CSE 306 Assignment 3
A1
Group 5

Md. Zarif Ul Alam : 1705010
Jamilus Sheium : 1705012
Naeem Ahmed : 1705014
Md. Shadmim Hasan Sifat : 1705021
Solaiman Ahmed : 1705022

June 18, 2021

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 instructions 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 five components:

1. Program Counter : 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 : Register File is a bank of seven Registers. They are denoted as $zero, $t0, $t1, $t2, $t3, $t4, $t5, $sp. $sp register is the stack pointer which has initial value ffH. $zero register stores 00H. Other Registers are used as General Purpose Registers.

3. ALU : The ALU does all the calculations. It is controlled by 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 main memory. It has 256 bytes storage capacity. It stores Data as 8-bit value.

2 Instruction Set

2.1 Instruction Set With Instruction ID

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

2.2 Instruction Set With Op-Code

Op-Code	Category	Type	Instruction
0000	Memory	I	sw
0001	Control-Conditional	I	beq
0010	Logic	R	nor
0011	Logic	R	sll
0100	Logic	I	andi
0101	Logic	I	ori
0110	Control-Unconditional	J	j
0111	Arithmetic	I	addi
1000	Logic	R	and
1001	Arithmetic	R	sub
1010	Control-Conditional	I	bneq
1011	Logic	R	or
1100	Logic	R	srl
1101	Memory	I	lw
1110	Arithmetic	R	add
1111	Arithmetic	I	subi

3 Complete Block Diagram

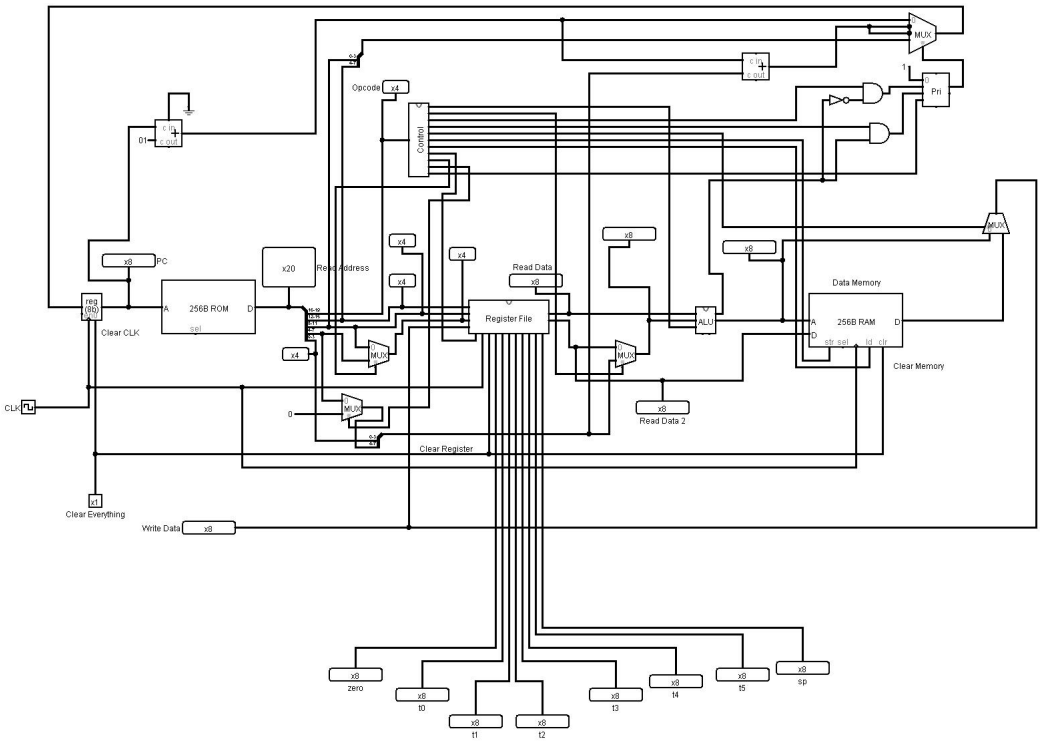

Figure 1: Complete Block diagram of an 8-bit MIPS processor

4 Block Diagrams of The Main Components

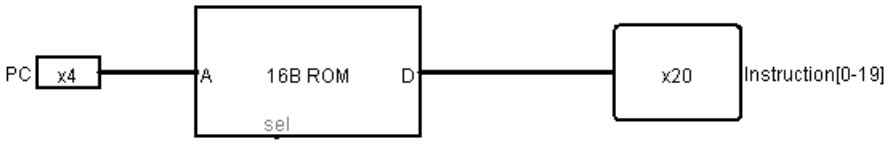

Figure 2: Instruction memory with PC

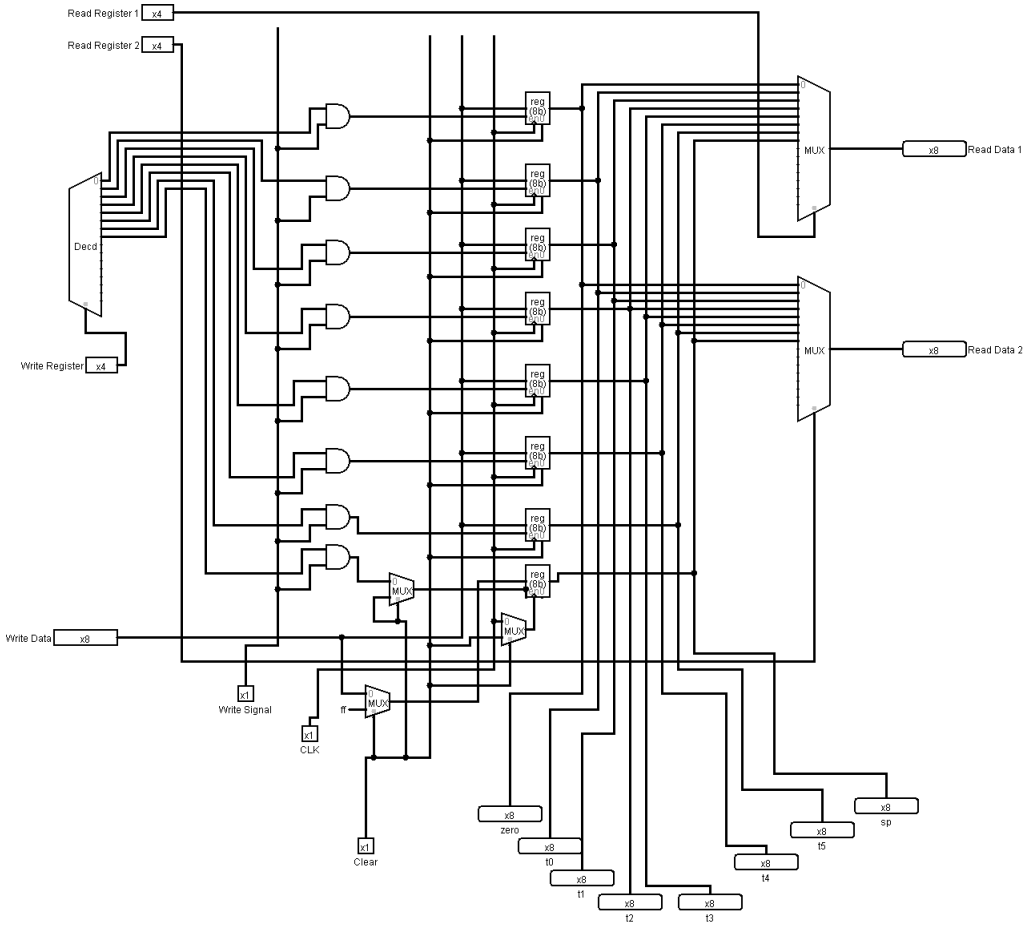

Figure 3: Register file

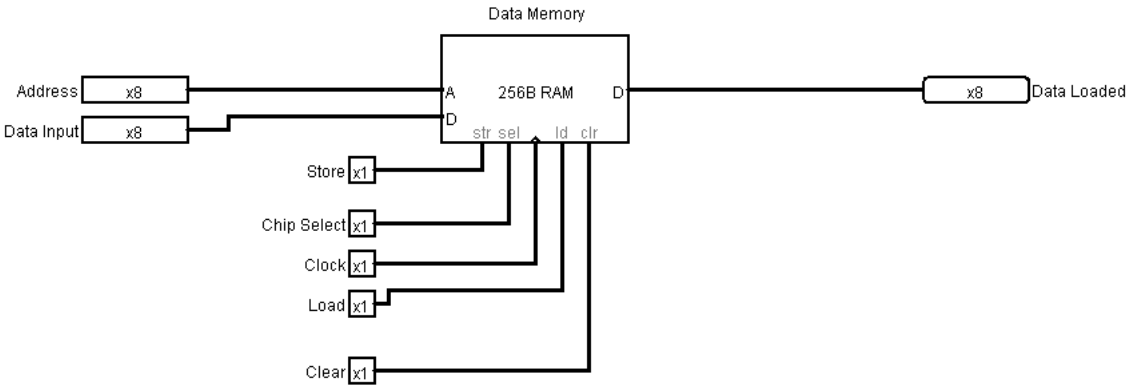

Figure 4: Data memory with the stack

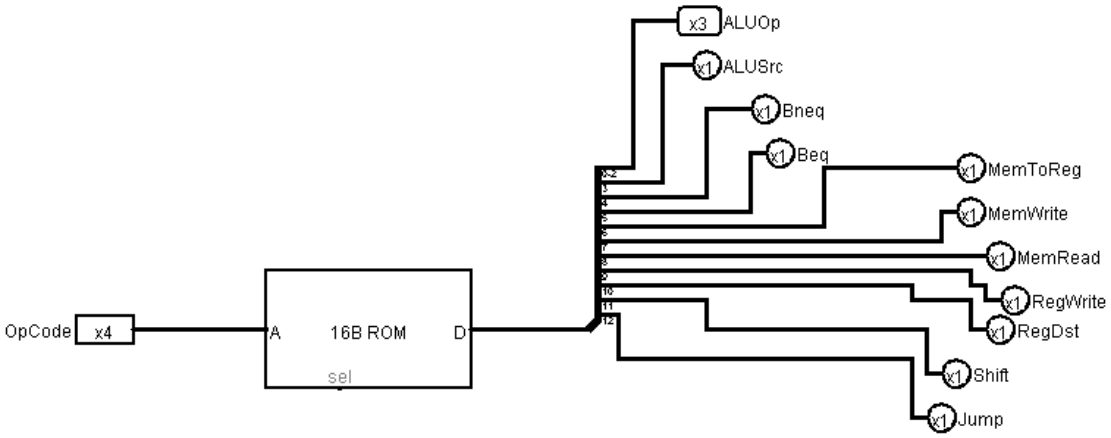

Figure 5: Control unit

5 Approach To Implement The Push and Pop Instructions

For implementing this part, we introduced two extra registers :

- $t5 (as temporary register)
- $sp (as stack pointer register) which will be initialized with 0xFF.

For the first instruction in this part:

Instruction	Description
push $t0	mem[$sp] = $t0

Equivalent Mips Code we generated :

```
subi    $sp,$sp,1
sw      $t0,0($sp)
```

For the second instruction in this part:

Instruction	Description
push offset($t0)	mem[$sp] = mem[$t0 + offset]

Equivalent Mips Code we generated :

```
lw      $t5,offset($t0)
subi    $sp,$sp,1
sw      $t5, 0($sp)
```

For the third instruction in this part:

Instruction3	Description3
pop $t0	$t0 = mem[$sp]

Equivalent Mips Code we generated :

```
lw      0($sp)
addi    $sp,$sp,1
```

6 ICs used with count as a chart

Gate	Gate Count	IC	IC Count
MUX(4-to-1)	1	74153	1
MUX(2-to-1)	7	74157	2
MUX(8-to-1)	1	74151	1
MUX(16-to-1)	2	74150	2
Adder(8-bit)	3	7483	6
Decoder(4-to-16)	1	74154	1
AND(2-bit)	10	7408	3
AND(8-bit)	1	7430	1
NOT(1-bit)	1	7404	1
NOR(8-bit)	3	4078	3
Priority Encoder(2-bit)	1	Priority Encoder(2-bit)	1
Register(8-bit)	9	Register(8-bit)	9
ROM(256X20)	1	ROM(256X20)	1
ROM(256X8)	1	ROM(256X8)	1
ROM(16X13)	1	ROM(16X13)	1
Subtractor(8-bit)	1	Subtractor(8-bit)	1
RAM(256X8)	1	RAM(256X8)	1
Clock	1	Clock	1
Right-Shifter(8-bit)	1	Right-Shifter(8-bit)	1
Left-Shifter(8-bit)	1	Left-Shifter(8-bit)	1

7 Simulator

Logisim Version 2.7.1

8 Discussion

While implementing the circuit, we had to change our design several times.Some designs required a lot of ICs. To optimize number of ICs in our design, we had to discard those. Again, we invested enough of our time in cross-checking corner cases for each sample test case cautiously.Considering all these aspects, we finally implemented the most optimized design we could find.