CSE 210

Computer Architecture Sessional

Assignment-3: MIPS Design and Simulation

Section - A2
Group - 02

Members of the Group:

1. 2105045 - Sadia Binte Sayad
2. 2105058 - Elias Mainur
3. 2105060 - Jonayed Mohiuddin

1 Introduction

A processor or processing unit is a digital circuit which performs operations on some external data source, usually memory or some other data stream. The term is frequently used to refer to the Central Processing Unit(CPU) in a system. A central processing unit is the electronic circuitry that executes instructions comprising a computer program. The CPU performs basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions in the program.

Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that orchestrates the fetching (from memory) and execution of instruc-tions by directing the coordinated operations of the ALU, registers and other components.

There are many types of Processor Design Implementation. MIPS(Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer(RISC) instruction set architecture(ISA). The Processor Design implementing MIPS ISA is called MIPS processor.

In this assignment, we have designed an 8-bit processor that implements the MIPS ISA. Each instruction will take 1 clock cycle to be executed. We have designed instruction memory, data memory, register file, ALU, and a control unit of the Processor.

The Processor is composed of the following five components:

1. **Program Counter (PC)**: The program counter (PC) is a 8-bit Register which activates at the falling edge of the clock signal. After every clock it adds 1 to its previous address. The value it stores is used as the Instruction Memory Address.

2. **Register File**: The Register File is a bank of seven Registers. They are denoted as $zero, $t0, $t1, $t2, $t3, $t4, $t5, $sp. The $sp register is the stack pointer, which has an initial value of 0x00. The $zero register stores 0x00. Other Registers are used as General Purpose Registers.

3. **Arithmetic Logic Unit (ALU)**: The ALU performs all the calculations. It is controlled by a 3-bit ALUop code, which sets the type of calculation it performs. It performs calculations with two 8-bit binary numbers.

4. **Control Unit**: The Control Unit decodes the instruction by giving the selection input to all the MUXs, Register File, Data Memory, and ALU.

5. **Data Memory**: The Data Memory stores the Stack values and works as the main memory. It has 256 bytes of storage capacity. It stores data as 8-bit value.

2 Instruction Set

2.1 Instruction Set With Instruction ID

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	sw
M	Memory	lw
N	Control	beq
O	Control	bneq
P	Control	j

2.2 Instruction Set With Op-Code

No	Binary	Type	Instruction	Category
0	0000	Arithmetic	sub	R-Type
1	0001	Logic	srl	R-Type
2	0010	Logic	andi	I-Type
3	0011	Logic	ori	I-Type
4	0100	Logic	nor	R-Type
5	0101	Control	bneq	I-Type
6	0110	Logic	and	R-Type
7	0111	Control	beq	I-Type
8	1000	Memory	sw	I-Type
9	1001	Arithmetic	subi	I-Type
10	1010	Arithmetic	addi	I-Type
11	1011	Logic	sll	R-Type
12	1100	Arithmetic	add	R-Type
13	1101	Memory	lw	I-Type
14	1110	Logic	or	R-Type
15	1111	Control	j	J-Type

3 Complete Block Diagram

This section presents the complete circuit diagram for the MIPS processor, including all relevant components such as the Register File, ALU, Instruction Memory, Data Memory, and Control.

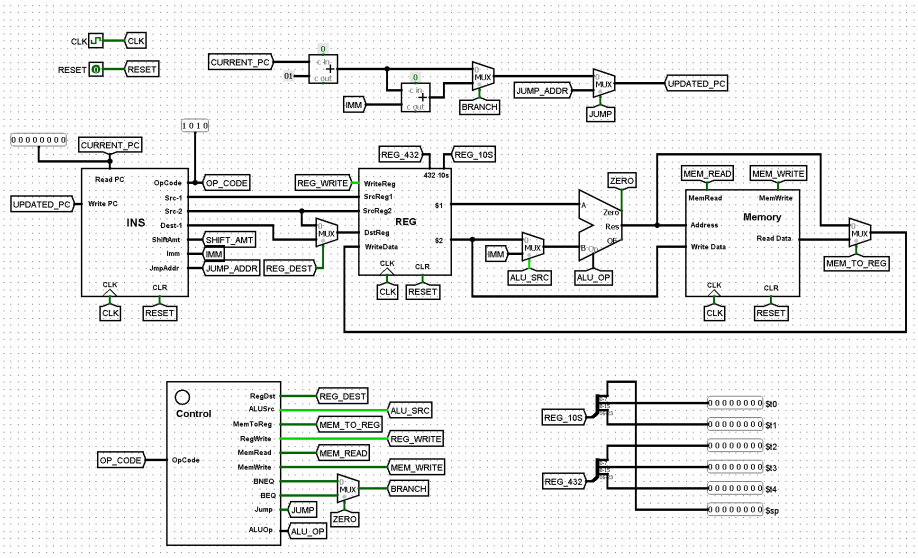

Figure 1: Complete Block diagram of an 8-bit MIPS processor

3.1 All Components

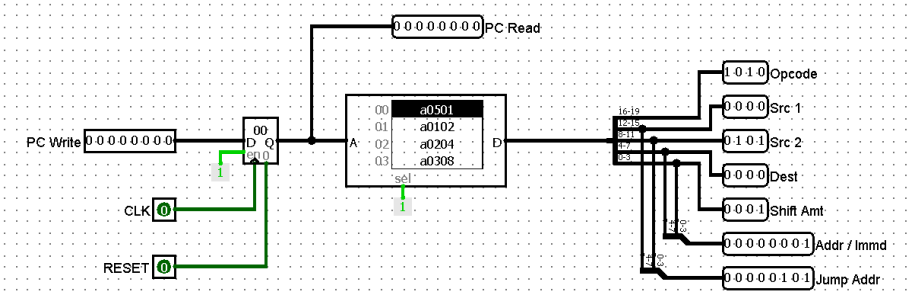

Figure 2: Instruction memory with PC

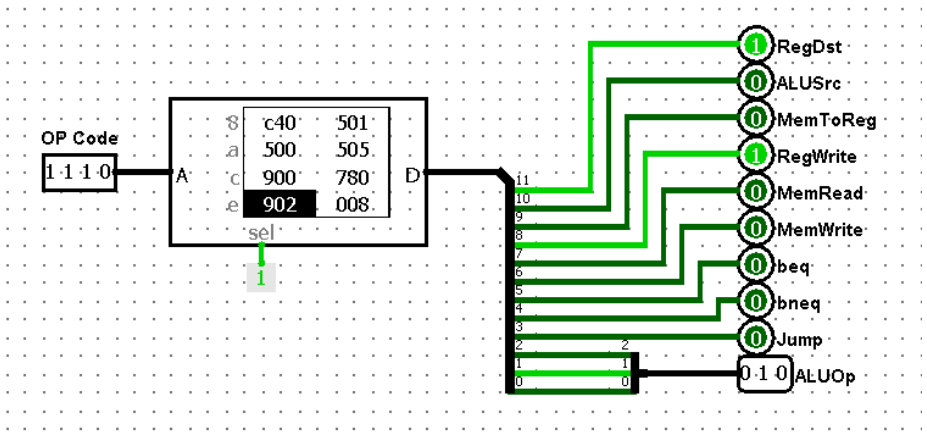

Figure 3: Control Unit

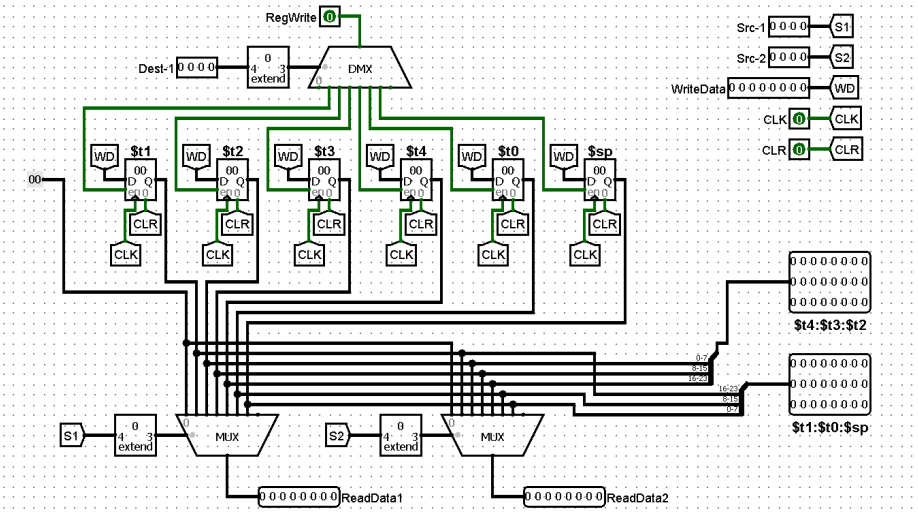

Figure 4: Register File

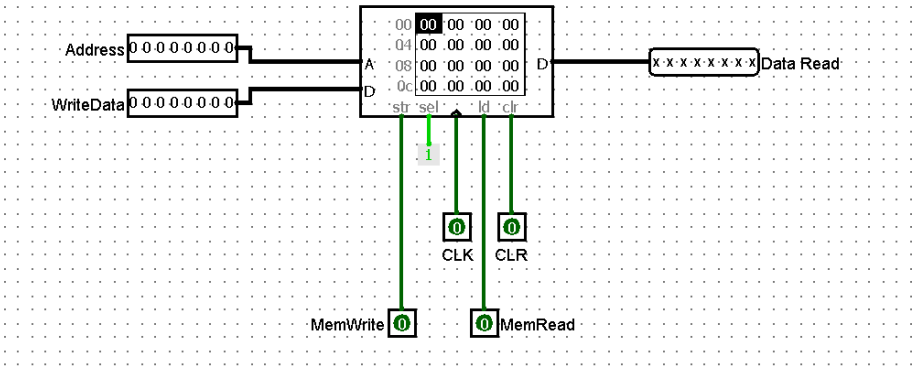

Figure 5: Data Memory with the stack

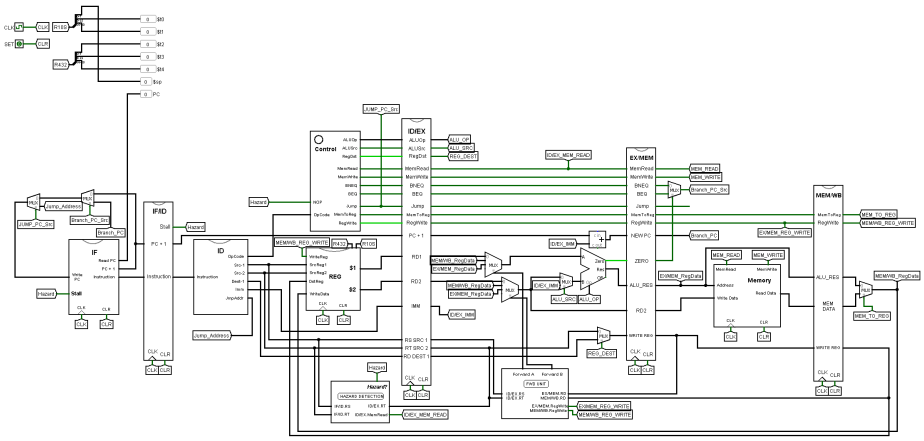

Figure 6: Complete Block diagram of an 8-bit MIPS processor with Pipelining

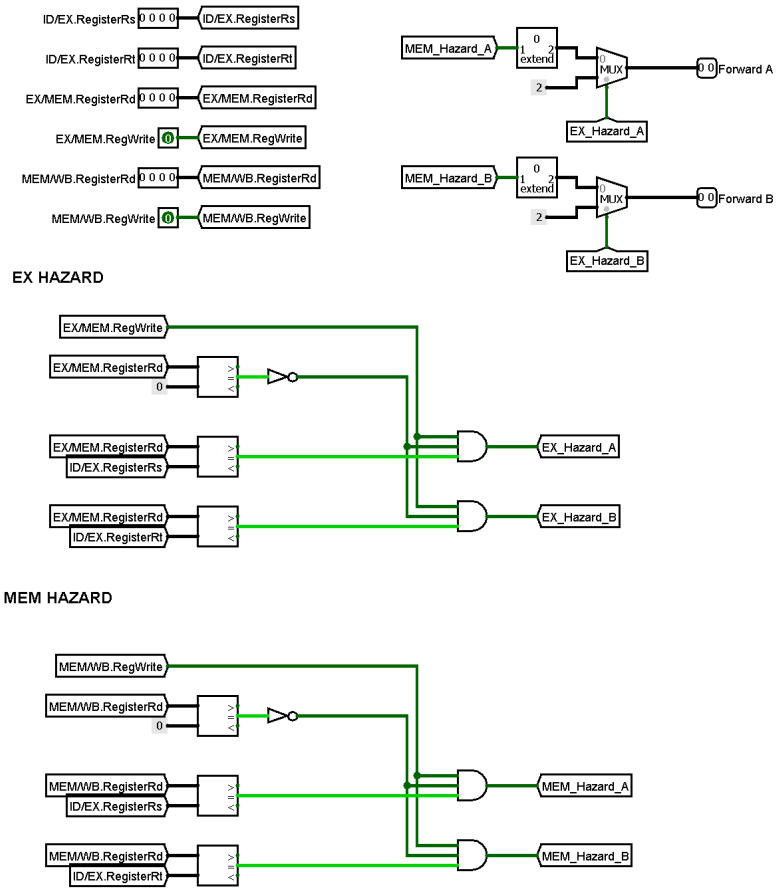

Figure 7: Forwarding Unit

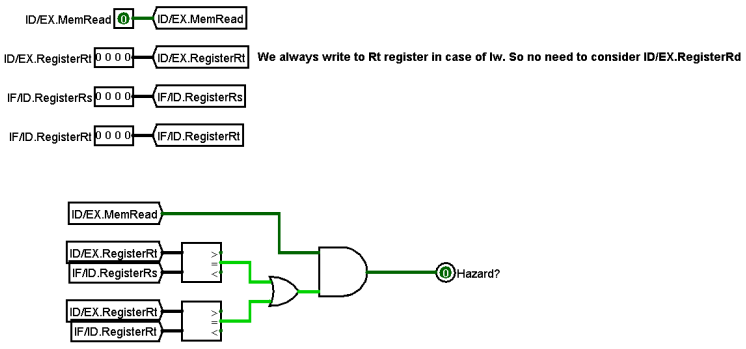

Figure 8: Hazard Detection

4 Steps for Writing and Executing a Program on a MIPS Processor

This section outlines the procedure for writing and running a program on a MIPS processor architecture.

Step 1: Creating the MIPS Assembly Code

The first task in executing a program is writing the MIPS assembly code. This can be done using any standard text editor or Integrated Development Environment (IDE). MIPS assembly language includes instructions that the processor can decode and execute, such as load, store, arithmetic, and control operations.

Step 2: Translating the Code into Machine Language

Once the MIPS code is written, it is processed by an assembler. The assembler converts the high-level MIPS assembly code into machine-readable hexadecimal instructions, which represent the binary encoding of the assembly commands that the processor understands. The output is a series of hexadecimal values corresponding to the instructions.

Step 3: Storing the Instructions in Instruction ROM

After generating the machine code, it is loaded into the Instruction ROM (Read-Only Memory), which stores the instructions for execution. The processor retrieves these instructions one by one during the program's execution.

Step 4: Configuring the Control Signals

Along with the instruction storage, a separate control file holds the necessary control signals. These signals manage the operations of the processor's components, including the ALU (Arithmetic Logic Unit), the Register File, and the Data Memory. Typically stored in a text file, the control file is loaded into the processor to guide the control unit in generating the appropriate signals for each instruction.

Step 5: Executing the Program via Clock Pulses

The program executes step-by-step in sync with clock pulses. Each pulse prompts the processor to fetch the next instruction from the Instruction ROM. After fetching an instruction, the processor decodes it and generates the corresponding control signals, based on the control file. These signals direct the processor on how to perform the operation, whether it involves arithmetic computation through the ALU or data access from memory.

Once the instruction is executed, the processor writes the results to the registers or memory as needed. This cycle repeats with each clock pulse, progressively advancing the program through its instructions.

5 ICs

Components	Count
MUX	8
DEMUX	1
ADDER	2
Register	7
ROM	2
RAM	1

Contribution

- 2105046 : Control Unit and Data Memory
- 2105058 : Register file and Instruction Memory
- 2105060 : Assembler and Merging

6 Discussion

In this assignment, the given instruction set architecture (ISA) was simple yet comprehensive, covering arithmetic, logical, memory access, and control flow operations. The decision to use a microprogrammed control unit simplifies the design process and makes it easier to debug and modify the control logic. Implementing a Harvard architecture (separate buses for instruction and data memory) have provided significant benefits in terms of speed and parallelism compared to a shared bus design. In designing the 8-bit MIPS processor, we faced some problems but learned a lot along the way. One major issue was with setting the control signals properly in the control memory; they were't working properly at first, leading to errors in program execution. After some debugging, we determined what was wrong and fixed it. A critical part of the project was writing a program to convert MIPS assembly code to machine code. Data dependencies between consecutive instructions in out pipelined version can introduce stalls. So we implemented a simple data forwarding mechanism that reduce delays without significantly increasing hardware complexity. The Stack was implemented from the top of Data Memory (`0xFF`). We added an extra command to set `$sp` to `0xFF`. We also tackled the shifting operations, specifically the SLL and SRL instructions in the R-type format instead of using a extra control flag for them. The corresponding MIPS-like code was parsed using the C++ assembler. The assember generated the Machine Language code corresponding to our given specifications. This was loaded onto the Instruction Memory and after that enabling tick executed the program successfully.