Project 2 Design Documentation

Zaibo Wang (zw85) Andrea Yang (yy545)

September 20, 2017

1 Overview

The pipelined MIPS processor has 5 stages:

- Fetch
- Decode
- Execute
- Memory
- Writeback

We'll also discuss in our handling of the following:

- Hazards
- Testing Procedure

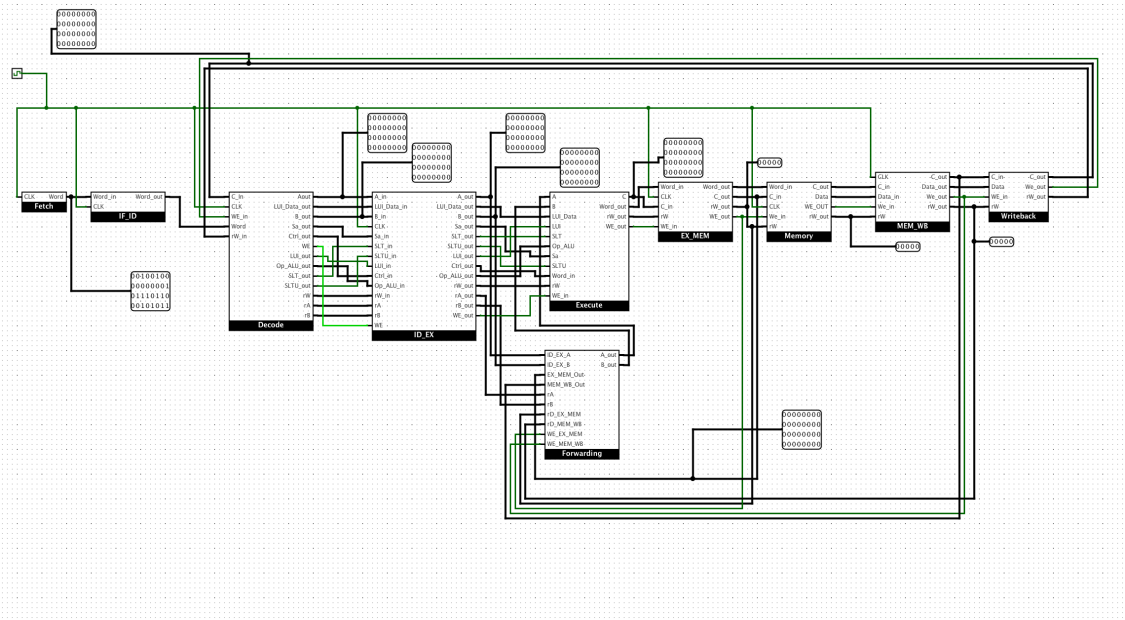

Figure 1: The main circuit, with some added monitoring outputs for testing purposes.

2 The Fetch Stage

In this stage, the PC provides an address in program memory to read from. After this, PC is incremented by 4 for the next word. From program memory, a 32-bit word is read and sent to the pipeline register. For this project, we assume no branching/jumping so PC will always increment by 4.

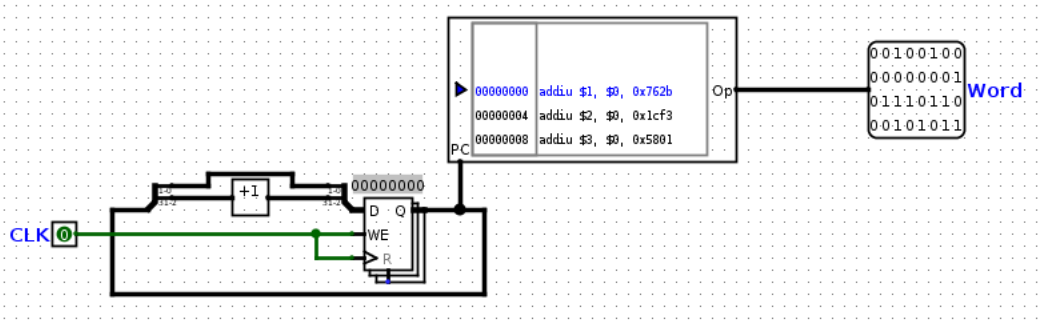

Figure 2: The Fetch stage subcircuit, with Fibonacci loaded in to the RAM

3 Decode

In the Decode stage, we need to parse the instructions to decide the operations. The following subsections will describe the control logic for different outputs

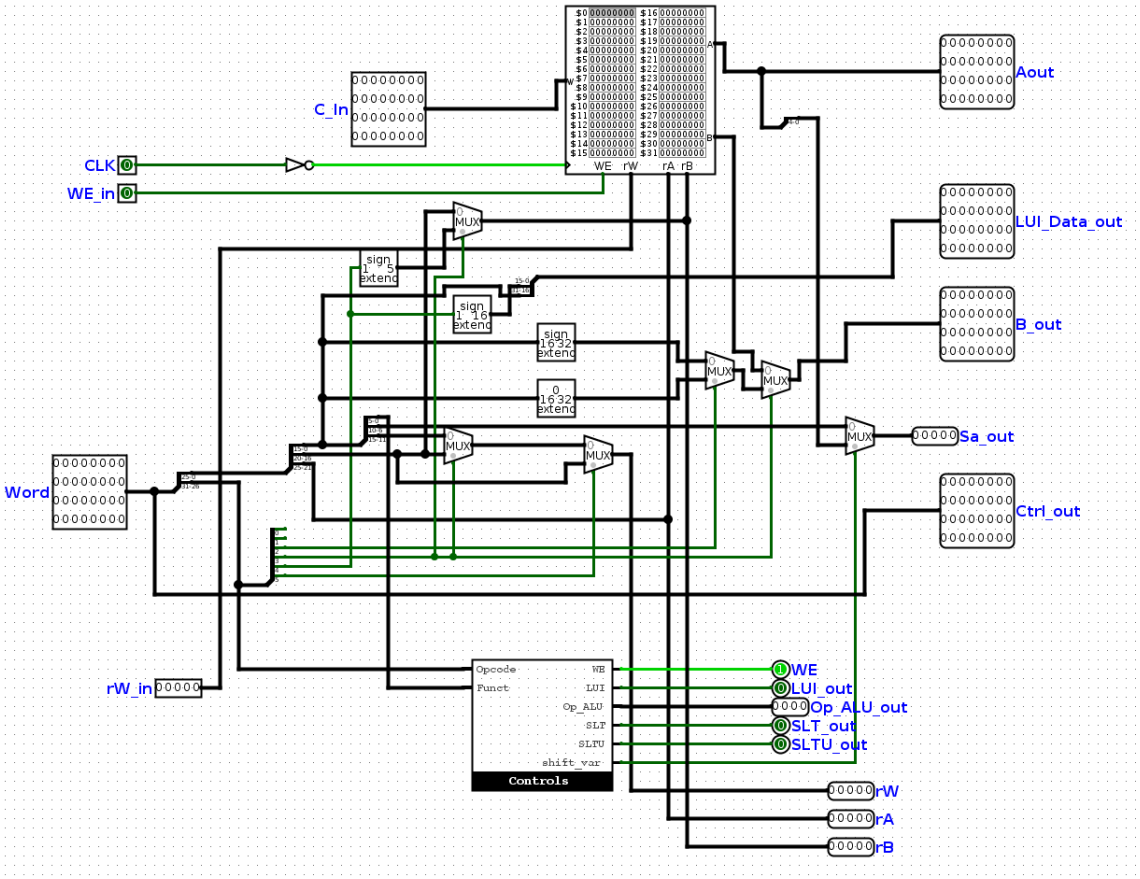

Figure 3: Circuit will be reorganized for clarity later

3.1 A_out

Simply reads the register output for A

3.2 LUI_DATA_out and LUI_out

A load word value is calculated by parsing the 16 low bits in word and 16 0's. This value is passed on along with LUI_out until after the execute stage, at which point there is a mux using the signal LUI_out to choose between the output of execute and LUI_Data. Another possible way to have done this was to take the immediate value from the word, and apply SLL with shamt = 16. We believe that this method was simpler to implement, although it does require passing on more data.

3.3 B_out

The control logic for B is perhaps the most complicated of all outputs. First, the bottom 16 bits of the word are both sign extended and 0 extended into 32 bit values. A mux after this uses bit 2 from Opcode to signal.

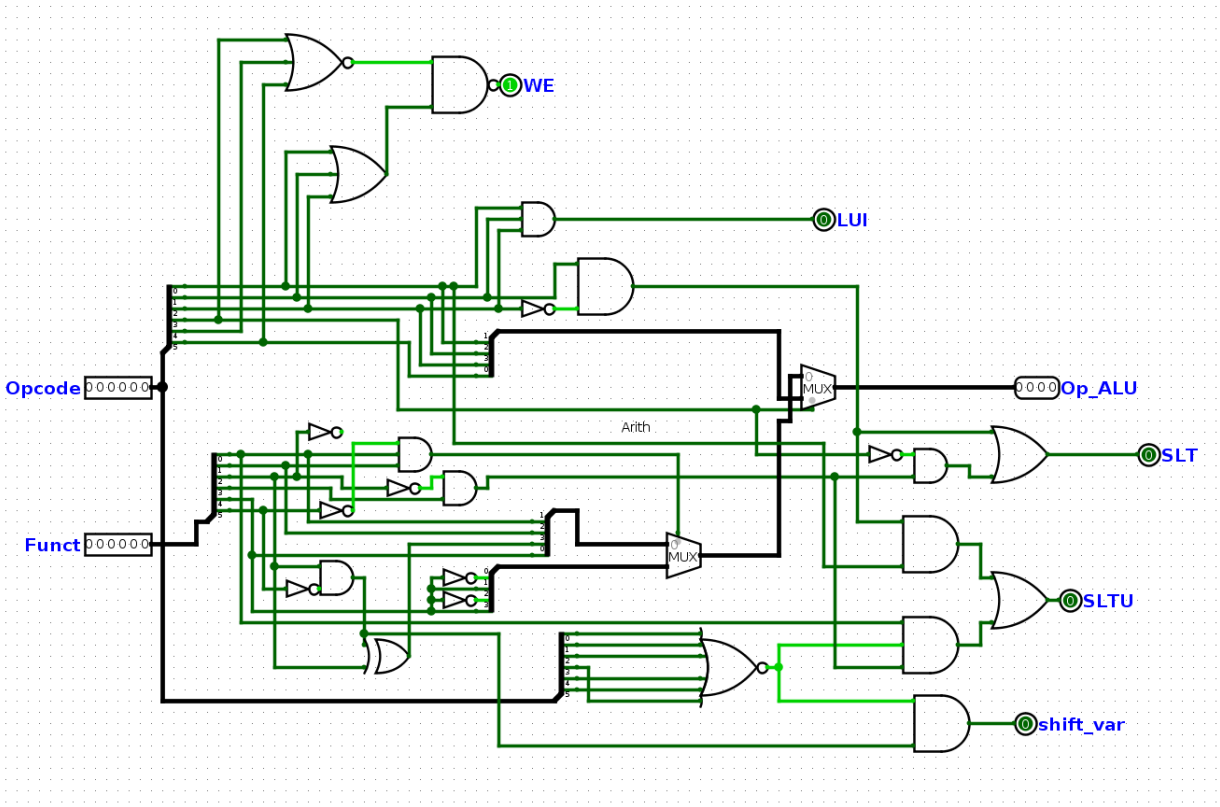

Figure 4: Control is a subcircuit of Decode. It's primary function is to decode all signals needed by the ALU

Sign extend is used for signed immediate values (ADDUI, SLTI, SLTIU) and 0 extend is used for unsigned immediate values (ANDI, ORI, XORI.)

A second mux takes in the input from the previous mux, and the output B from the register. The signal for this mux is bit 3 in Opcode because this bit indicates whether the operation will be I-type or R-type. I-type instructions always use immediate (with exception of LUI) and correspond to bit 3 as 1, and R-type instructions always use B and correspond to bit 3 as 0.

3.4 Sa_out

Sa_out is the shift amount which goes to the ALU in the case of a shift operation. In the case in which the operator is SLLV, SRLV or SRAV, the shift amount must come from the lowest 5 bits of rA. A signal for this is provided by the Controls subcircuit in the form of shift_var which outputs as 1 if Opcode is 000000, bit 2 of function is 1 and bit 5 of function is 0. In all other situations, either the operator is SLL, SRL, SRA, and in these situations, the shift amount is given by bits 6-10 in the word. For non shifting operations, there will still be a shift amount but this will not affect the output of the ALU.

3.5 Ctrl_out

This is the instruction, passed along the pipeline

3.6 WE

WE or Write-Enable is an important signal which prevents instructions such as branch or jump from changing memory and register values. In other words, WE should be 1 when the word is R or I type and 0 when the word is J type. There are of course exceptions to this rule, such as JALR, however, JALR which must be taken care of, if necessary, in the next project. The write enable signal looks for Opcode in which bits 0-2 are 0, or not all bits 3-5 are 0.

3.7 OP_ALU_out

This signal decodes the opcode and/or function to determine what opcode the ALU should execute. The general rule for R-type is take the low-order 3 bits from function and append a 0 to the end. The exceptions to this rule include SRA, SRAV, SLT, and SLTU. SRA and SRAV are handled within Controls. The signal for the mux to indicate SRA is bits 0, 1 AND not 5. If such is the case, the mux will output 0101 (Opcode for SRA) which is essentially hard coded using the 4th bit of function which is always 0. SLT and SLTU are handled separately, detailed below.

3.8 SLT_out and SLTU_out

SLT_out and SLTU_out are two signals which indicate if the operation is a set less than. This is separated from the Opcode for the ALU because SLT operations use comparators instead. Muxes later on in execute will override the output of the ALU and use the SLT value if either of the SLT signals are 1. The signals are made in such a way that if SLTU is 1, SLT must be 1. This is done by using an AND gate for SLTU where one of the inputs is the signal from SLT.

3.9 rW

rW is the address for the register to be written to. It uses one mux (picture not updated) to choose between bits 11-15 and 16-20. This is necessary because the write register for immediate and load operations uses bits 16-20 while R-type instructions have rW as bits 11-15.

3.10 rA

rA is the register for the first register. It is always bits 21-25 so it is split from the word.

3.11 rB

rB is usually bits 16-20, except in the case of I-type instructions. The 3rd bit of Opcode is 1 only for I-types, so it is used as the signal for the mux. If the operation is an I-type, rB should be a 5-bit 0 so it reads out 0. While this does not affect B_out because the I-type case in B_out is handled, this mux is important for the forwarding phase, because there are certain edge cases where if rB is not 'zeroed', the Forwarding Unit detects a data hazard that doesn't exist.

4 Execute

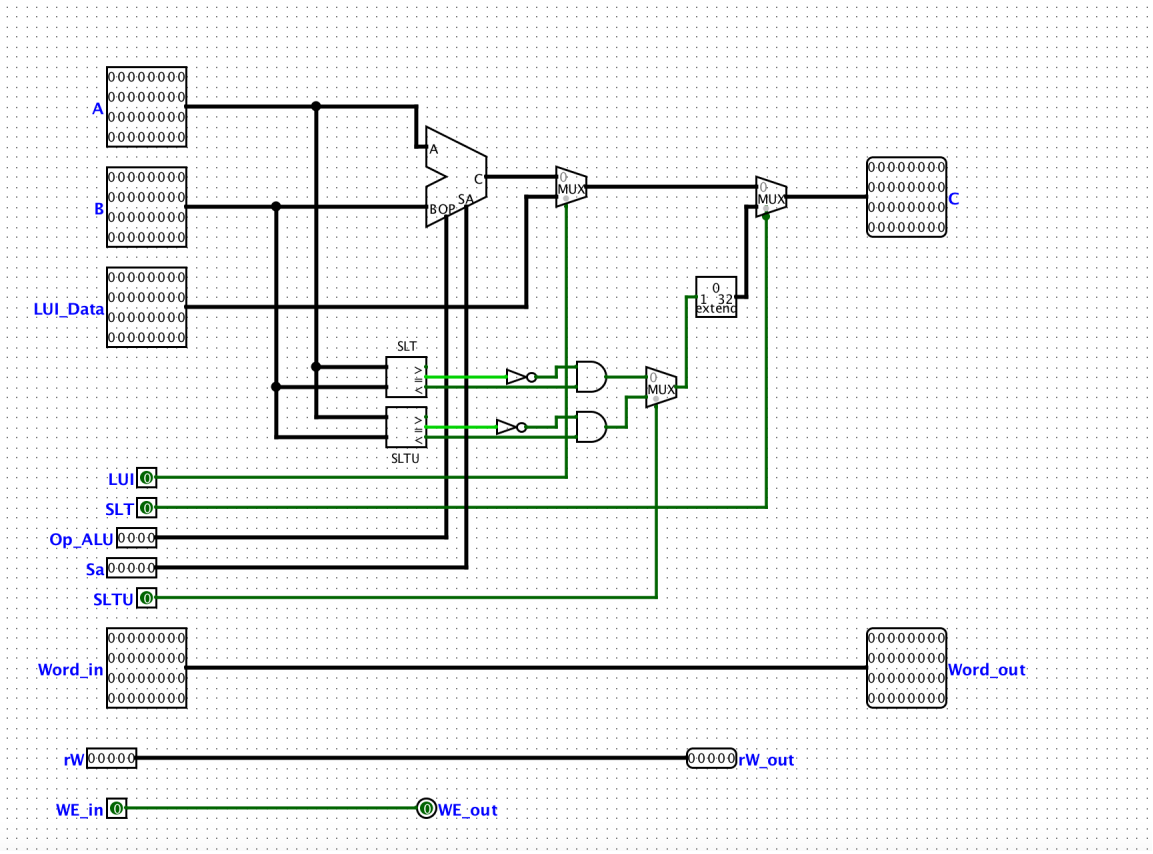

Figure 5: Circuit will be reorganized for clarity later

4.1 ALU

The ALU takes values A, B, Op, SA and outputs C. Input A is always from register memory. Input B has a mux so the input may come from the register memory or directly from the instruction word in the case of an immediate instruction. The operations of the ALU include SRL, SLL, SRA, SRAV, SRLV, SLLV, ADDU, SUBU, AND, OR, XOR, NOR, ADDUI, ANDI, ORI, XORI. This is quite straightforward as all the decoding operations were done in the Decode stage.

4.2 Other operations

There are 5 operations in the Execute which do not use the ALU: SLT, SLTI, SLTU, SLTUI and LUI. However, because the B/immediate conflict was resolved, SLT/SLTI and SLTU/SLTUI behave the same way.

The LUI operation is technically completed in decode. LUI_Data and the LUI signal are inputs to execute. If the LUI signal is 1, the mux after the ALU will take LUI_Data instead of C, the output of the ALU.

Two comparators are used, one for signed and one for unsigned logic. The first mux uses the control signal SLTU to check if the output should come from the signed or unsigned comparator. The bit is then extended to 32 bits and a second mux uses the SLT signal. As mentioned previously, because SLTU uses

takes the SLT signal in an AND gate, SLT must be 1 if SLTU is 1. With this in mind, if SLT is 1, it will take the value of one of the comparators instead of the ALU or LUI. While it appears that SLT may override LUI, there is no case in which both of these signals can both be 1.

5 Memory

For this project, the memory stage does not have to be implemented. The instructions for memory writing can instead be transmitted as a signal to the LCD Video screen.

6 Writeback

Reads previous pipeline register. If writeback is signaled, it stores data from register into the address given by the register.

7 Hazard Handling

In this project the circuit can encounter structural and data hazards.

7.1 Structural

We deal with the problem of reading from and writing to the register file at the same time by negating the clock for the register file, so that write happens in the first half of the cycle, and read happens in the second half.

7.2 Data Hazards

We do both the detecting and the handling of data hazards in a subcircuit unit Forwarding. The logic of the unit follows the forwarding detection logic outlined in the specifications for EX hazards and MEM hazards.

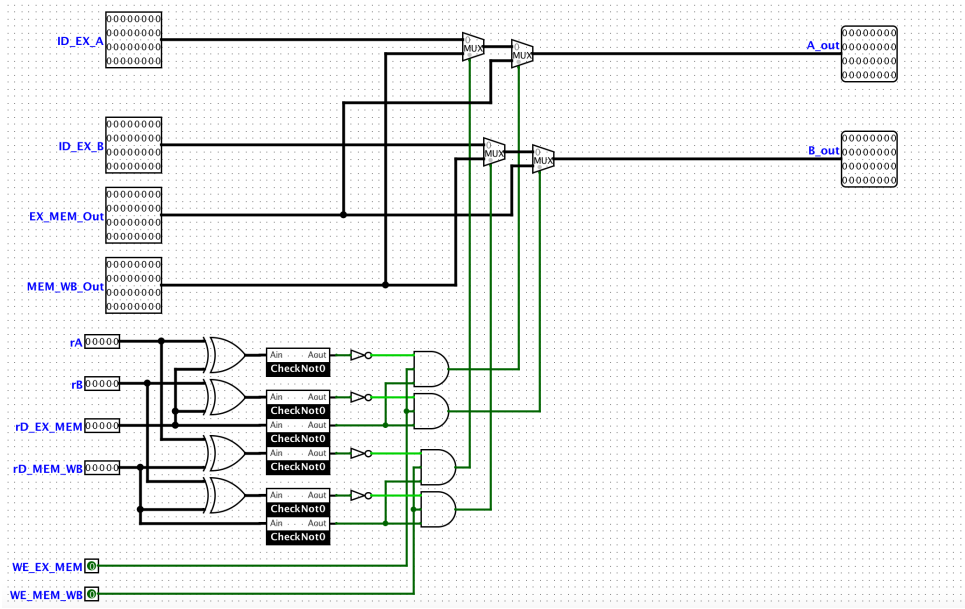

Figure 6: Forwarding is a subcircuit with inputs pulled from from ID/EX, EX/MEM, and MEM/WB.

We use four 2x1 muxes to determine the A and B outputs, with the top two being ForwardA and the bottom two being ForwardB (see diagram) The inputs are chosen based on the following:

- ID_EX_A and ID_EX_B are taken from the outputs of ID/EX, and are each passed as the first input for the first set of 2x1 muxes.
- MEM_WB_Out comes from C_out of MEM/WB, which is the output about to be passed into writeback for writing to the register file, if RegWrite = 1. This is passed as the second input into the first set of 2x1 muxes.The outputs will be passed into the second set of muxes as their first inputs.
- EX_MEM_Out comes from C_out of EX/MEM, which is essentially the result from the ALU in the most recent execution.This is passed as the second input into the second set of 2x1 muxes.

Control signals are determined through the following steps:

- rA and rB inputs correspond to rs and rt in the Word, retrieved from the Decode stage.

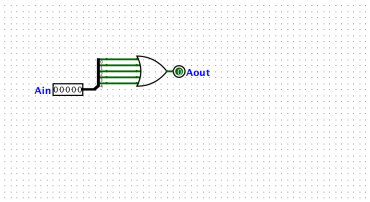

Figure 7: CheckNot0, a helper component

- rD_EX_MEM and rD_MEM_WB are the write destinations in the Word, pulled from EX/MEM and MEM/WB. They're passed into XOR gates along with rA and rB separately.
- The XOR gates and the component CheckNot0 checks if the outputs of the XOR gates or the rDs are zero. If WE is 1, rD is not zero (which is why we negate the output), and matches either rA or rB, then forwarding must happen. And if both hazards occur, we must choose the most recent result, which is EX_MEM_Out.

8 Testing

The following text is the documentation which details how tests where generated for the circuit. Generally, the MIPS interpreter was compared to circuit register values. Assembly code was generated using python, relevant scripts are on Github.

6 lines generate random 16 bit numbers to assign to registers

Next 26 lines take ADDU and SUBU and applies on the first 7 assigned registers. Edge cases such as when both registers are the same are accounted for, but no data hazards will occur

Next 26 lines are in the form ADDU/SUBU $n, $(n-1), $(n-2). This will test for MEM/WB and EX/MEM data hazards INDEPENDENTLY

Register outputs are checked with MIPS interpreter

Next 26 lines are in the form ADDU/SUBU $n, $n, $register(1-6). This will test for when MEM/WB and EX/MEM data hazards both occur, in which case the value from EX/MEM should be taken

Register n is checked after all instructions are run

31 Lines generate a random 16-bit uint. Saves LUI operated number in new register each time to be checked with MIPS interpreter

First 4 lines assign random number to 4 registers. Next 50 lines test ORI, ANDI, XORI with random immediate numbers

The output registers are compared to the register values in MIPS interpreter

First 6 lines assign random number to 6 registers.

Next 26 lines test ORI, ANDI, XORI with random pairs of first 6 registers

The output registers are compared to the register values in MIPS interpreter

Next 26 lines are in the form LOGICAL $n, $(n-1), $(n-2). This will test for MEM/WB and EX/MEM data hazards INDEPENDENTLY

Register outputs are checked with MIPS interpreter

Next 26 lines are in the form LOGICAL $n, $n, $register(1-6). This will test for when MEM/WB and EX/MEM data hazards both occur, in which case the value from EX/MEM should be taken

Register n is checked after all instructions are run

6 lines generate random 16-bit numbers to assign to first 6 registers.

26 lines generate random 0¡shamt¡16 and applies a random shift (SLL, SRA, SRL) to a random register.

The final registers are then compared to MIPS interpreter

First 4 lines assign random numbers to 4 registers. Next 4 lines test SLTI/SLTIU manually with edge cases

The next 30 lines test SLTIU/SLTI with random immediate numbers

The output registers are compared to the register values in MIPS interpreter

4 lines assign random numbers to 4 registers.

The next 30 lines test SLTIU/SLTI with random registers from first 4 generated. Because it's possible to compare the same register to itself, edge case is tested for.

The output registers are compared to the register values in MIPS interpreter

6 Lines generate random unsigned numbers.

26 Lines randomly generate pairs of first six registers and applies SRLV/SRAV/SLLV Register values are compared to those from MIPS interpreter