Bangladesh University of Engineering and Technology

CSE306
Computer Architecture Sessional

Assignment on 8-bit MIPS Design and Simulation

Submitted By:

Group: A1-1

Kowshic Roy
(S201705001)
Sheikh Azizul Hakim
(S201705002)
Mahdi Hasnat Siyam
(S201705003)
Saiful Islam
(S201705006)
Khandokar Md. Rahat Hossain
(S201705024)

Department of CSE

Submitted To:

Abu Wasif
Assistant Professor

Madhusudan Basak
Assistant Professor

T. M. Tariq Adnan
Assistant Professor

Date of Submission
July 4, 2021

1 Introduction

MIPS *(Microprocessor without Interlocked Pipelined Stages)* is a reduced instruction set computer (RISC) instruction set architecture. In this assignment, we had to design an 8-bit processor that implements a subset of MIPS instruction set using a pipelined datapath. In this design, each instruction is divided into five stages: instruction fetch (IF), instruction decode (ID), execution and address calculation (EX), data memory access (MEM), and write back (WB). Since the datapath is pipelined, each stage is supposed to be executed in one clock cycle. Hence, the length of the clock cycle equals the maximum time to execute any single stage and we need registers to hold values between the stages. Each instruction thus may take up to five clock cycles to be executed. The main components of the processor are as follows: instruction memory, data memory, register file, ALU, control unit, four pipeline registers, and a forwarding unit.

1.1 Design Specification

Required specifications for our design are given below-

- Address bus and data bus which are multiplexed in the design.
- Each data and address is kept a size of 8 bits for simplicity.
- An 8 bit ALU is required.
- Register file with temporary registers `$zero,$t0,$t1,$t2,$t3,$t4`.
- Pipeline registers are used between any two stages of the instruction. A pipeline register can be a single register or a collection of registers.
- As the *control unit*, a ROM is used in the design.
- Each instruction is fetched and executed in atmost five clock cycles.

2 Instruction Set

Only the R-type instructions were required to be implemented.

OPCODES	**TYPE**	**INSTRUCTION**	**HEX CODES (Control Unit)**
1000	R	add	0x831
1001	R	sub	0x821
1011	R	or	0x851
1100	R	and	0x861

Table 1: Required Instruction Set

However, with 4 bits of opcode, we can execute upto 16 instructions. For the facilitation of data loading and the illustration of the data memory's role in pipelining, our design supports some additional instructions.

OPCODES	**TYPE**	**INSTRUCTION**	**HEX CODES (Control Unit)**
0000	I	andi	0x063
0001	I	addi	0x033
0110	I	ori	0x053
0111	I	subi	0x023
1101	I	sw	0x0D6
1110	I	lw	0x0DB

Table 2: Additional Instruction Set

According to the opcodes, a hex code is loaded in the control unit which is a ROM in our specific design.

Another ROM is used as instruction memory unit where we load the hex code assembled from the written assembly program. This code gets divided as:

R-type :	Opcode(4-bits)	Src Reg1(4-bits)	Src Reg2(4-bits)	Dest Reg(4-bits)	ShAmt(4-bits)
I-type :		Src Reg(4-bits)	Dest Reg(4-bits)	Address/Immediate(8-bits)	

Thus each instruction is fetched from instruction set memory to control unit and further operations in respective units of our MIPS design.

2.1 Hazards

A major pitfall of pipelined designs arise from hazards. In our design, only data hazards have been considered:

EX Hazard An instruction depends on the data calculated in its preceding instruction.

MEM Hazard An instruction depends on the data calculated in its second preceding instruction.

Double Data Hazard An instruction depends on the data calculated in both its preceding and second preceding instruction. In that case, the value of the preceding instruction gets prioritized.

2.1.1 Conditions of Hazards

The condition for EX hazard to occur in the first source register is

$$\begin{gathered}\text{C1} = \text{EX/MEM.RegWrite AND}\\ \text{EX/MEM.RegisterRd} \neq 0 \text{ AND}\\ \text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRd}\end{gathered} \tag{1}$$

and the condition for EX hazard to occur in the first source register is

$$\begin{gathered}\text{C2} = \text{EX/MEM.RegWrite AND}\\ \text{EX/MEM.RegisterRd} \neq 0 \text{ AND}\\ \text{EX/MEM.RegisterRd} = \text{ID/EX.RegisterRt}\end{gathered} \tag{2}$$

The condition for MEM hazard to occur alone in the first source register is

$$\begin{gathered}\text{C3} = \text{MEM/WB.RegWrite AND}\\ \text{MEM/WB.RegisterRd} \neq 0 \text{ AND}\\ \text{MEM/WB.RegisterRd} = \text{ID/EX.RegisterRd}\end{gathered} \tag{3}$$

and the condition for MEM hazard to occur alone in the first source register is

C4 = MEM/WB.RegWrite AND
MEM/WB.RegisterRd $\neq$ 0 AND
MEM/WB.RegisterRd = ID/EX.RegisterRt

(4)

However, for handling double data hazards, MEM hazard will occur only if EX hazard does not occur. Hence MEM hazard will occur for the first source register if (3) holds true and (1) holds false, and will occur for the second source register if (4) holds true and (2) holds false.

3 Implementation

3.1 Simulator

We used **Digital** v0.26.1, which is an open-source software for simulating digital circuits. The software can be found at: `https://github.com/hneemann/Digital`.

3.2 Complete Block Diagram

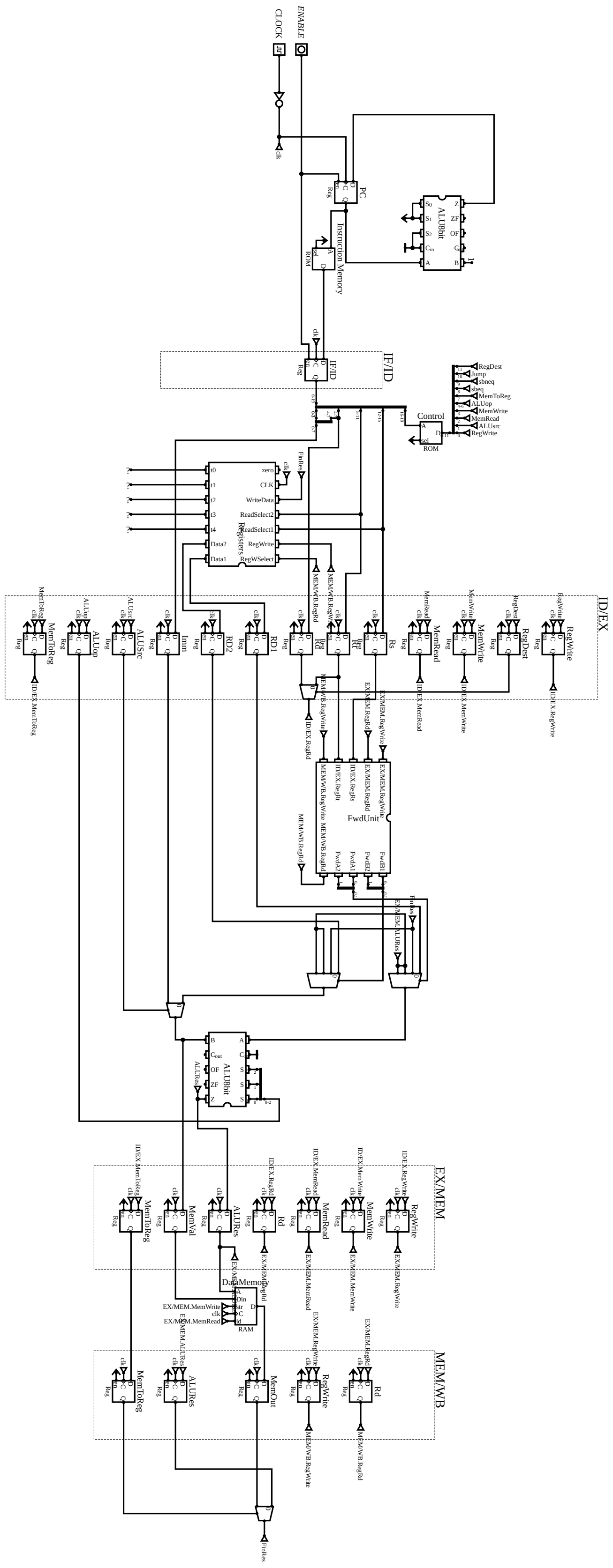

Figure 1: Block Diagram of a pipelined 8-bit MIPS-subset processor

4 Pipeline Registers

4.1 Overview

Four sets of registers have been used for facilitating pipelining.

Set of Registers	No of Bits
IF/ID	20
ID/EX	45
EX/MEM	24
MEM/WB	22

Table 3: Overview of Pipelining Resisters

4.2 IF/ID Register

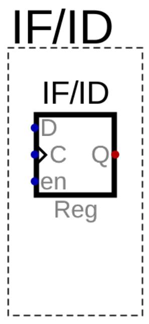

Figure 2: IF/ID Register

4.3 ID/EX Register(s)

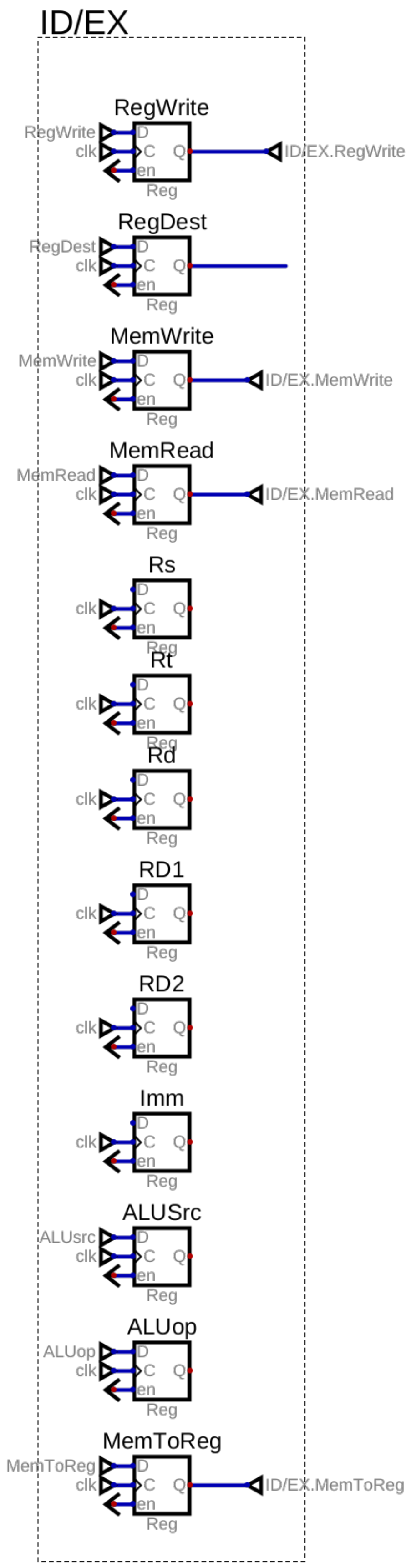

Figure 3: ID/EX Register(s)

4.4 EX/MEM Register(s)

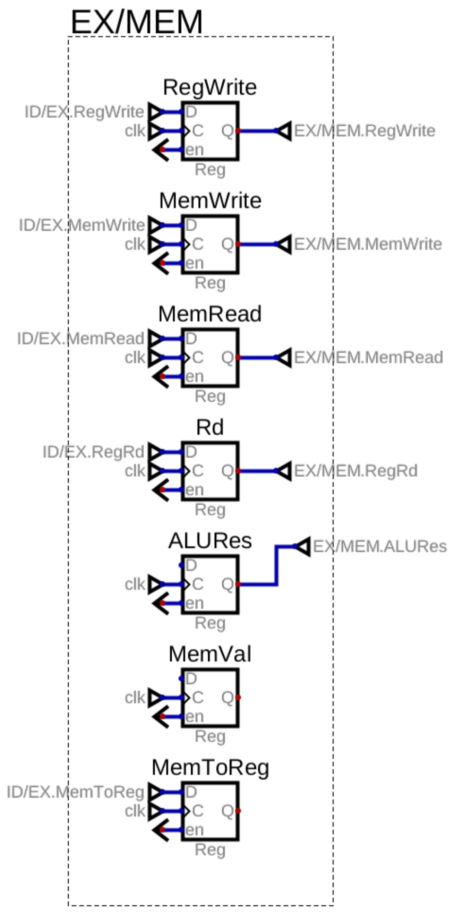

Figure 4: EX/MEM Register(s)

4.5 MEM/WB Register(s)

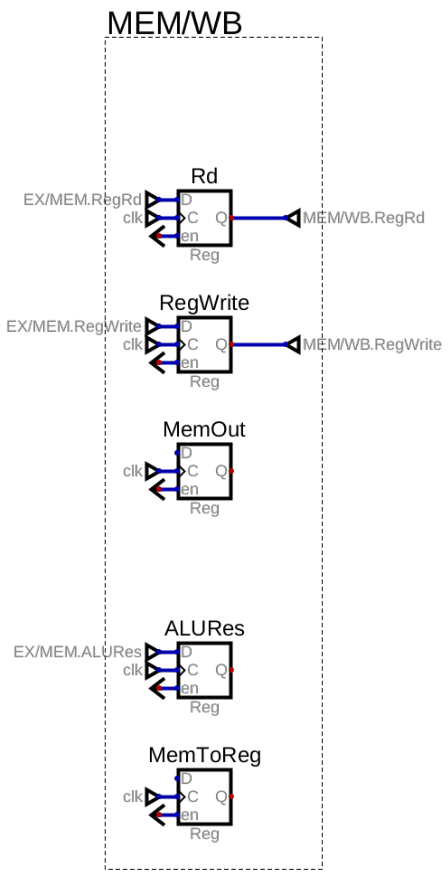

Figure 5: MEM/WB Register(s)

5 Forwarding Unit

5.1 Block Diagram

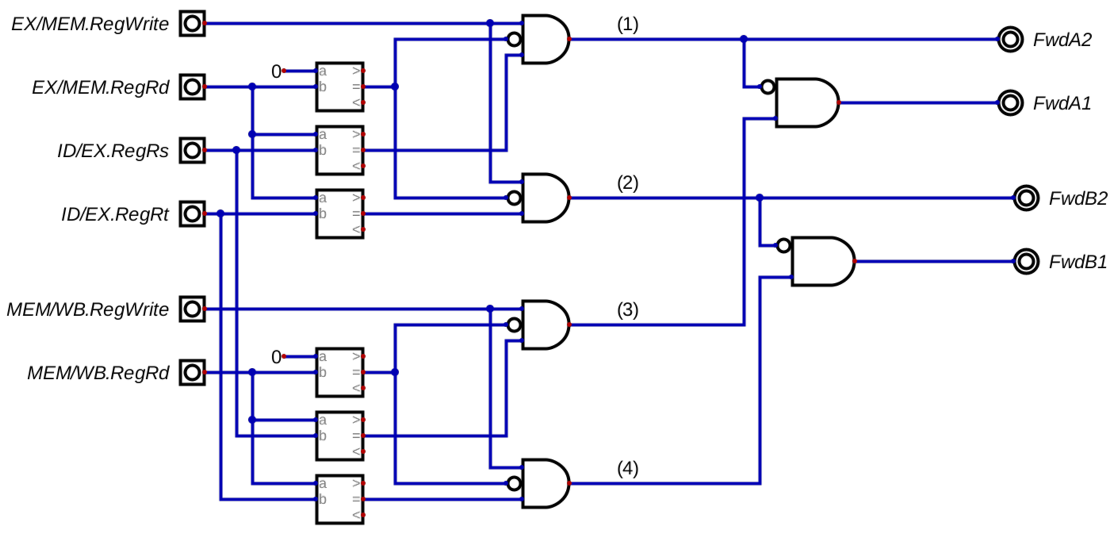

Figure 6: MEM/WB Register(s)

5.2 Mechanism

We are using comparators to check the conditions (1)-(4).

C1	C3	FwdA2	FwdA1	Description
0	0	0	0	No forwarding
0	1	0	1	Forward from MEM state
1	0	1	0	Forward from EX state
1	1	1	0	Forward from EX state

Table 4: Forwarding values for the first source register

C2	C4	FwdB2	FwdB1	Description
0	0	0	0	No forwarding
0	1	0	1	Forward from MEM state
1	0	1	0	Forward from EX state
1	1	1	0	Forward from EX state

Table 5: Forwarding values for the first source register

Hence,

$$
\begin{aligned}
FwdA2 &= C_1 \\
FwdA1 &= \overline{C_1}C_3 \\
FwdB2 &= C_2 \\
FwdB1 &= \overline{C_1}C_4
\end{aligned}
$$

6 Discussion

The focus of our design was to implement it efficiently with pipelining mechanism. Pipelining increases the throughput of the processor by letting several instuctions in different stages occuring simultaneously. However, the latency might not always increase. Although only four R-type instructions were required, our design also supports some immediate and memory operations, for the ease of data loading.

We used built-in RAM and ROMs of our simulator software as the data memory, control unit and instruction memory which made our task easier. We also impelemeted an Assembler in Java to transform the assembly program to machine (hex) code.