CSE 306
Computer Architecture Sessional

Assignment-3: 8-bit MIPS Design, Simulation, and Implementation

Section - A1
Group - 01

Members of the Group:

i 1905001 - Mohammad Sadat Hossain

ii 1905002 - Nafis Tahmid

iii 1905004 - Asif Azad

iv 1905005 - Md. Ashrafur Rahman Khan

v 1905008 - Shattik Islam Rhythm

Contents

1 Introduction

MIPS (Microprocessor without Interlocked Pipeline Stages) is a Reduced Instruction Set Computing (RISC) architecture that has been widely used in the design of microprocessors and embedded systems due to its simplicity, efficiency, and flexibility. In this report, we present the design and implementation of an 8-bit MIPS PC with pipelining, which is a technique used to increase the throughput of a processor by overlapping the execution of multiple instructions.

The pipelining technique is based on the observation that most instructions in a program can be broken down into a series of smaller and simpler operations that can be executed independently and in parallel. By dividing the processor into several stages, each responsible for one of these operations, instructions can be processed simultaneously, resulting in a significant improvement in performance.

The design of our MIPS computer is based on the five-stage pipeline architecture commonly used in modern processors, which consists of the following stages: instruction fetch, instruction decode, execution, memory and write back. Since the last stage does not require any significantly new hardware, it can be merged with the previous stage. Each of these stages is responsible for a specific part of the instruction processing and is connected to the next stage through a pipeline register, which holds the data until it is needed by the next stage.

We have created an 8-bit computer that consists of an 8-bit data bus, an 8-bit address bus, and an 8-bit Arithmetic and Logic Unit (ALU). To conform to standard design principles, we have included separate instruction and data memory in our design.

2 Instruction Set

2.1 Instruction Set Description

Instruction ID	Instruction Type	Instruction
A	Arithmetic	add
B	Arithmetic	addi
C	Arithmetic	sub
D	Arithmetic	subi
E	Logic	and
F	Logic	andi
G	Logic	or
H	Logic	ori
I	Logic	sll
J	Logic	srl
K	Logic	nor
L	Memory	lw
M	Memory	sw
N	Control	beq
O	Control	bneq
P	Control	j

Table 1: Instruction Set Assignment

2.2 Instruction Format

- R-type

Opcode	Src Reg 1	Src Reg 2	Dst Reg
4 bits	4 bits	4 bits	4 bits

- S-type

Opcode	Src Reg 1	Dst Reg	Shamt
4 bits	4 bits	4 bits	4 bits

- I-type

Opcode	Src Reg 1	Src Reg 2/Dst Reg	Addr./Immdt.
4 bits	4 bits	4 bits	4 bits

- J-type

Opcode	Target Jump Address	0
4 bits	8 bits	4 bits

2.3 Instruction Set Assignment

Serial Number	Instruction ID	Opcode
0	K	0000
1	J	0001
2	E	0010
3	I	0011
4	O	0100
5	P	0101
6	F	0110
7	D	0111
8	L	1000
9	M	1001
10	C	1010
11	B	1011
12	G	1100
13	N	1101
14	H	1110
15	A	1111

Table 2: Instruction Set Assignment

3 Circuit Diagram

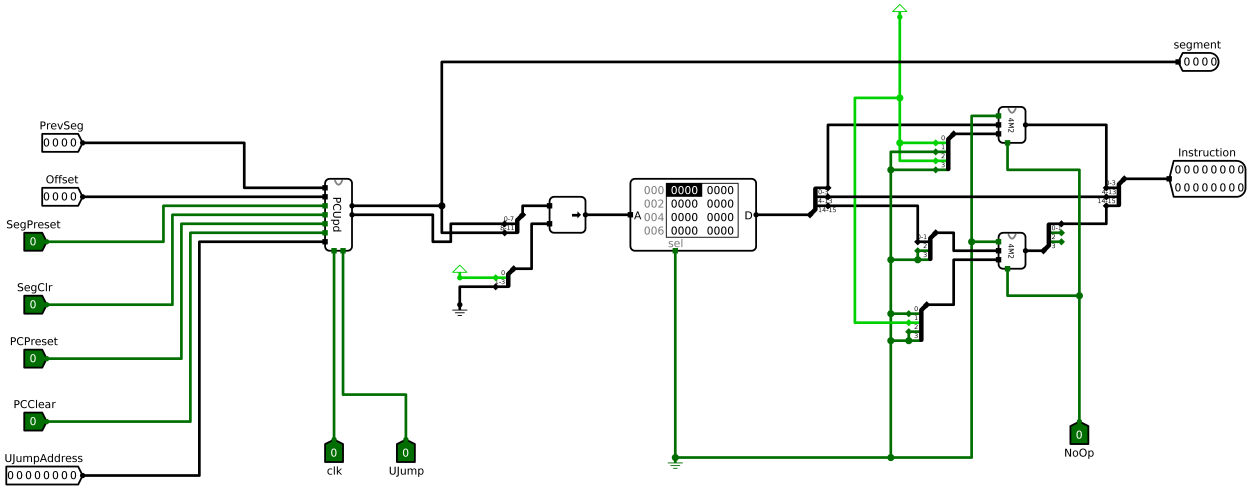

Figure 1: Instruction Fetch

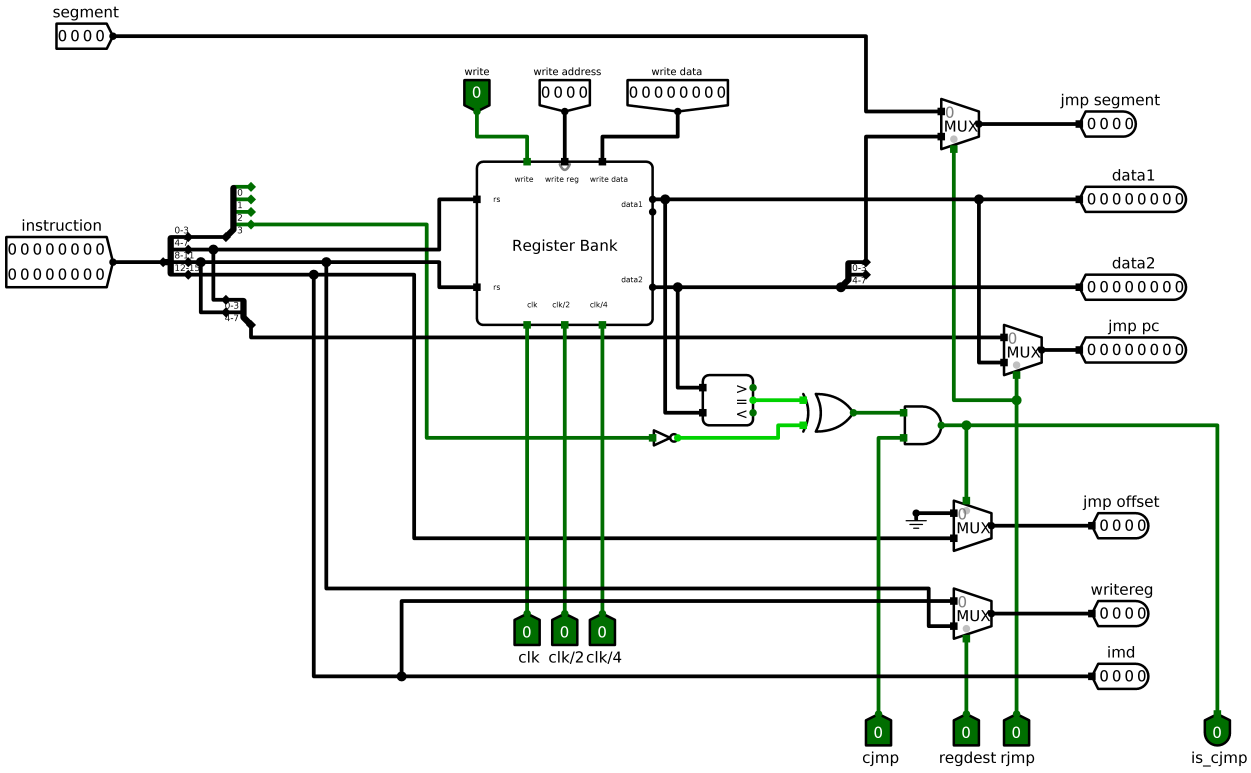

Figure 2: Instruction Decode

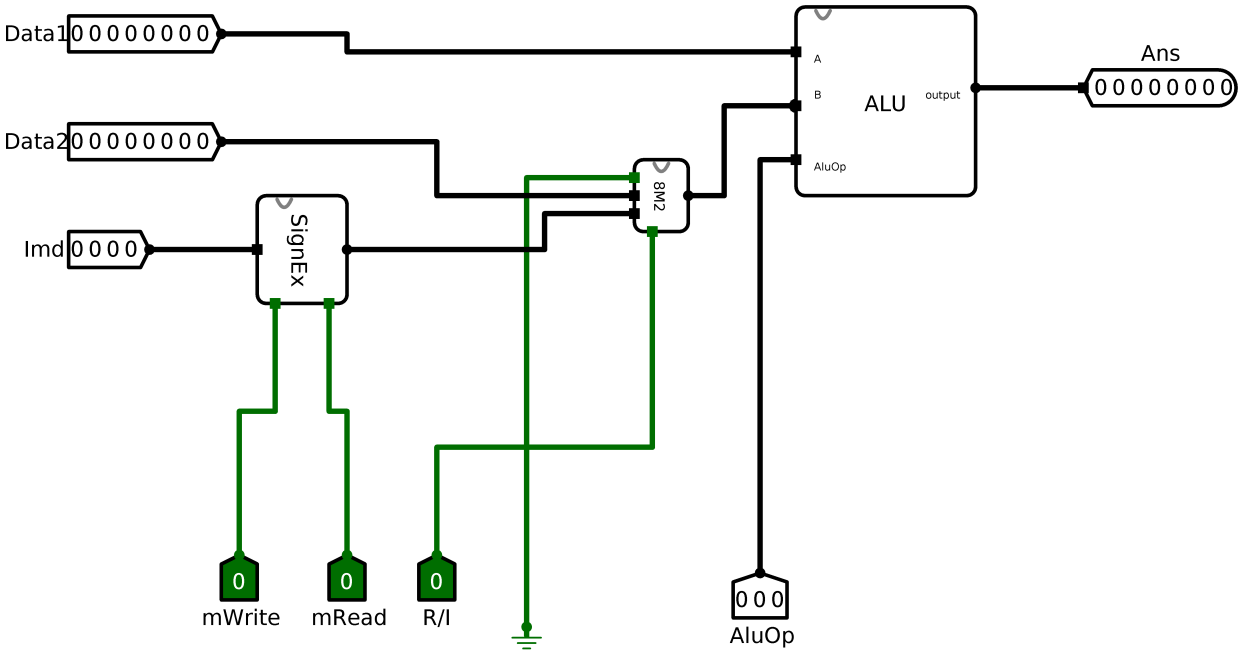

Figure 3: Execution Unit

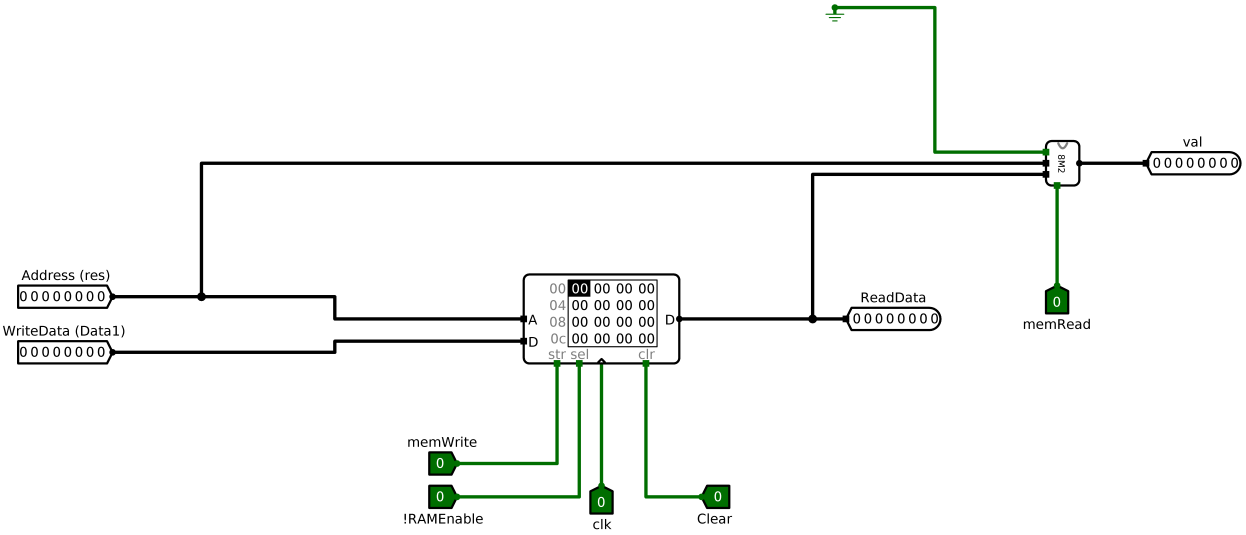

Figure 4: Memory Unit

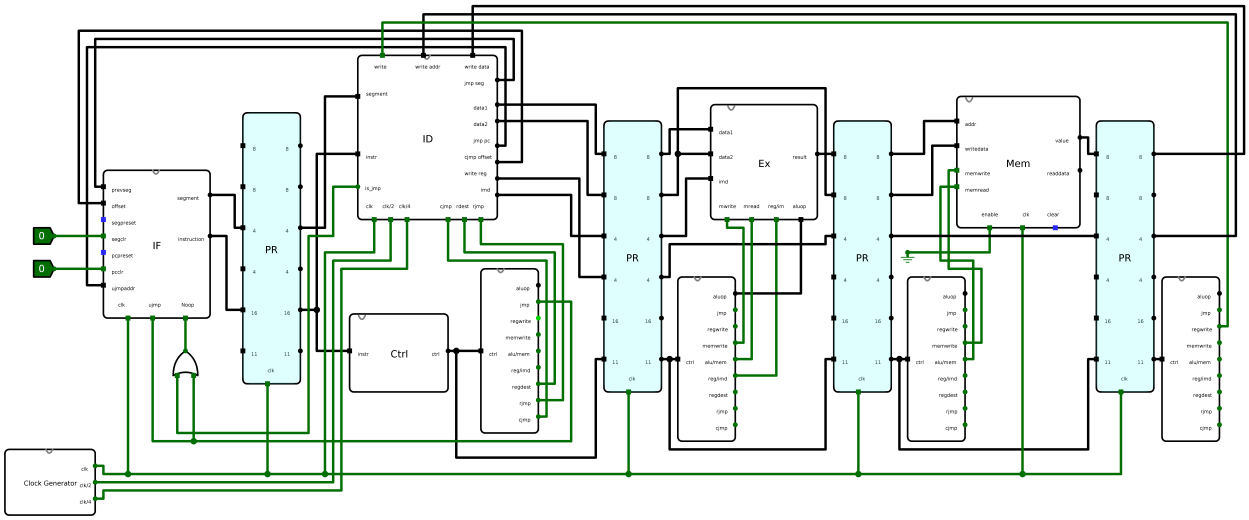

Figure 5: The Complete MIPS

4 How to write and execute a program in this machine

Along with implementing the hardware, we have constructed a basic assembler, which will take a MIPS program and generate machine code instructions in accordance with Instruction Set. This instruction needs to be burnt in the instruction memory (In our implementation, an Atmega32). If there are hazards in the program, the assembler will handle that inserting NoOps. For giving clocks, we have two options - automatic (Using Atmega) and manual (Using Push Button).

5 Special Features Implemented

5.1 Stack Memory

We have kept two special registers SP and BP to help in stack operations. SP will point to the top of the stack if not updated manually. BP can be used to support function calling. AT the beginning of the program execution SP and BP will be FF in hexadecimal.

5.2 8 Bit Data

We were primarily instructed to build a 4-bit MIPS computer. We have extended our MIPS computer and now it has 8-bit data bus and 8-bit ALU. SO, our computer can operate on larger data.

5.3 Pipelining

Pipelining is used in most of the modern architectures. Pipelining improves the performance of a computer significantly. We have implemented a simple version of pipelining in our computer. There are five pipelining stages: Instruction Fetch, Instruction Decode, Execution in ALU, Memory Read or Write, Write Back to Register.

5.4 Segment for PC

Our system has 8 bit program counter. But with 8 bit there can be at most 255 instructions in a program. So, we introduced segment register to extend this limit. Our system includes 4 bit segment along with 8 bit PC, so there are total 12 bits for instruction memory addressing. That means there can be at most 4095 instructions which is much larger

5.5 Special Instructions

We have introduced two special instructions along with the instruction set specified. J-type format of our instruction set has four spare bits. These bits can be utilized to introduce extra instructions. We used those spare bits for our special instructions, so bits needed for opcode remains the same. The special instructions are:

5.5.1 Jump Register

This instruction will read two registers from the register bank. One will be considered as the new PC and the low 4 bits of the other is the new segment. Then the instruction will jump to that segment and PC. This instruction will help to implement functions. To return from a function to its calling routine, we need this jump register instruction. Format of the instruction:

Opcode for j	Src Reg 1	Src Reg 2	4
4 bits	4 bits	4 bits	4 bits

5.5.2 No Operation

We have implemented a simple version of pipelining, which is not hazard free. So, we must stall one or two cycles to avoid data corruption if hazard arises. No operation or NoOp can help to stall the pipeline. This instruction blocks the register bank and data memory to write anything. Format of the instruction:

Opcode for j	Don't Care	Don't Care	8
4 bits	4 bits	4 bits	4 bits

5.6 Support for Function Calling

Our system can support function calling with some help of the assembler. As we have both BP and SP, functions can use them to manipulate local variables. We also have introduced jump register instruction. The assembler needs to save the return address in the stack when a function is called. When the callee function ends execution, it will pop the PC and segment of the return address in two registers. Then, using jump register instruction, the control will be returned to the caller function. Thus, function calling can be supported.

5.7 Control Forwarding

As we have implemented pipelining, we might need to flush 3 stages if branch instruction is taken. This is not efficient. So, we have placed this decision in the ID (Instruction Decode) stage. Data from two source registers are compared immediately after they are read. The result of this comparison is used along with control flags and sent to IF(Instruction Fetch) stage. Also control flags for unconditional jump are directly sent to IF stage. For this, at most one flush is needed. SO, this is an improvement. All kind of jumps (Conditional, Unconditional, Register) are directly materialised immediately after ID stage.

6 ICs used with their count

6.1 Instruction Fetch

IC	Quantity
IC 7483	3
IC 7408	1
IC 7432	1
IC 74157	5
IC 7495	3
Atmega32A	1
Total	14

6.2 Instruction Decode

IC	Quantity
IC 7408	1
IC 7486	1
IC 74157	5
Atmega32A	2
Total	9

6.3 Execution

IC	Quantity
IC 74157	4
Atmega32A	1
Total	5

6.4 Memory

IC	Quantity
IC 74157	2
Atmega32A	1
Total	3

6.5 Pipeline Registers

Number of IC 7495 as Pipeline Registers between different stages

Stage	Quantity
IF-ID	5
ID-EX	8
EX-MEM	6
MEM-RW	4
Total	23

6.6 Total

IC	Quantity
IC 7483	3
IC 7408	2
IC 7432	1
IC 7486	1
IC 74157	16
IC 7495	26
Atmega32A	6
Total	55

7 Discussion

In this lab report, we discuss our implementation of an 8-bit MIPS PC with pipelining, which goes beyond the initial requirements of the project that called for a 4-bit MIPS PC in both hardware and software. While we acknowledge that our implementation exceeds the original scope, we believe that the additional complexity and features add value to our project and provide a more comprehensive understanding of the MIPS architecture and its implementation.

The fact that an 8-bit PC allows for a wider range of memory addresses, which can be useful in more complex programs inspired us to go for it. More wires made the circuit more complex, also debugging was more difficult for this. We also faced the challenge of running out of available pins on a ATmega microcontroller in Instruction Decode Stage. To address this issue, we utilized another ATmega and used USART (Universal Synchronous/Asynchronous Receiver/Transmitter) communication between these two. This allowed us to transfer data and instructions between the two microcontrollers, freeing up additional pins for other functions. The USART communication was implemented using the UART protocol, which provided a reliable and efficient means of transmitting data between the two microcontrollers. This introduction allowed us to overcome the hardware limitations and successfully implement our 8-bit MIPS PC with pipelining.

Pipelining is a widely used technique in modern processors and plays a critical role in improving their performance. We decided to introduce it in our hardware despite it being a daunting task. Implementing pipelining in our MIPS PC project posed several challenges. One of the primary challenges was to properly incorporate the Parallel In Parallel Out (PIPO) registers. We had to rigorously test a significant amount of these registers to obtain enough ones for our implementation. Managing so many of them and ensuring proper data flow in between the stages was indeed a challenge. Pipelining also introduced new complexities in debugging and testing the processor design. However, regardless of these challenges, implementing pipelining was a valuable learning experience that enhanced our knowledge of processor architecture and design.

Overall, we believe that our pipelined 8-bit MIPS PC adds value to our project and provides a more comprehensive understanding of the MIPS architecture and its implementation.