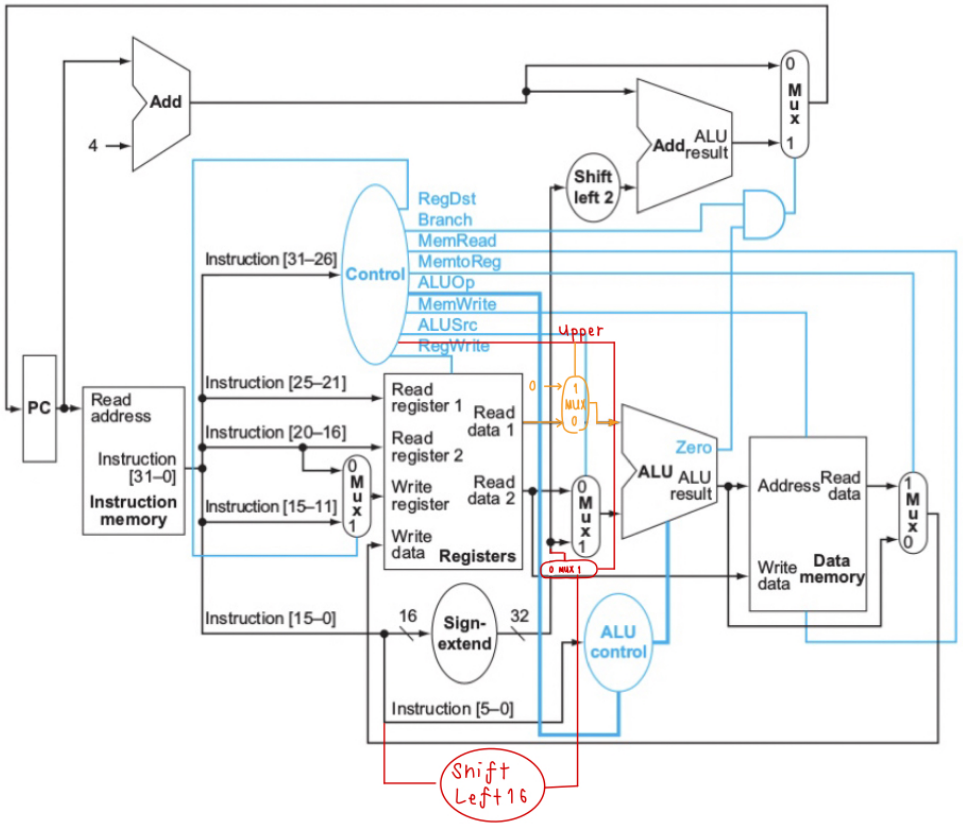


Figure 3: The simple control and datapath extended to handle the load upper immediate instruction.

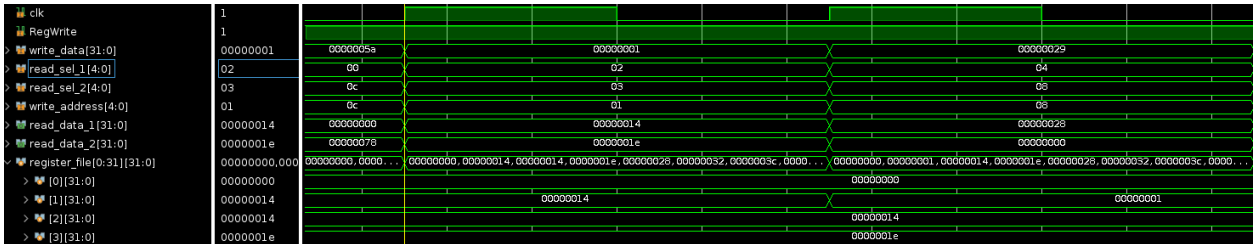
Testing and Results

SLT Testing:

This test validates the Set on Less Than (SLT) instruction. The instruction executed is SLT R1, R2, R3. Here:

- R2 holds the value 0x14.
- R3 holds the value 0x1E.

Since 0x14 is less than 0x1E, the SLT instruction should set R1 to 1. The waveform confirms that the value in R1 is updated to 1 after the SLT instruction completes, indicating successful execution.



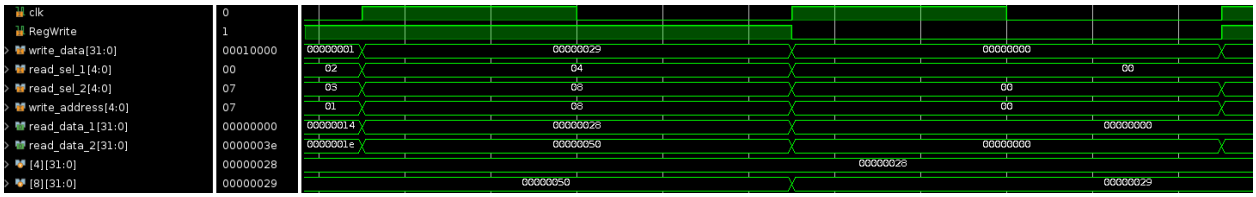
Waveform 1: Set on Less Than Testing

ADDI Testing:

This test validates the Immediate Addition (ADDI) instruction. The instruction executed is ADDI R8, R4, 1. Here:

- R4 holds the value 0x28.
- Immediate value is 1.

After the execution of this instruction, the expected result is that the value 0x29 (the result of 0x28 + 0x1) is stored in R8. The waveform confirms that register R8 is updated to 0x29 following the instruction, which verifies the correct execution of the ADDI instruction.



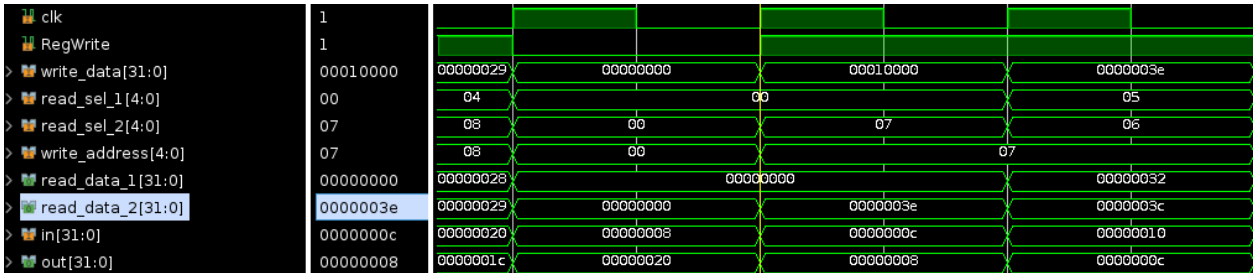
Waveform 2: Immediate Addition Testing

Jump Testing:

This test validates the Jump (J) instruction. The instruction executed is J 2. Here:

- The jump address is specified as 2.
- Since jump addresses are calculated by multiplying the address by 4, the target address after executing this jump instruction is 2 * 4 = 8.

After the Jump instruction is executed, the program counter (PC) should be updated to the instruction address 8. As confirmed by the waveform, the PC correctly reflects this jump to address 8, indicating that the jump operation was successful.



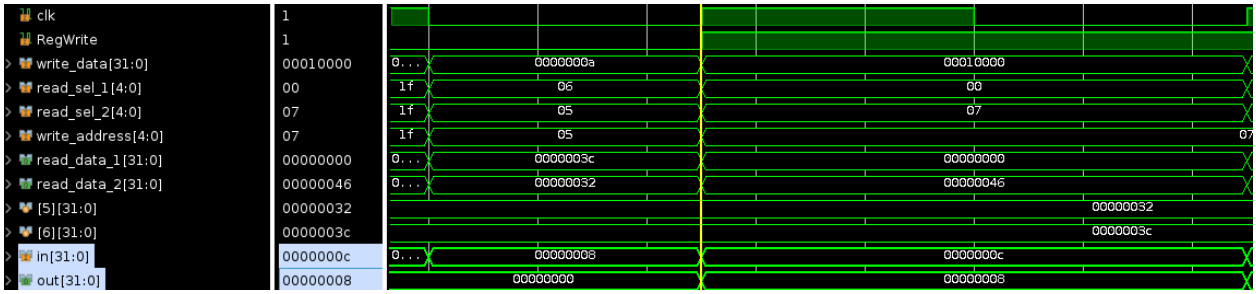
Waveform 3: Jump Testing

BNE Testing:

This test validates the Branch Not Equal (BNE) instruction. The instruction executed is BNE R6, R5, 1. Here:

- R5 holds the value of 0x32
- R6 holds the value of 0x3c
- The branch offset is specified as 1, meaning that if R6 and R5 are not equal, the program counter (PC) should jump to the address calculated by adding 1 * 4 = 4 to the next PC, resulting in address 8.

Since R6 is not equal to R5 in this case, the branch condition is met, and the program counter correctly branches to the target address based on the offset.



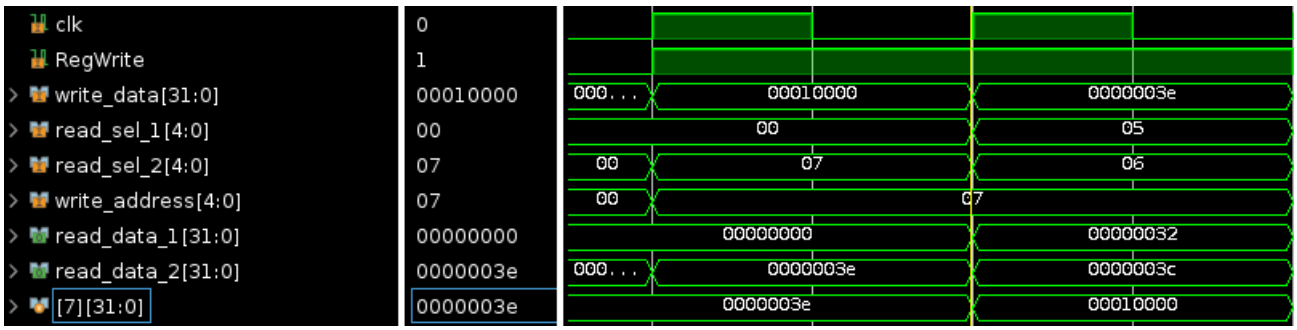
Waveform 4: Branch not Equal Testing

LUI Testing:

This test validates the Load Upper Immediate (LUI) instruction. The instruction executed is LUI R7, 1. Here:

- The LUI instruction loads an immediate value into the upper 16 bits of the specified register while setting the lower 16 bits to zero.
- With an immediate value of 1, the expected outcome is that the upper 16 bits of R7 will be set to 0x0001, resulting in R7 holding the value 0x00010000.

As observed in the waveform, after the LUI instruction executes, R7 is updated to hold the value 0x00010000, which confirms the correct operation of the LUI instruction.



Waveform 5: Load Upper Immediate Testing

Conclusion

The goal of this lab was successfully achieved by implementing and testing additional MIPS instructions, expanding the functionality of a single-cycle CPU design. Each instruction, including SLT, ADDI, J, BNE, and LUI, was integrated into the CPU's control logic and datapath to ensure accurate processing of different operations.

Rawisara Chairat

U65053623

Citation

Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design: The hardware/software interface (MIPS Edition)* (5th ed.). Morgan Kaufmann. Figure 4.24.