CSE 306

Computer Architecture Sessional

Assignment 3: 8-bit MIPS Design and Simulation

Submitted By:

Group No: 05

Sub-Section: A2

Group Members:

1. Kawshik Kumar Paul (ID: 1705043)
2. Iftekhar Hakim Kaowsar (ID: 1705045)
3. Rasman Mubtasim Swargo (ID: 1705051)
4. Apurba Saha (ID: 1705056)
5. Maisha Rahman (ID: 1705060)

Level: 3   **Term:** 1

Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology (BUET)

Date of submission: 19 June 2021

Problem Specification :
In this assignment, we had to design an 8-bit processor that implements the MIPS instruction set. Each instruction we designed takes 1 clock cycle to be executed. The length of the clock cycle is long enough to execute the longest instruction in the MIPS instruction set. The main components of the processor are as follows: instruction memory, data memory, register file, ALU and a control unit.

Introduction:
MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems.
We will construct the datapath and control unit of MIPS Instruction set:
■ The memory-reference instructions load word (lw) and store word (sw).
■ The arithmetic-logical instructions add, sub, addi, subi, AND, OR, NOR, ANDI, ORI, sll and srl.
■ The instructions branch equal (beq), branch not equal (bneq) and jump (j), which we add last.

Instruction Set:
Our MIPS instruction is 20 bits long and we have 3 formats.

- **R Type**

Opcode	Src Reg1	Src Reg2	Dest Reg	Shift Amount
4 bits	4 bits	4 bits	4 bits	4 bits

- **I Type**

Opcode	Src Reg	Dest Reg	Address / Immediate
4 bits	4 bits	4 bits	8 bits

- **J Type**

Opcode	Target Jump Address	0	0
4 bits	8 bits	4 bits	4 bits

Control ROM Map as value of 8 bits consists of Function(3), R_Enable(1) Immediate/DataGot2_selector(1), Memory_enable(1), Destination_selector(1), Write_Value_Selector(1).

Instruction ID	MIPS Opcode(binary)	Control ROM Map (hex)	Operation	Type	Category
K	0000	90	NOR	R	Logic
I	0001	b0	sll	R	Logic
L	0010	04	sw	I	Memory
H	0011	7A	ORI	I	Logic
F	0100	5A	andi	I	Logic
D	0101	3A	subi	I	Arithmetic
P	0110	00	j	J	Control - unconditional
N	0111	00	beq	I	Control - conditional
B	1000	1A	addi	I	Arithmetic
M	1001	13	lw	I	Memory
J	1010	d0	srl	R	Logic
G	1011	70	OR	R	Logic
E	1100	50	AND	R	Logic
A	1101	10	add	R	Arithmetic
C	1110	30	sub	R	Arithmetic
O	1111	00	bneq	I	Control - conditional

Block Diagram:

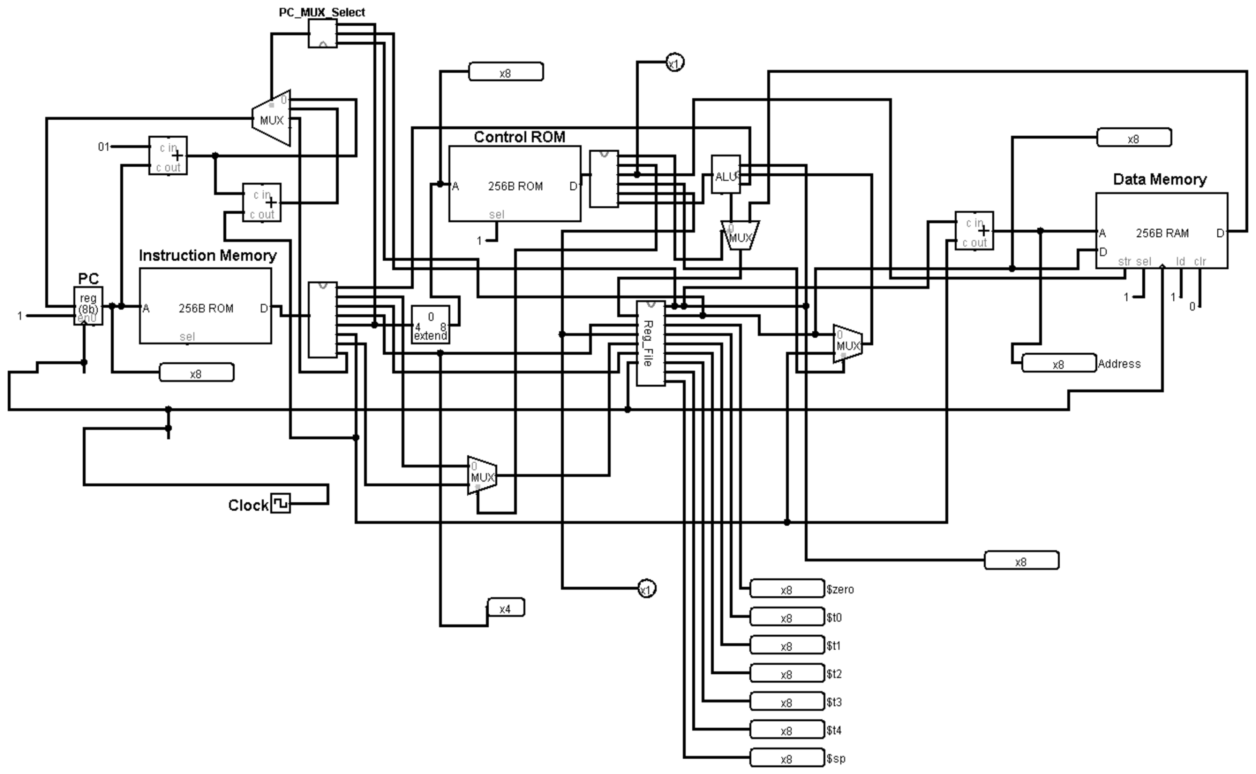


Figure 1: Block Diagram

This is a block diagram of our MIPS Processor, which works in a single cycle clock pulse, passing opcode from the instruction memory to the control unit. Then the ALU is operated as instructed and store / write data to the specific register, memory or stack.

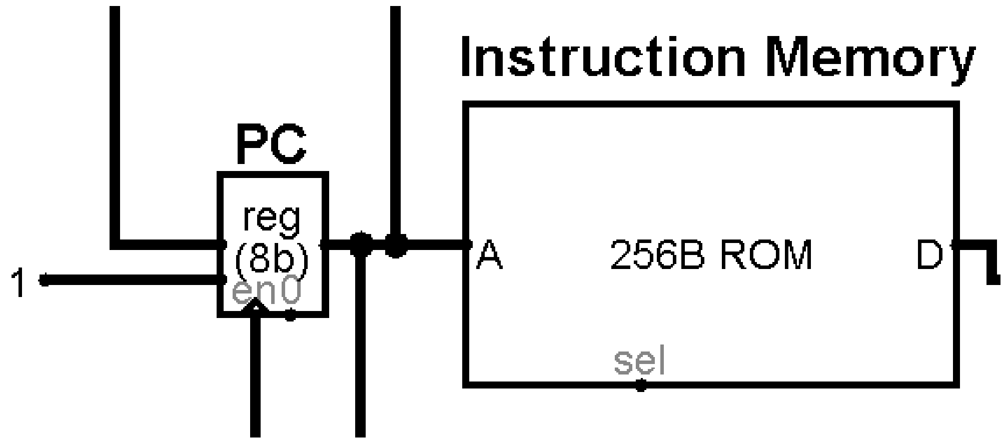


Figure 2: Instruction Memory with PC

After every clock pulse, the PC is increased by 1 and MIPS instructions are loaded from Instruction Memory and pass the instructions to the control unit and register file.

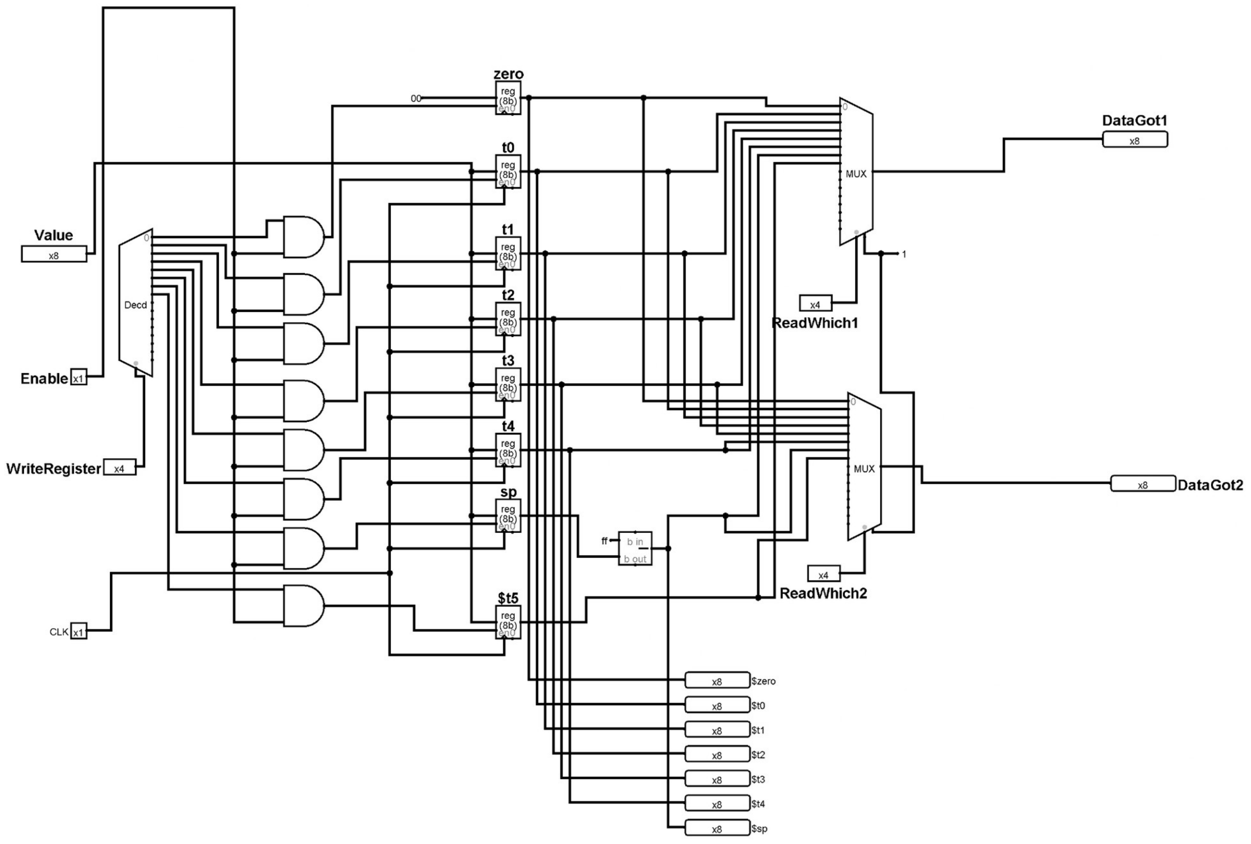


Figure 3: Register File

Register files consist of all the registers in which the register values are stored, sometimes it is read and sometimes is written depending on the instructions.

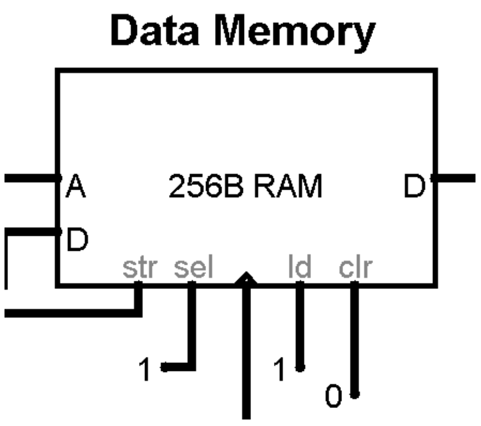
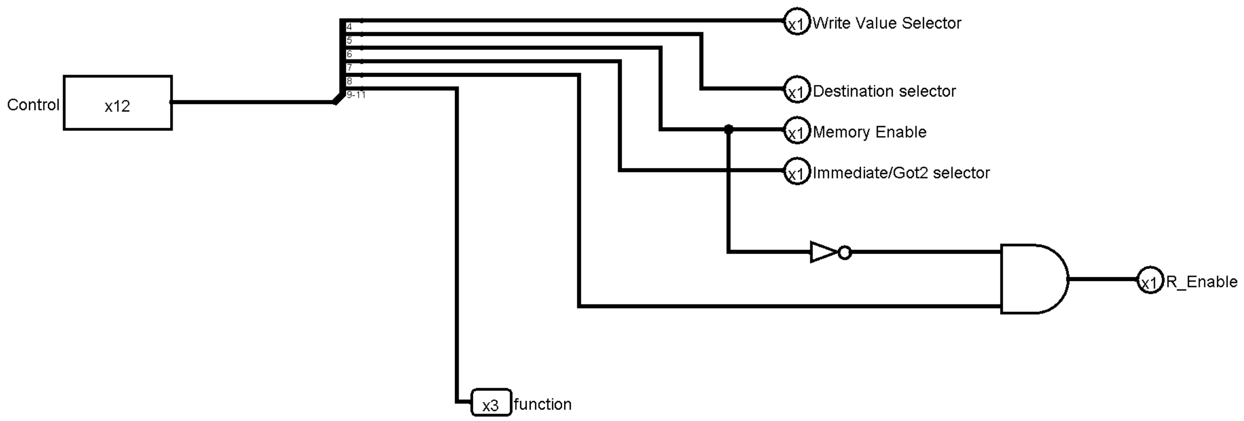


Figure 4: Data Memory

Data memory stores the data and holds the stack from last.



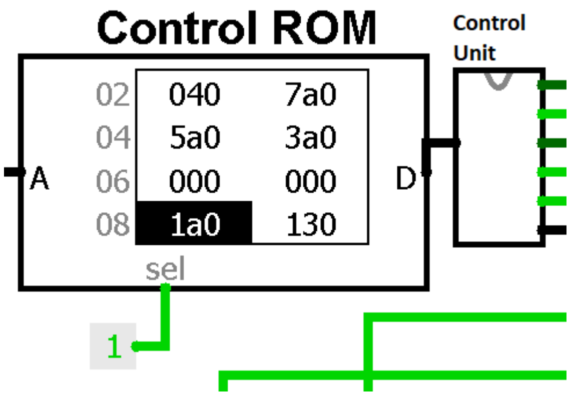


Figure 5: Control Unit

Control unit controls the ALU and other components to do operations depending on the MIPS instructions.

<u>Approach to implement the push and pop instructions:</u>

We had taken a register sp(stack pointer) to store the position of stack-pointer from the topmost address(0xFF). Internally, it is stored as stack size and shown after subtracting from 0xFF. We could not do it directly because we can only initialize the flip-flop's value to zero. Every type of push and pop operations has been converted to a set of memory and addition operations. So, a complete push or pop operation does not take place in a single clock pulse.

1. When a number is pushed we save the number into that address and increase the registers' value by one(which basically subtracts the address from the top).
2. If a value from an array is pushed, an intermediate register($t5) is used.
3. In case of popping, the number stored in the address is loaded and the value of stack-pointer is subtracted by 1(address is increased).

It is notable that we used an extra register($t5) here to make it easier and our register file allows it.

Example of the three cases:

1. **push $t0**

```
sw $t0, 0($sp)
addi $t5,$zero, 0xFF
subi $t5,$t5,$sp
addi $sp,$t5,1
```

2. **push 3($t0)**

```
lw $t5, 3($t0)
sw $t5, 0($sp)
addi $t5,$zero, 0xFF
subi $t5,$t5,$sp
addi $sp,$t5,1
```

3. **pop $t0**

```
lw $t0, 0($sp)
addi $t5,$zero, 0xFF
subi $t5,$t5,$sp
subi $sp,$t5,1
```

IC used with count:

IC	Count
7408 (AND)	4
7404 (NOT)	1
7432 (OR)	1
7480 (FULL ADDER)	7
8 Bit Shifter	2
74151 (8 input multiplexer)	3
4 x 8 ROM	1
8 x 20 ROM	1
8 x 8 RAM	1
74157 (Quad 2-to-1 multiplexer)	9
74154 (4X16 decoder)	1
74173 (4 bit D type register)	18

Simulator used along with the version number:

Logisim 2.7.1

Discussion:

In this assignment, we designed an 8-bit processor that implements the MIPS instruction set. For this purpose, we used basic gates (AND, OR, NOT, NOR), universal gates (XOR, XNOR), some other necessary gates (MUX, Shifter, Bit Finder, Adder, Subtractor, Bit Extender), RAM, ROM and register. The Processor takes a 20 bit binary number (Instruction) and the circuit reads/stores the register values and memory values. We used the known technique of designing the processor for MIPS instruction set. Here all types of instructions are executed in a single clock pulse other than pop and push commands. Moreover, Control ROM made our mapping quite easier, otherwise it would have been tedious by introducing combinational logic or elsehow. We designed an assembler in CPP and loaded the produced code in another ROM (Instruction Memory). We could not use the single memory (i.e. using Data Memory as Instruction Memory) because we can not include 2 different addresses (or 2 different data read at a time) in a single memory unit while working in a single clock cycle method.

We followed the provided circuit diagram, and implemented the circuit. While designing, we put emphasis on simplifying the circuit. We asserted our design by testing various inputs and matching the corresponding outputs. However, testing the outputs, we successfully finished our simulation.