CSE306: Computer Architecture Sessional

<u>Assignment 4: Assignment on 8-bit MIPS Pipelined Execution</u>

<u>Submitted By:</u>

Group No: 05

Sub-Section: A2

Group Members:

1. Kawshik Kumar Paul (ID: 1705043)
2. Iftekhar Hakim Kaowsar (ID: 1705045)
3. Rasman Mubtasim Swargo (ID: 1705051)
4. Apurba Saha (ID: 1705056)
5. Maisha Rahman (ID: 1705060)

Department: CSE

Level 3 Term 1

Date of submission: 01 July, 2021

<u>Problem Specification :</u>

In this assignment, we designed an 8-bit processor that supports pipelined datapath for a subset of MIPS instruction set. In this design, each instruction was divided into five stages: instruction fetch (IF), instruction decode (ID), execution and address calculation (EX), data memory access (MEM), and write back (WB). Each instruction took up to five clock cycles to be executed.

<u>Introduction:</u>

Instruction pipelining is a technique for implementing instruction-level parallelism within a single processor. Pipelining improves performance by increasing instruction throughput, as opposed to decreasing the execution time of individual instruction, but instruction throughput is the important metric because real programs execute billions of instructions.

Pipelining in MIPS:

- All MIPS instructions are the same length. This restriction makes it much easier to fetch instructions in the first pipeline stage and to decode them in the second stage.
- MIPS has only a few instruction formats, with the source register fields being located in the same place in each instruction. This symmetry means that the second stage can begin reading the register file at the same time that the hardware is determining what type of instruction was fetched.

Pipeline Stages:

Each MIPS instruction is divided into five stages. Which means that up to five instructions will be in execution during any single clock cycle.Thus, we must separate the datapath into five pieces, with each piece named corresponding to a stage of instruction execution:

1. IF: Instruction fetch
2. ID: Instruction decode and register file read
3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back

Pipeline Hazards:

There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards. For this assignment, only data hazard was considered. Data hazards occur when the pipeline must be stalled because one step must wait for another to complete. We considered only EX hazard, MEM hazard and double data hazard.

Complete Block diagram of pipelined datapath:

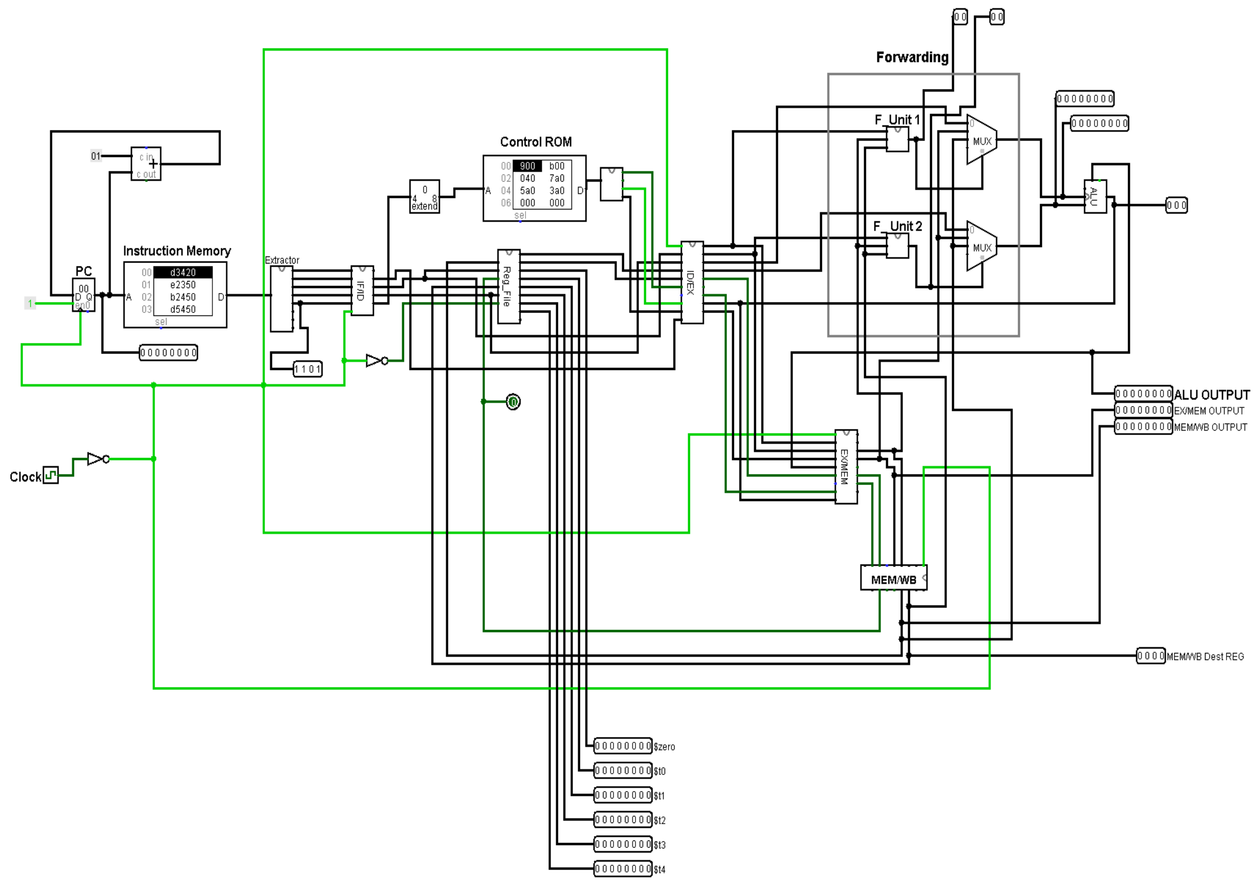

Fig: Main block diagram

Block diagrams and size of pipeline registers:

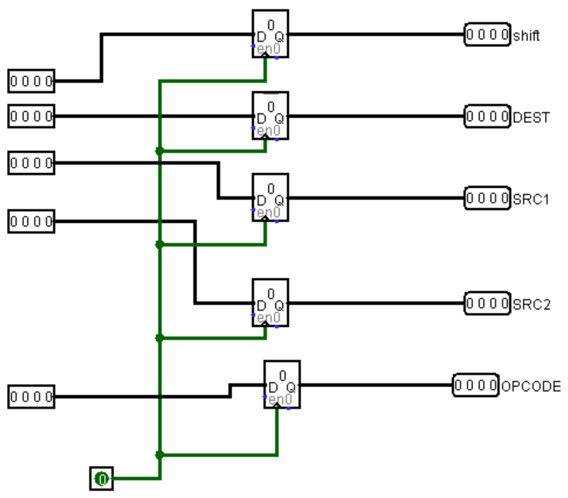

Fig: IF/ID Register

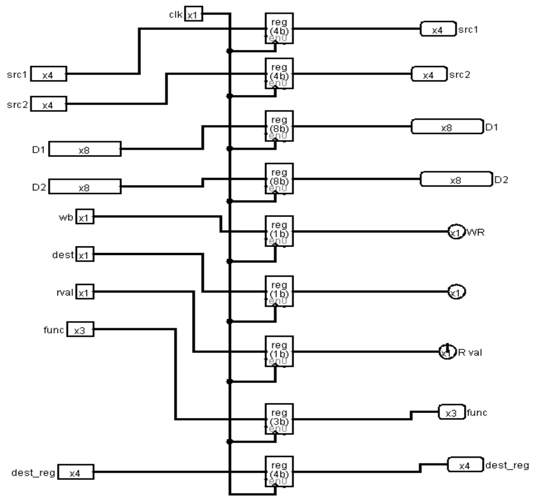

Fig: ID/EX Register

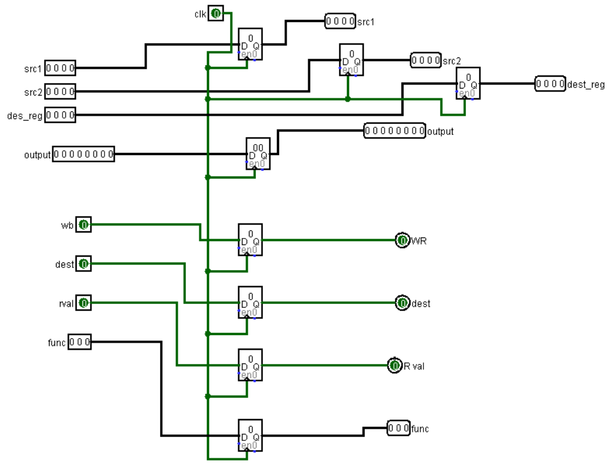

Fig: EX/MEM Register

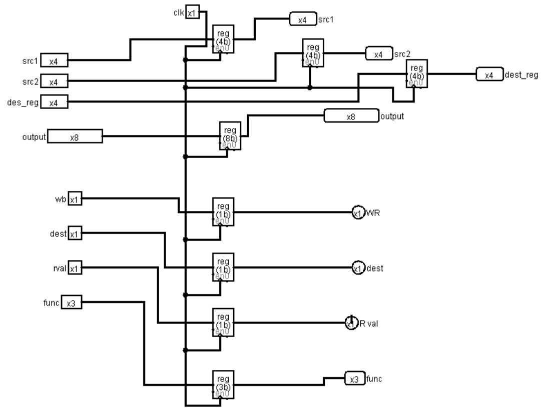

Fig: MEM/WB Register

The sizes of pipelined registers are given ,

- IF/ID 20 bits
- ID/EX 34 bits
- EX/MEM 26 bits
- MEM/WB 26 bits

<u>Mechanism and block diagram of forwarding unit:</u>

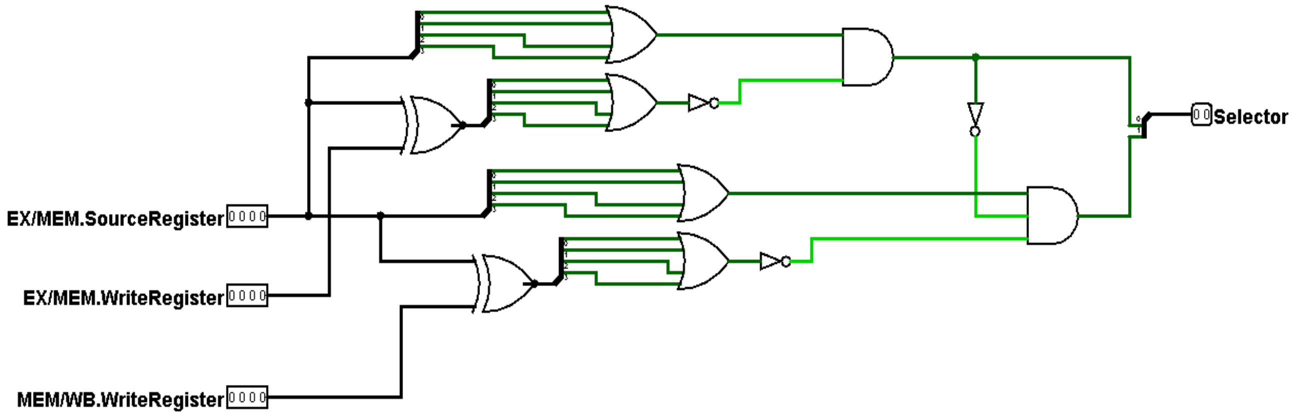

Fig: Forwarding Unit implementation to get ForwardA/ForwardB

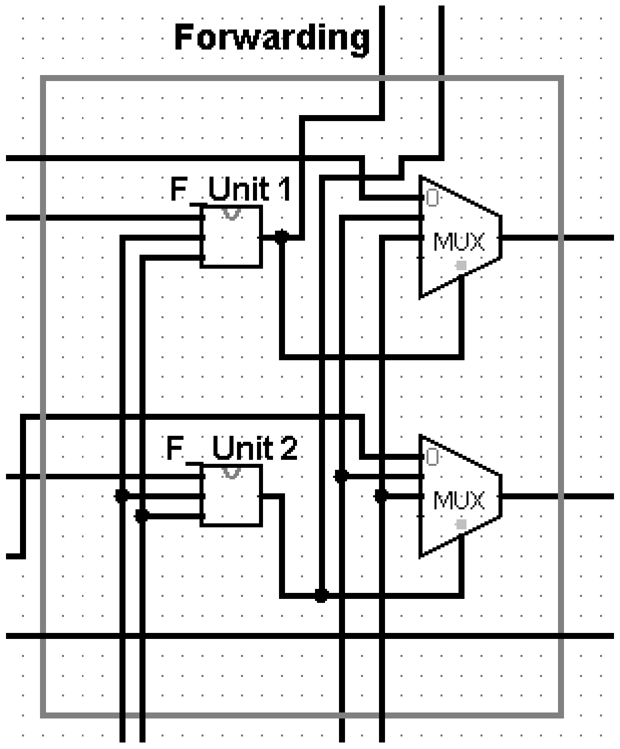

Fig: Using Forwarding Unit in main circuit

The following data hazards were resolved by using forwarding techniques.

- **<u>EX Hazard</u>**: Occurs when the dependent instruction is in the EX stage and the prior instruction is in MEM stage.

if (**EX/MEM.RegWrite** and (**EX/MEM.RegisterRd** ≠ 0) and (**EX/MEM.RegisterRd** = **ID/EX.RegisterRs**))
ForwardA = 01

if (**EX/MEM.RegWrite** and (**EX/MEM.RegisterRd** ≠ 0) and (**EX/MEM.RegisterRd** = **ID/EX.RegisterRt**))
ForwardB = 01

- **<u>MEM Hazard</u>**: Occurs when the dependent instruction is in the EX stage and the prior instruction is in the WB stage.

  if (**MEM/WB.RegWrite** and **(MEM/WB.RegisterRd ≠ 0)** and **(MEM/WB.RegisterRd = ID/EX.RegisterRs)**) **ForwardA** = 10

  if (**MEM/WB.RegWrite** and **(MEM/WB.RegisterRd ≠ 0)** and **(MEM/WB.RegisterRd = ID/EX.RegisterRt)**) **ForwardB** = 10

- **<u>Double Data Hazard</u>**: Occurs when the dependent instruction is in the EX stage and it depends on two prior instructions, one of which is in the MEM stage, and the other is in the WB stage.

  if (**MEM/WB.RegWrite**
  and **(MEM/WB.RegisterRd ≠ 0)**
  and **(MEM/WB.RegisterRd = ID/EX.RegisterRs)**)
  and **(EX/MEM.RegisterRd = ID/EX.RegisterRs)**)
  ForwardA = 01

  if (**MEM/WB.RegWrite**
  and **(MEM/WB.RegisterRd ≠ 0)**
  and **(MEM/WB.RegisterRd = ID/EX.RegisterRt)**)
  and **(EX/MEM.RegisterRd = ID/EX.RegisterRt)**)
  ForwardB = 01

  MUX will select values for ALU inputs as follows:

ForwardA	ALU Input A value
00	ID/EX.valueA
01	EX/MEM.ALUoutput
10	MEM/WB.ALUoutput

ForwardB	ALU Input B value
00	ID/EX.valueB
01	EX/MEM.ALUoutput
10	MEM/WB.ALUoutput

Simulator used along with the version number:

Logisim 2.7.1

Discussion:

In this assignment, an 8-bit mips pipelined execution was designed. It supports the pipelined datapath for a subset of MIPS instruction set. For this purpose, we used basic gates (AND, OR, NOT, NOR), universal gates(XOR, XNOR), and some other necessary gates(MUX, Adder, Subtractor, Bit Extender). The Processor takes a 20 bit binary number (Instruction) and the circuit uses/stores the register values. We used the known technique of designing the 8-bit mips pipelined execution. Moreover, Control ROM made our mapping quite easier, otherwise, it would have been tedious by introducing combinational logic or elsehow. We designed an assembler in CPP and loaded the produced code in another ROM (Instruction Memory). As of our subset of instruction, each instruction will take exactly 5 clock cycles. So, right after the 5th cycle, the first instruction's impact will take place on register files; the second instruction's impact after the 6th cycle, and so on.

Moreover, writing to register files was done in the first half of cycles, and writing to pipelined registers was done in the second half of cycles. If we did both in the similar half, there could be some issues. We would have read and written in register files at the same time.

There is a way in our logisim software which lets us preset register values before starting clock pulses.

In our implementation, three given data hazards were handled. We sketched out the inputs for ALU when different hazards happen.

We followed the provided circuit diagram and implemented the circuit. While designing, we put emphasis on simplifying the circuit. We asserted our design by testing various inputs and matching the corresponding outputs. However, testing the outputs, we successfully finished our simulation.