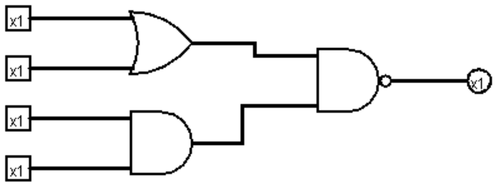
CSE 306

COMPUTER ARCHITECTURE SESSIONAL

Assignment 4 : 8-bit MIPS Pipelined Execution

Section : A(2)



Group no : 03

Student id :

1.1605051

2.1705040

3.1705042

4.1705048

5.1705054

Submission Date: 04/07/2021

Introduction

In this assignment, we are going to design a 8-bit pipelined execution. Pipelining is an implementation technique in which multiple instructions are overlapped in execution. Today, pipelining is nearly universal.

MIPS instructions classically take five steps:

1. Fetch instruction from memory.

2. Read registers while decoding the instruction. The regular format of MIPS instructions allows reading and decoding to occur simultaneously.

3. Execute the operation or calculate an address.

4. Access an operand in data memory.

5. Write the result into a register.

Pipelining is faster that the single cycle implementation. The single cycle implementation takes one clock cycle to finish an entire instruction, on the other hand, pipelining reuses the registers and memory while another instruction is still on the run. And this makes pipelining faster. Pipelining uses one clock cycle for each stage. Firstly, 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. Secondly, 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. Thirdly, memory operands only appear in loads or stores in MIPS. This restriction means we can use the execute stage to calculate the memory address and then access memory in the following stage. And finally, as the operands must be aligned in memory, we need not worry about a single data transfer instruction requiring two data memory accesses; the requested data can be transferred between processor and memory in a single pipeline stage.

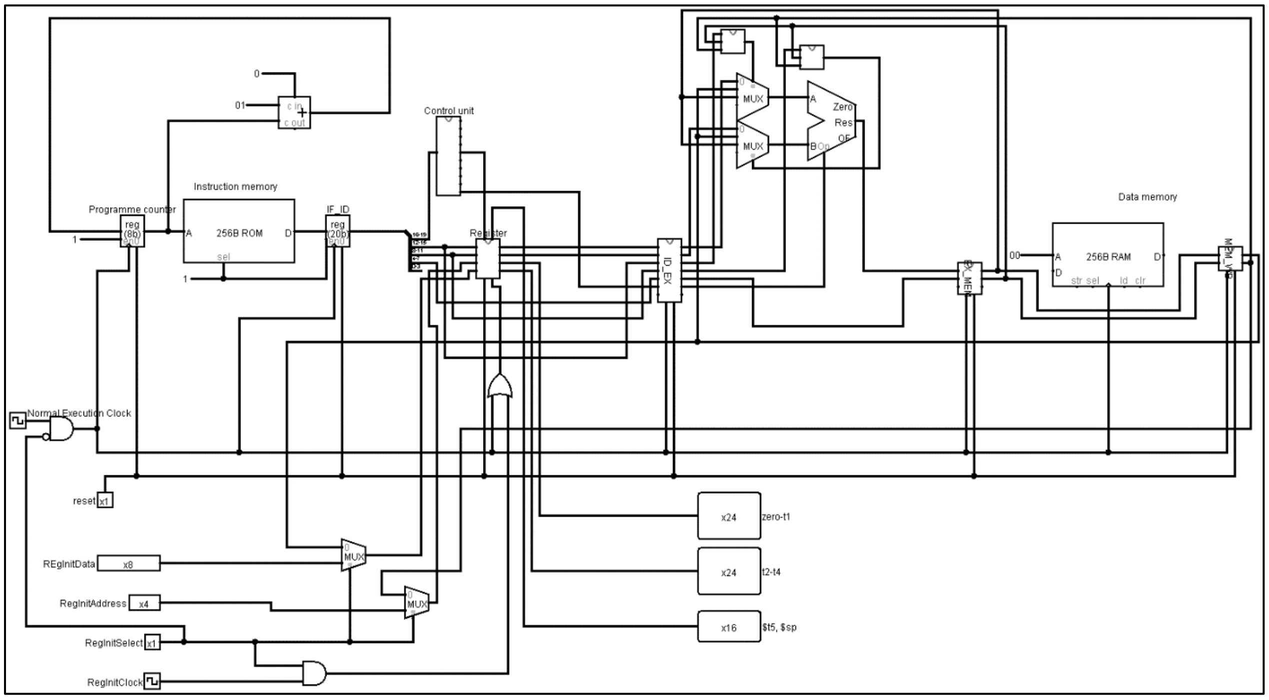
Pipeline Hazards

There are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards, and there are three different types.

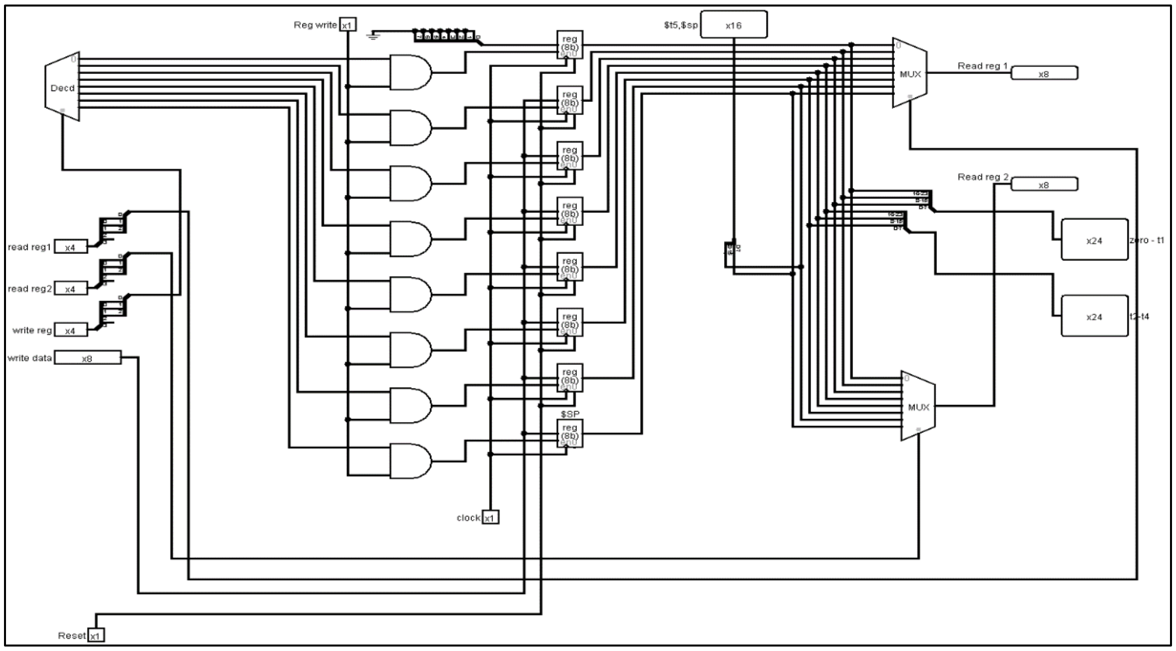
1. The first hazard is called a **structural hazard**. It means that the hardware cannot support the combination of instructions that we want to execute in the same clock cycle.
2. **Data hazards** occur when the pipeline must be stalled because one step must wait for another to complete.
3. The third type of hazard is called a **control hazard**, arising from the need to make a decision based on the results of one instruction while others are executing.

We will use pipelined registers and forwarding unit to eliminate these types of hazards in our design.

Complete Block diagram of Pipelined Datapath



8 bit register



Block diagrams and size of Pipelined registers

IF/ID Register: In this pipeline register, we only need to pass the current instruction which is 20 bits.

Total Size: 20 bits

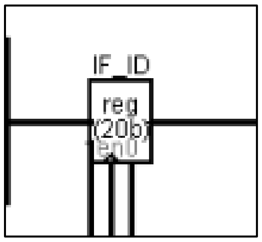


Fig : IF/ID pipeline register

ID/EX Register: This pipeline register consists of total 6 individual registers. They are –

1. Read data1 from rs register: 8bit
2. Read data2 from rt register: 8bit
3. rs register number: 4bit
4. rt register number: 4bit
5. rd register number: 4bit
6. ALU control bit generated by ALU control unit: 3bit

Total size :31 bits

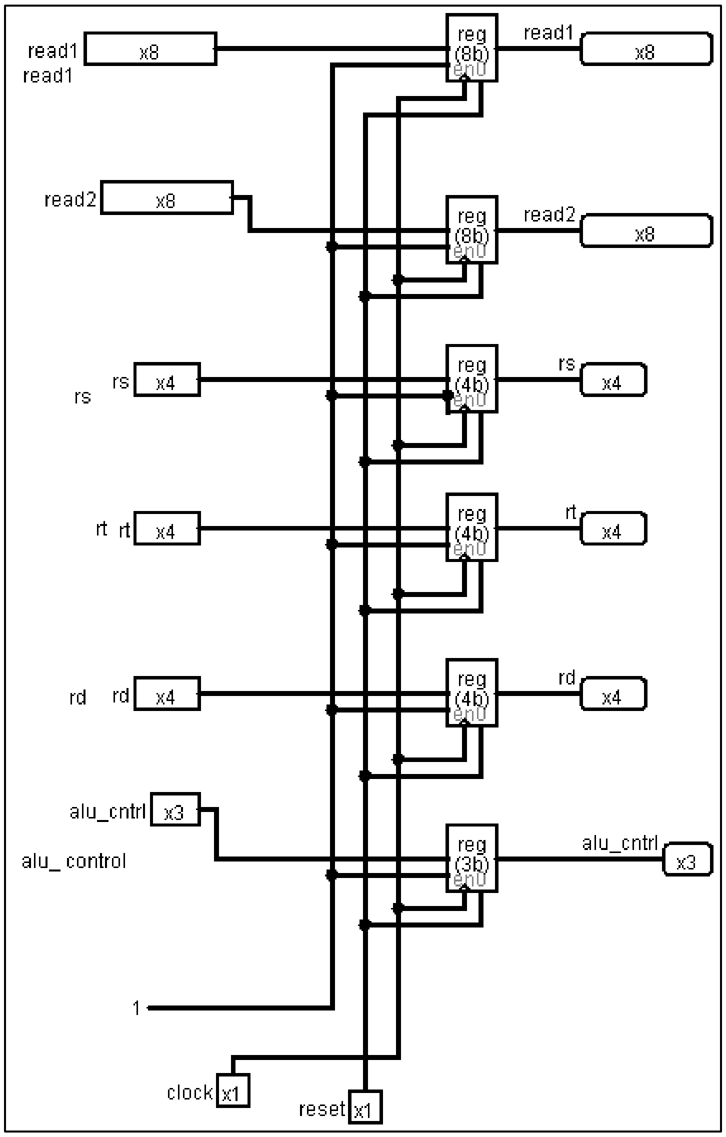


Fig: ID/EX pipeline register

EX/MEM Register: This pipeline register consists of total 2 individual registers. They are –

1. ALU result: 8bit
2. rd register number: 4bit

Total Size: 12 bits

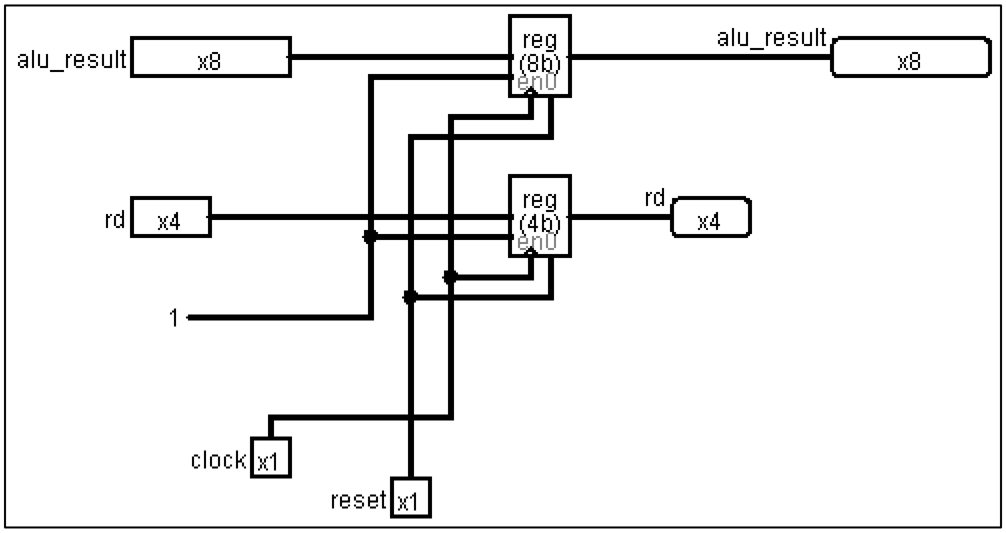


Fig: EX/MEM pipeline register

MEM/WB Register: This pipeline register consists of total 2 individual registers. They are –

1. ALU result: 8bit
2. rd register number: 4bit

Total Size : 12 bits

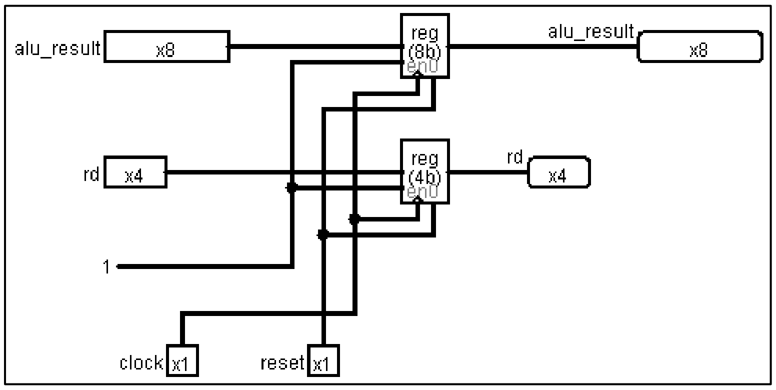


Fig: MEM/WB register

Mechanism and block diagram of Forwarding unit

In execution state, the operands of the Arithmetic logic unit can vary if any hazard occurs. So, we need mux to select operand for the ALU. The required control bit of the mux as follows:

MUX control	Hazard type	Source	Explanation
ForwardA=00	no hazard	ID/EX	the first ALU operand comes from the register file
ForwardA=10	data hazard	EX/MEM	the first ALU operand is forwared from the prior ALU result
ForwardA=01	memory hazard	MEM/WB	the first ALU operand is forwared from data memory or an earlier ALU result
ForwardB=00	no hazard	ID/EX	the second ALU operand comes from the register file
ForwardB=10	data hazard	EX/MEM	the second operand is forwared from the prior ALU result
ForwardB=01	memory hazard	MEM/WB	the second ALU operand is forwared from data memory or an earlier ALU result

Forwarding units are respectively generating the MUX control bit for the selection of ALU operands . In forwardA unit we are comparing the current instruction’s sourse register with the previous instruction’s destination register to determine the data hazard . And we are comparing the current instruction’s source with the destination register of instruction which is in write back in register state in that moment causing memery hazard. Similarly , ForwardB unit is generating control bit for second ALU operand determining mux.

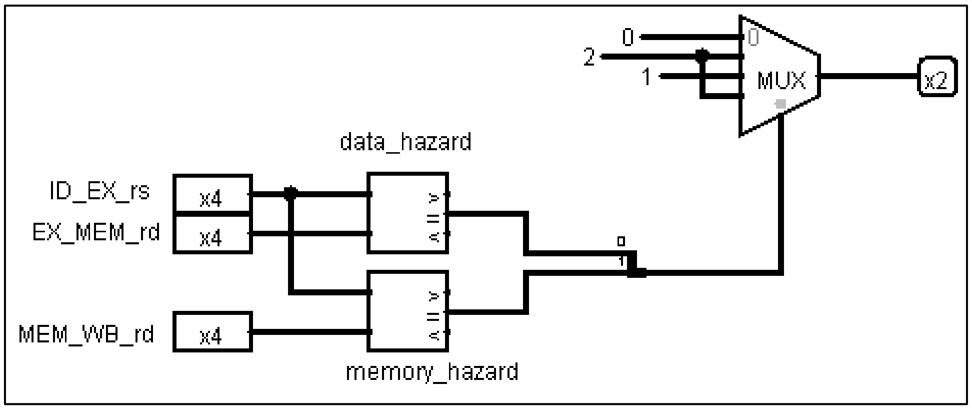


Fig: Forward_A unit

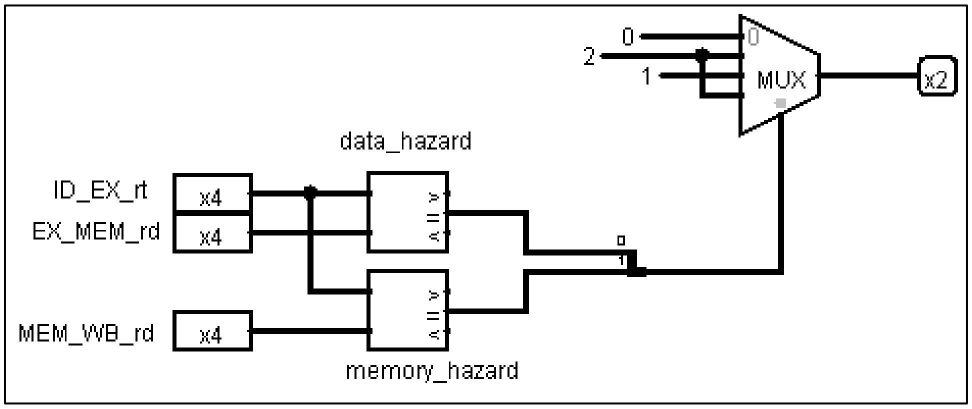


Fig: Forward_B unit

Simulator used

Logisim (Version: 2.7.1)

Discussion

In this assignment, we have designed and simulated a 8-bit pipelined processor of the MIPS architecture. The processor takes a 20-bit binary data which is the instruction and executes the instruction using the registers, data memory and clock cycle. We have designed an assembler using Python to generate the binary instruction from the assembly code and loaded that into a ROM. The ROM is the instruction memory from where the program is fetching the instructions. We have used four pipeline registers and forwarding unit to implement the pipelining.