Assignment on 4-bit MIPS Design, Simulation, and Implementation

1805006

Tanjeem Ajwad Zaman

1805016

Ahmed Hossain

1805019

MD Rownok Zahan Ratul

1805027

Nafis Karim

1805030

Md Toki Tahmid

August 2022

1 Introduction

This experiment focused on implementing a 4-bit microprocessor using both software and hardware. We utilized a 4-bit ALU and a 4-bit Data Bus. The address bus was 8 bits wide. We included 7 registers (6 mandatory - $zero, $t0, $t1, $t2, $t3, $t4 and 1 bonus register - $sp for stack implementation). The control unit was microprogrammed: each operation had a corresponding control signal stored in a ROM. All instructions needed 1 clock cycle to be executed (and bonus push and pop pseudo-instructions needed 2-3 cycles). Thus the main components included a Program Counter, an Instruction Memory, a Register File, an ALU and a Data Memory. The data bus and address bus included MUXes, adders aside from wiring. Thus a reduced and modified version of the MIPS Instruction Set was implemented.

2 Given Instruction Set

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

Figure 1: **Provided Instruction Set with Instruction Id's and Instruction Type.**

3 Assigned Instruction Serial

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

Table 1: Given Instruction Serial for Group 3

4 Control Unit Input and Outputs

	E	**K**	**I**	**M**	**N**	**H**	**P**	**G**	**O**	**F**	**L**	**A**	**J**	**D**	**B**	**C**
ALU2	0	1	1	0	0	0	x	0	0	0	0	0	1	0	0	0
ALU1	1	1	0	0	0	1	x	1	0	1	0	0	0	0	0	0
ALU0	0	0	0	0	1	1	x	1	1	0	0	0	1	1	0	1
RegDst	1	1	0	x	x	0	x	1	x	0	0	1	0	0	0	1
Beq	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
MemRead	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
MemWrite	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
MemtoReg	0	0	0	x	x	0	x	0	x	0	1	0	0	0	0	0
AluSrc	0	0	1	1	0	1	x	0	0	1	1	0	1	1	1	0
RegWrite	1	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1
Jmp	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
Bne	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Table 2: Control Unit Input Code and Corresponding Control Signal Outputs

5 Opcode For ALU

ALU Opcode	Action/Instruction
0000	Addition
0001	Subtraction
0010	And
0011	Or
0100	Shift Left
0101	Shift Right
0110	Nor
0111	Not Used

Table 3: ALU Opcode and Respective Action to Perform

6 Circuit Diagram

Complete circuit diagram of the 4 Bit MIPS is provided in the given subsections. It consists of 5 major modules:

1. Arithmetic Logic Unit (ALU)
2. Control Unit
3. Register File
4. Instruction Memory
5. Data Memory

6.1 Complete Circuit Diagram

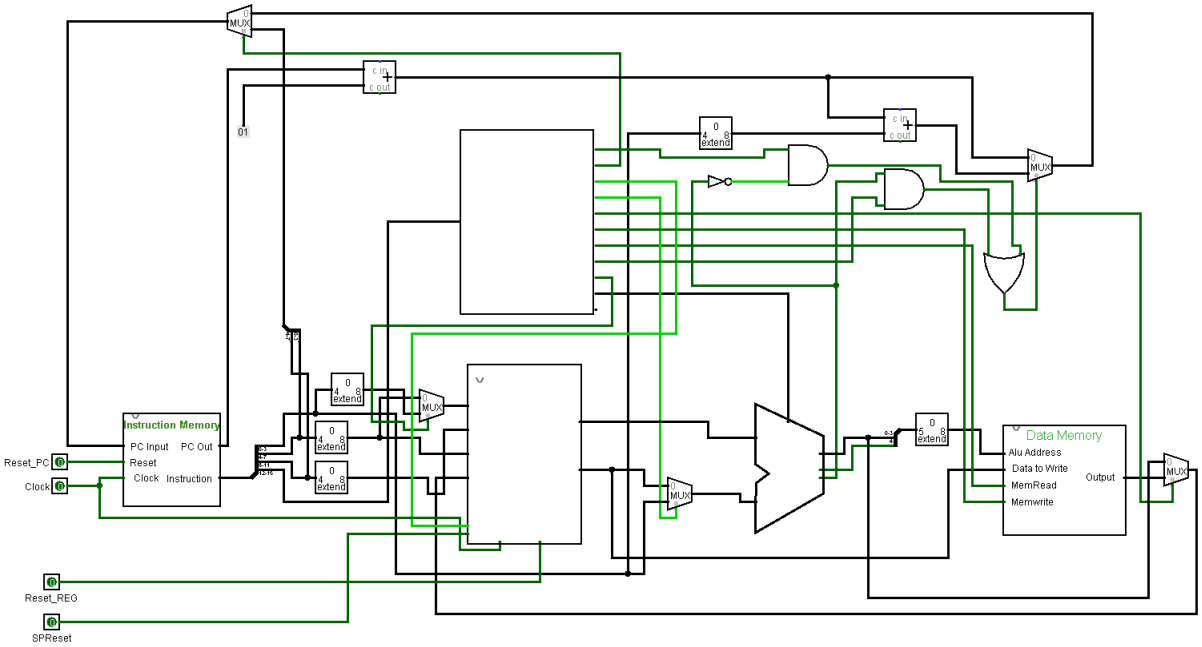

Figure 2: **Complete Circuit Diagram with All Additional Components**

6.2 Arithmetic Logic Unit (ALU)

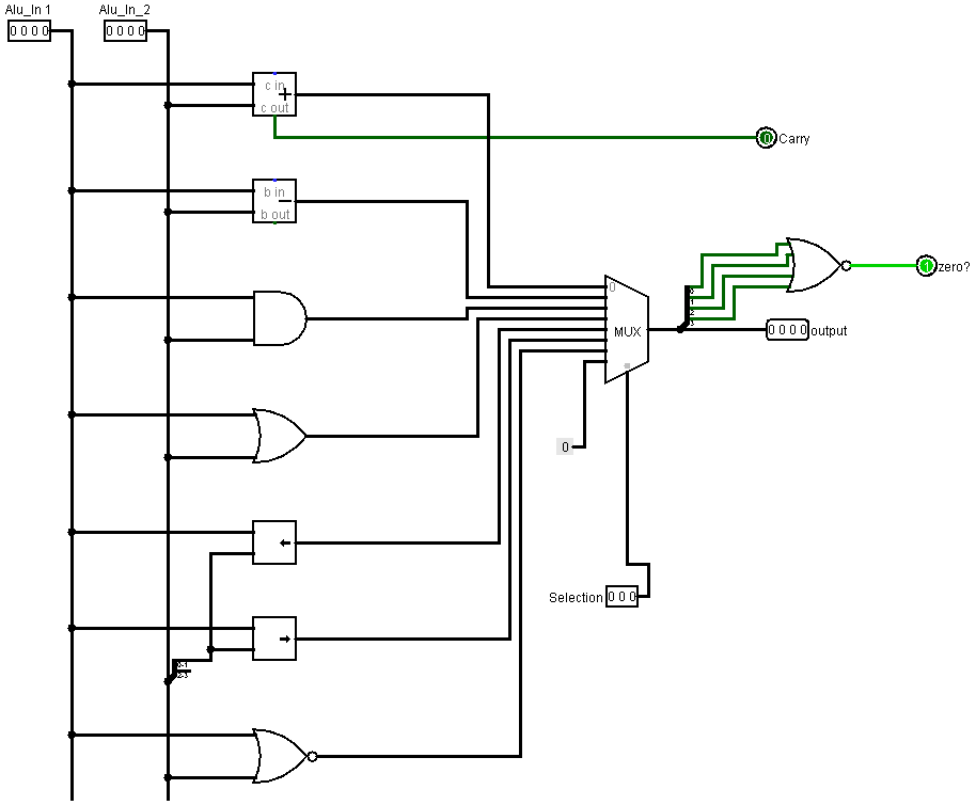

Figure 3: **Arithmetic Logic Unit(ALU)**

6.3 Control Unit

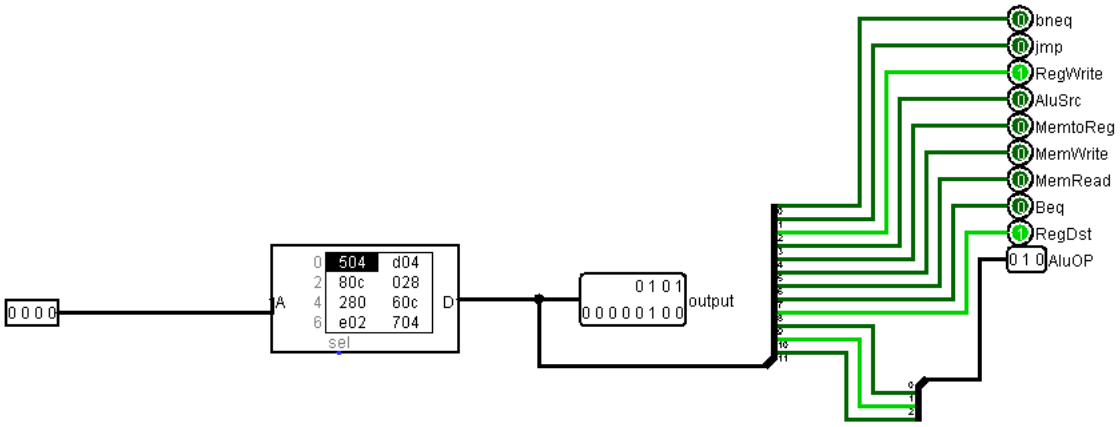

Figure 4: **Control Unit for Providing Appropriate Control Signals on Given Opcode Input.**

6.4 Register File

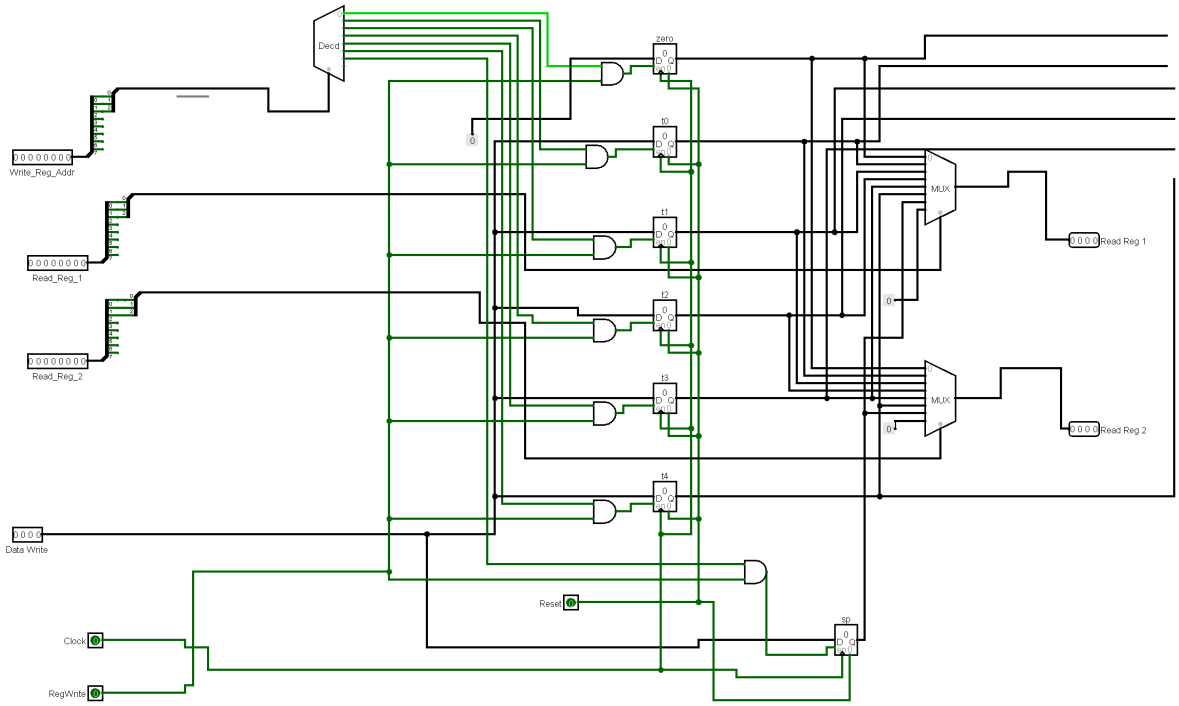

Figure 5: **Register File Has 7 Registers($Zero,$t0,$t1,$t2,$t3,$t4,$t5, and $sp) with 4 Data bits Output. The $sp Register Is Used for Handling Push and Pop Instructions.**

6.5 Instruction Memory

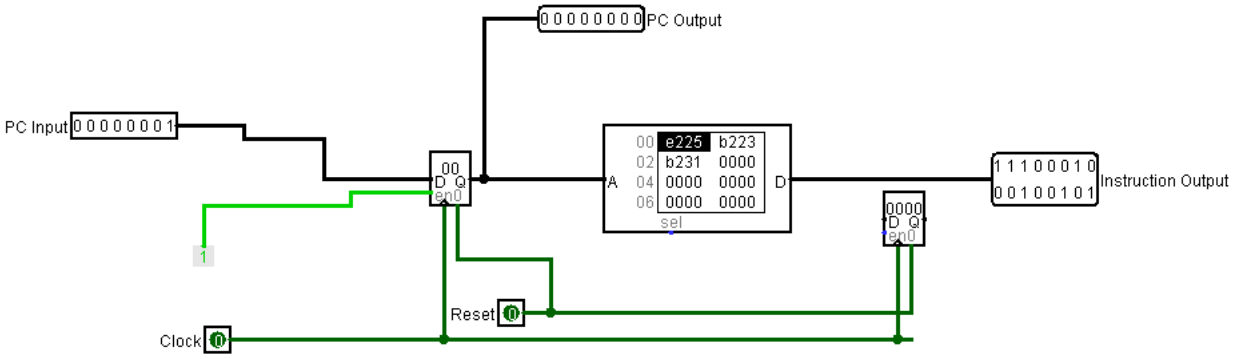

Figure 6: **The Instruction Memory is Used to Load the Whole program into a ROM. The ROM has Address Bus of 8 Bits and Data Output Bus of 16 bits. Output of This Unit is a 16 bit Instuction According to the Given Instruction Format.**

6.6 Data Memory

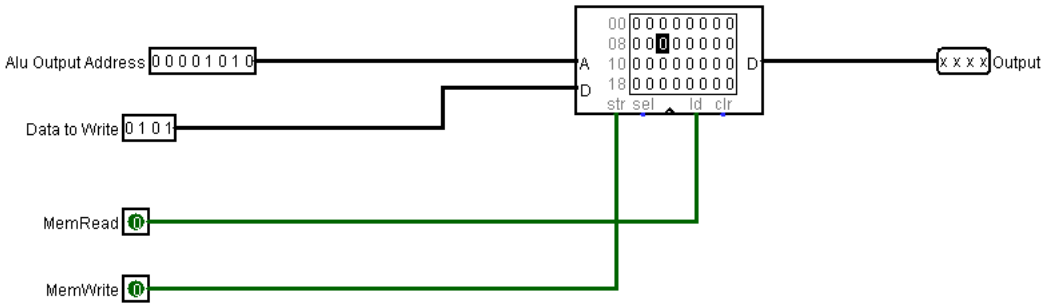

Figure 7: **Data Memory Consists of a RAM That Serves to Load and Store Data into It. An Input Address Along with Data to be Written is Provided into the Data Memory. In Case of Load, a 4 Bit Data is Given as Output from the Data Memory**

7 Gates Count

Gate Name	Count
And	10
Or	2
Not	2
Nor	2

Table 4: Used Logic Gates and Count

8 Discussion

- An instruction set similar to a miniature MIPS Instruction Set was implemented using a 4 bit microprocessor.
- Logisim was used for the software implementation.
- The hardware implementation used Atmega32 ICs for abstraction; ROMs (used in Instruction Memory, Control Unit), RAM (used in Data Memory), Register File, ALU.
- Seven registers (6 mandatory $zero, $t0, $t1, $t2, $t3, $t4 and one optional stack pointer $sp) were implemented
- The corresponding MIPS-like code was parsed using a C++ program. The program generated the Machine Language code corresponding to our given specifications. This was loaded onto the IMs.
- **The PC register was operated on "rising-edge" and the Register File, Data Memory was operated on "falling edge". Otherwise, some specific instruction sequences would create issues of unwanted writes.**
- The Stack was implemented from the top of Data Memory (0xFF). We added an extra command to set $sp to 0xFF. This was done before starting operation.
- Push and Pop can be done by using the $sp pointer accordingly.

  $$push \quad \$t0$$

  was broken into 2 instructions :

  $$sw \quad \$t0, 0(\$sp)$$

$$subi \quad \$sp, \$sp, 1$$

similarly we expanded

$$pop \quad \$t0$$

into 2 instructions :

$$addi \quad \$sp, \$sp, 1$$

$$lw \quad \$t0, 0(\$sp)$$

- Bit extensions were used for all addresses into Register file, since instructions had 4 bit addresses, but the address bus was 8 bits.