CSE 306
Computer Architecture Sessional

8-bit MIPS Pipelined Execution

Group: 02
Section: A2

Group Members:
1705036
1705037
1705038
1705039
1705044

1 Introduction

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) with instruction set architecture (ISA). Pipelining is an implementation procedure where multiple instructions are executed simultaneously in an overlapped fashion. The vanilla MIPS architecture would complete executing a single instruction before processing the subsequent one, thus the execution time gets restricted by the instruction with the longest duration. In a pipelined architecture, the whole process is divided into stages and each of them will be executing a part of different instructions. Thus an instruction wouldn't need to wait for its predecessor to finish executing, it can enter into a stage preceding the stage its predecessor is in. In this manner, the overall execution time of instructions will decrease drastically resulting in a more efficient implementation.

In this assignment, we have to design an 8-bit processor that implements the MIPS instruction subset with support for pipelined datapath. The datapath is separated into 5 stages :

- **IF** : Instruction Fetch
- **ID** : Instruction Decoder
- **EX** : Execution and address calculation
- **MEM** : Data memory access
- **WB** : Write Back

Each of these stages will take one clock cycle to execute. Thus a single instruction will take 5 clock cycles to be fully executed and 5 instructions will be able to execute partially at the same time on a single clock cycle. Furthermore, only 4 R-type instructions are considered for this design: **add, sub, and, or**.

Lets consider the case where an instruction will need a value calculated in one of its preceding instructions which might not have finished executing due to the pipelined datapath architecture. Thus executing the current instruction would result in erroneous values. These events are called hazards. In this assignment, only data hazards are considered. These can be detected and resolved using forwarding techniques. The types of data hazard considered and their reason of occurrence are provided below :

- **EX Hazard** : Dependent Instruction is in EX stage and the Prior Instruction is in MEM stage.
- **MEM Hazard** : Dependent Instruction is in the EX stage and the Prior Instruction is in WB stage.
- **Double Data Hazard** : Dependent Instruction is in EX stage and it depends on two Prior Instructions, one of which, is in MEM stage, and the other is in WB stage.

In this manner, an efficient pipelined version for a 8-bit MIPS architecture is built with provisions for detection and resolution of the 3 data hazards mentioned.

2 Complete Block diagram of pipelined datapath

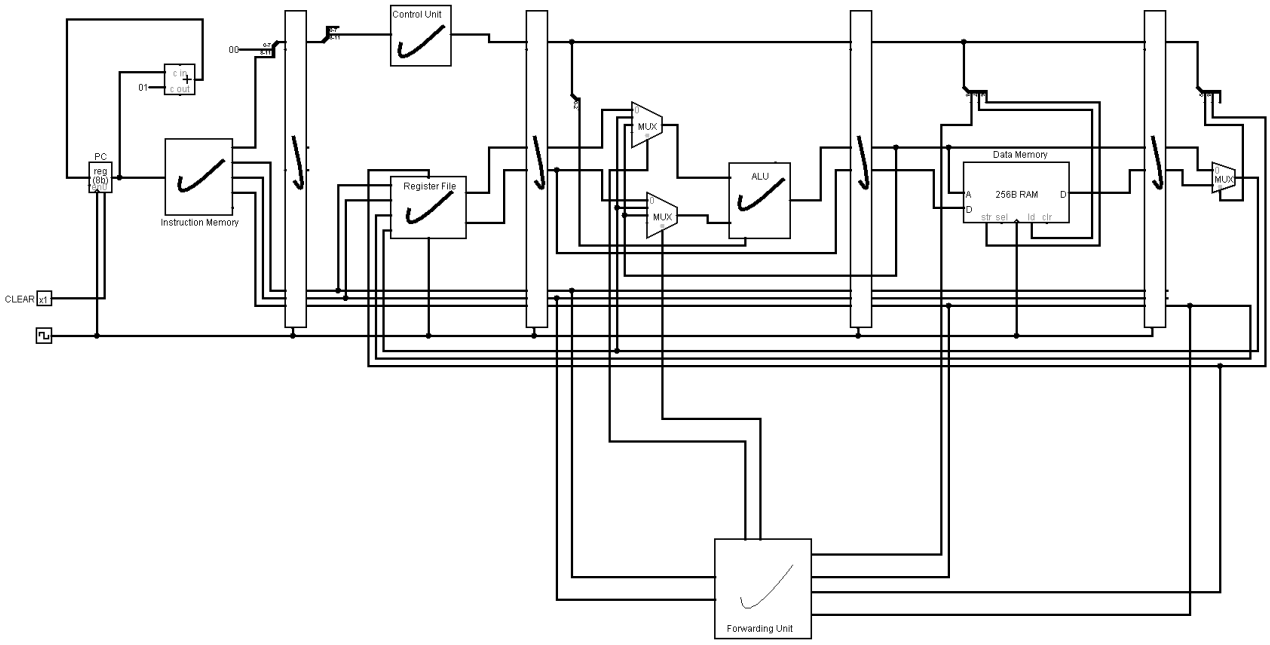

Figure 1: 8-bit MIPS pipelined processor

3 Block diagrams and size of pipeline registers

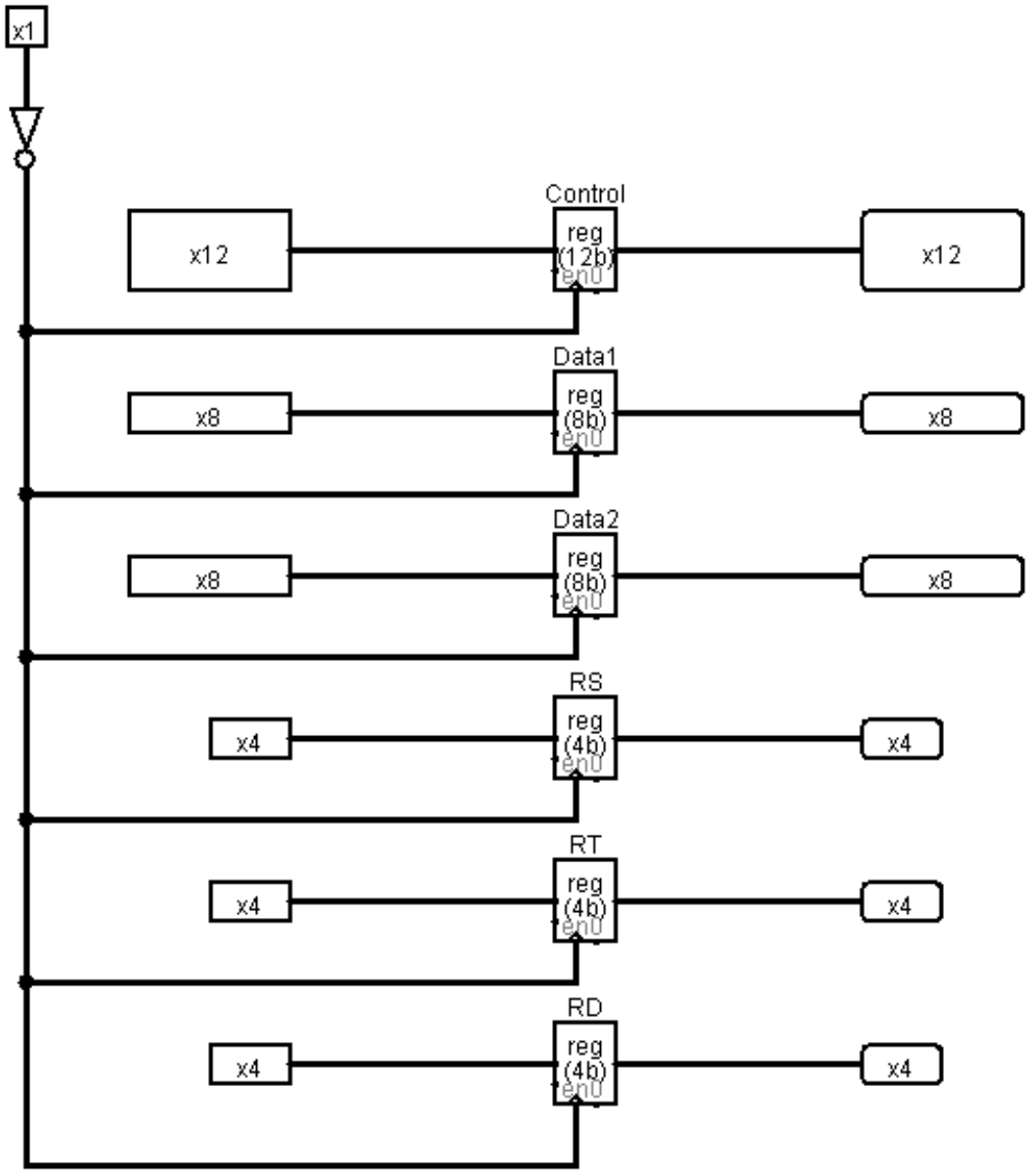

Figure 2: Pipeline Register

In our pipeline registers, we have used a collection of registers to create a pipeline. Pipelines have been used between any two stages of instruction flow. Our pipeline register consists of -

- **One** 12 bit Register For Passing Control Bits
- **Two** 8 bit Registers For Passing Data Bits
- **Three** 4 bit Registers For Passing The Source and Destination Registers (rs, rd & rt)

4 Mechanism and block diagram of forwarding unit

We implemented the forwarding unit so that it can detect data hazards and generate control bits to allow data forwarding.

- Implementation of EX hazard Checker

  Conditions of EX hazard :
  if (EX/MEM.RegWrite and (EX/MEM.RegisterRd!=0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
  ForwardA = 10
  if (EX/MEM.RegWrite and (EX/MEM.RegisterRd!=0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
  ForwardB = 10

  We implemented these conditions with a circuit. A few rows of the truth table of these conditions are shown below :

EX/MEM.RegWrite	EX/MEM.Rd				ID/EX.Rs				ForwardA
X	0	0	0	0	X	X	X	X	00
1	0	0	0	1	0	0	0	1	10
1	0	0	1	0	0	0	1	0	10
1	0	1	0	0	0	1	0	0	10
1	1	0	0	0	1	0	0	0	10

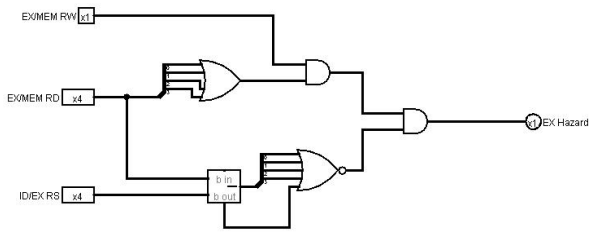

Figure 3: EX Hazard Checking Unit

We used a subtractor to check if the ID/EX.RegisterRs and EX/MEM.RegisterRd are equal. We also used an OR gate to determine if all bits of EX/MEM.RegisterRd is 0. The same procedure was used to generate forward B bits where ID/EX.RegisterRt was used instead of Rs.

- Implementation of MEM Hazard and Double Data Hazard:

  The checking of MEM Hazard and Double Data Hazard were implemented using the same circuit.

  Conditions of these hazards occuring are :
  if (MEM/WB.RegWrite and (MEM/WB.RegisterRd!=0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd!=0) and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
  ForwardA = 01
  if (MEM/WB.RegWrite and (MEM/WB.RegisterRd!=0) and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd!=0) and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
  ForwardB = 01

  Since some of these conditions match the structure of the conditions of the EX Hazard checker, we reused some components of the previous circuit and implemented these conditions using the following circuit.

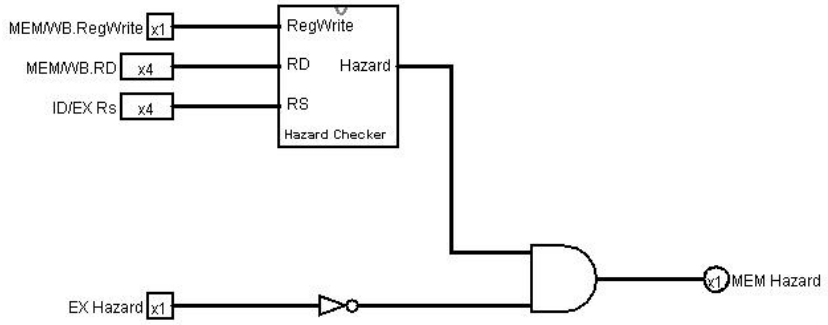

Figure 4: MEM and DD Hazard Checking Unit

Finally, these hazard checkers were integrated to create a component which generated the forwarding control bits based on hazards.

EX Hazard	MEM Hazard	Forward
0	0	00
0	1	01
1	0	10
1	1	10

The last condition on the truth table represents double data hazard.

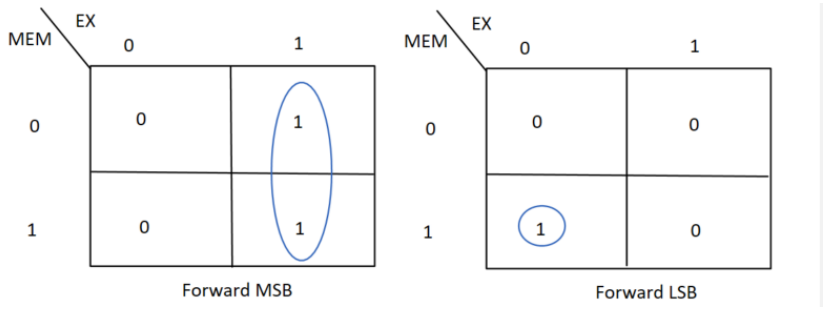

Figure 5: K-MAP for generating forwarding bits

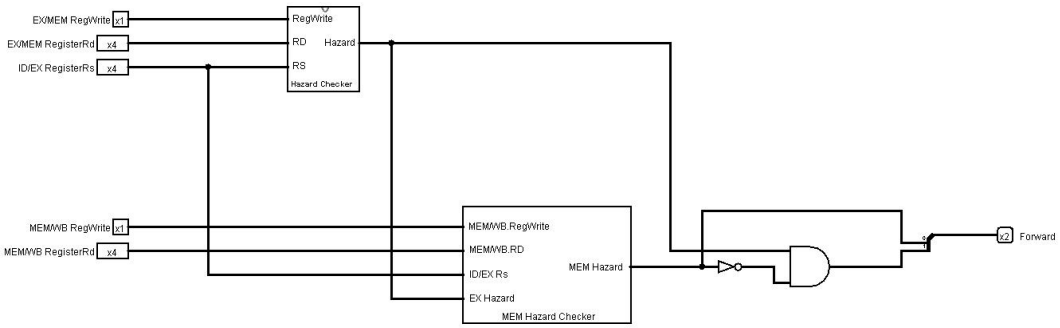

Figure 6: Circuit for generating forwarding bits

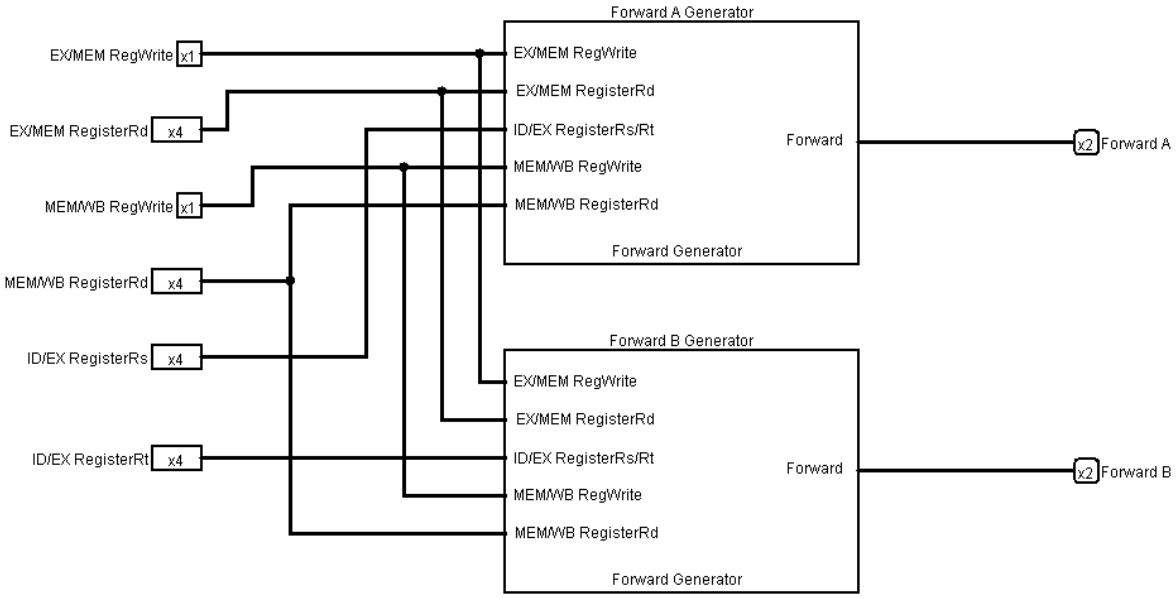

Figure 7: Block Diagram of Forwarding Unit

5 Discussion

In this assignment, we had to design and simulate an 8-bit processor of the MIPS architecture that supports **pipelined datapath** for a subset (add, sub, or, and) of MIPS instruction set. The required design consisted of many components, all of which we had to implement and integrate to complete the processor.

We also had to implement a **Forwarding Unit** to detect Data Hazard and generate control bits to allow data forwarding. As this step was relatively complicated compared to the rest of the circuit, we tried to be more cautious and generated various test cases to check for the Three Hazards (EX Hazard, MEM Hazard & Double Data Hazard) in our design. We provided all the clocks from a single clock source.

We tested and simulated our circuit with various test cases and noticed the pipelined datapath in action. Our implementation was made as efficient as possible.

6 Software Version

Logisim-Win-2.7.1