North South University

Course Code: CSE332L
Section: 09
Project Report

Name: Designing a Datapath for N-bit Processor

Submitted By:

MD. Ibrahim Siddik

ID: 2211632042

Date: 04/12/2024

Objectives: The primary aim of this project is to design a Datapath for a 19-bit processor that supports various range of instructions, including ADD, ADDI, SUB, INC, SHIFTLEFT, SHIFTRIGHT, XOR, JUMP, BRANCH, LOAD, and STORE. Also, upon completing this project, we will get an idea about the R-I-J instructions format of the MIPS structure.

Instruction Format:

For the bit assigned to me which was 19 bits, the instruction format is given below:

R-Type: 3 bits (opcode) | 4 bits (rs) | 4 bits (rt) | 4 bits (rd) | 4 bits (shamt)

I-Type: 3 bits (opcode) | 4 bits (rs) | 4 bits (rd) | 8 bits (immediate)

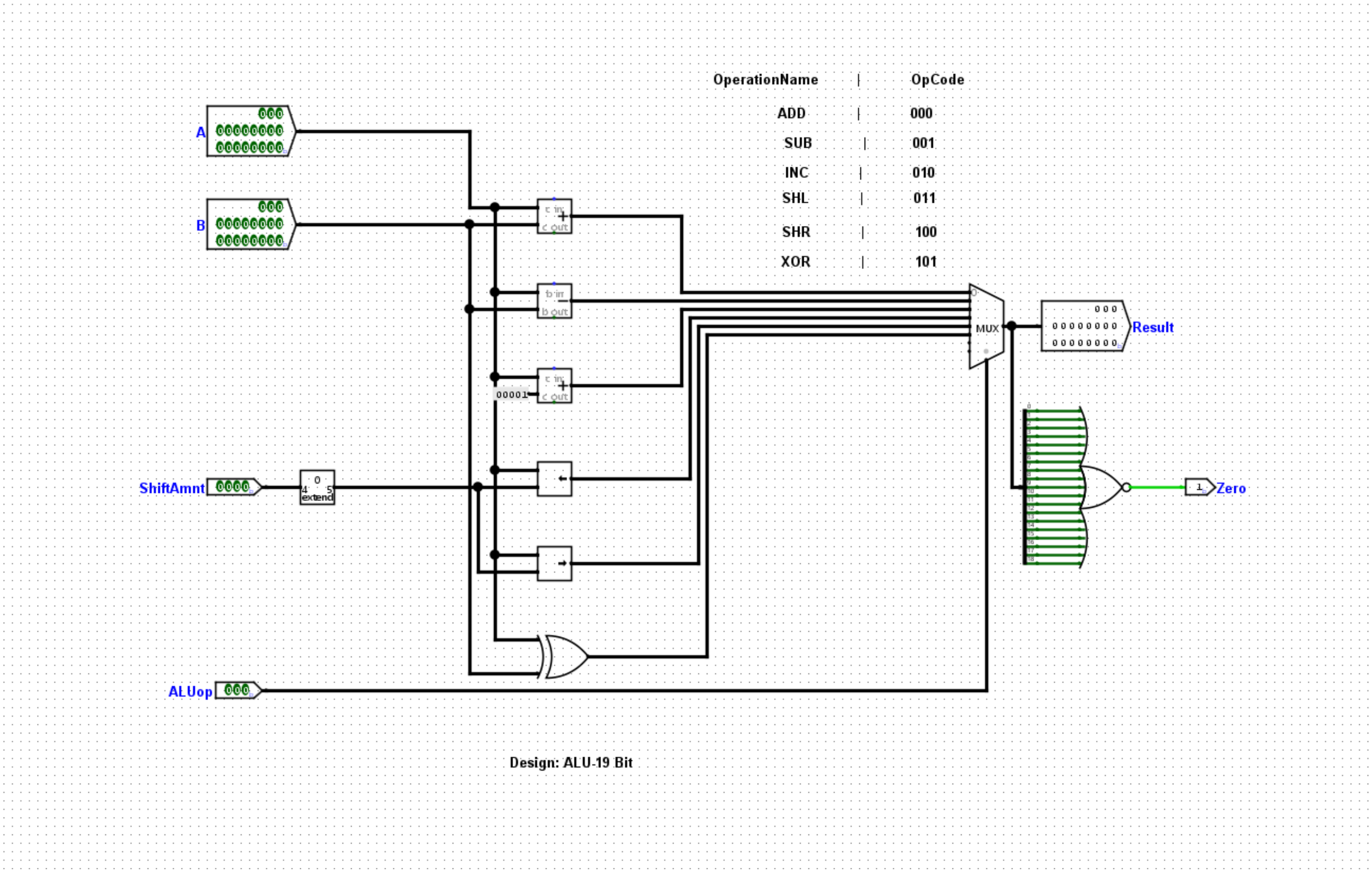
J-Type: 3 bits (opcode) | 16 bits (address)

ALU Design:

Table of operations:

Operation Name	Opcode
ADD	000
SUB	001
INC	010
SHL	011
SHR	100
XOR	101

Screenshot:



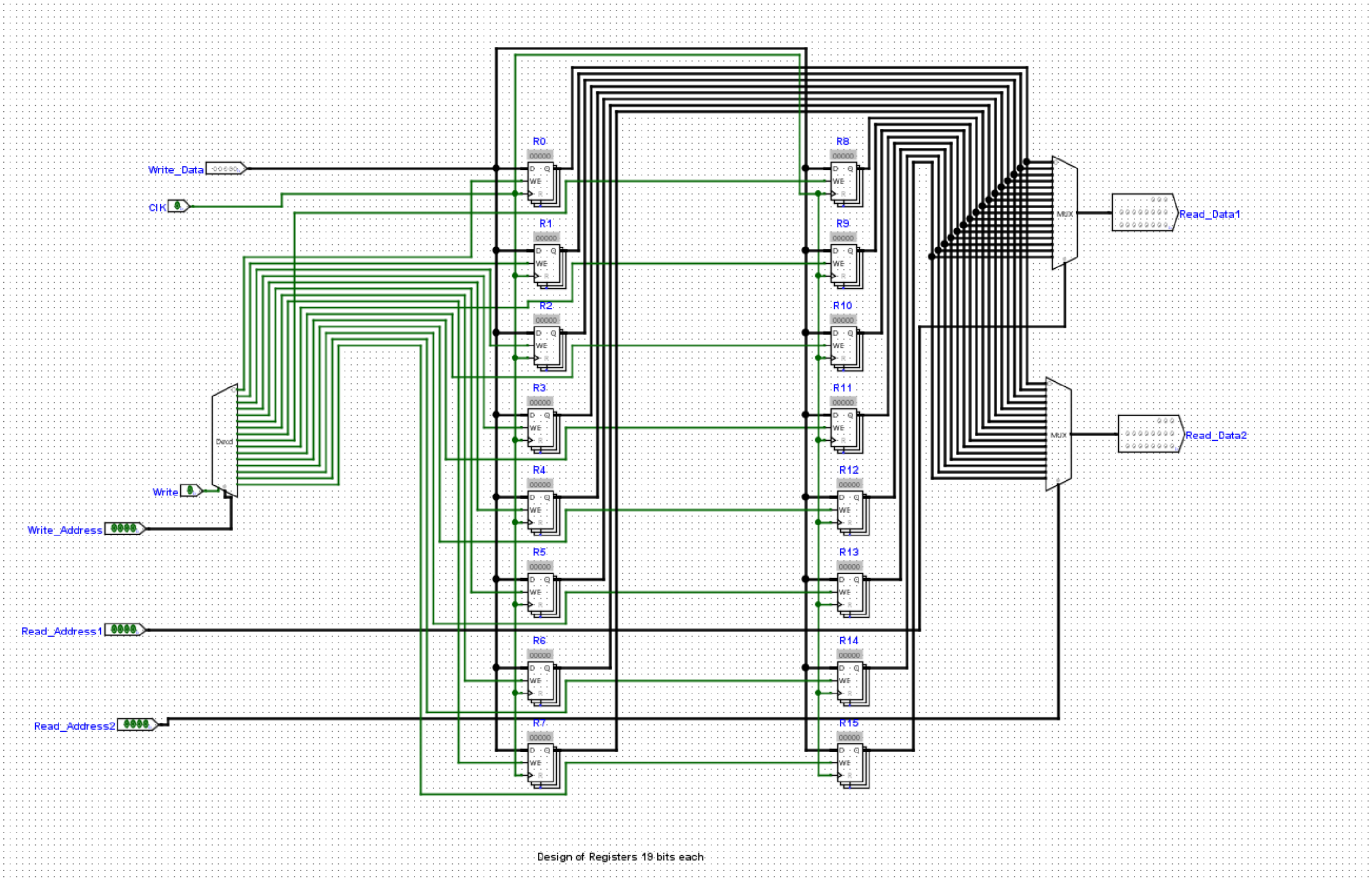
OperationName	OpCode
ADD	000
SUB	001
INC	010
SHL	011
SHR	100
XOR	101

Design: ALU-19 Bit

Register File:

16 registers were used to build the register file to load and store operations for the entire Datapath. Every register's data-bit width was 19 bits.

Screenshot:



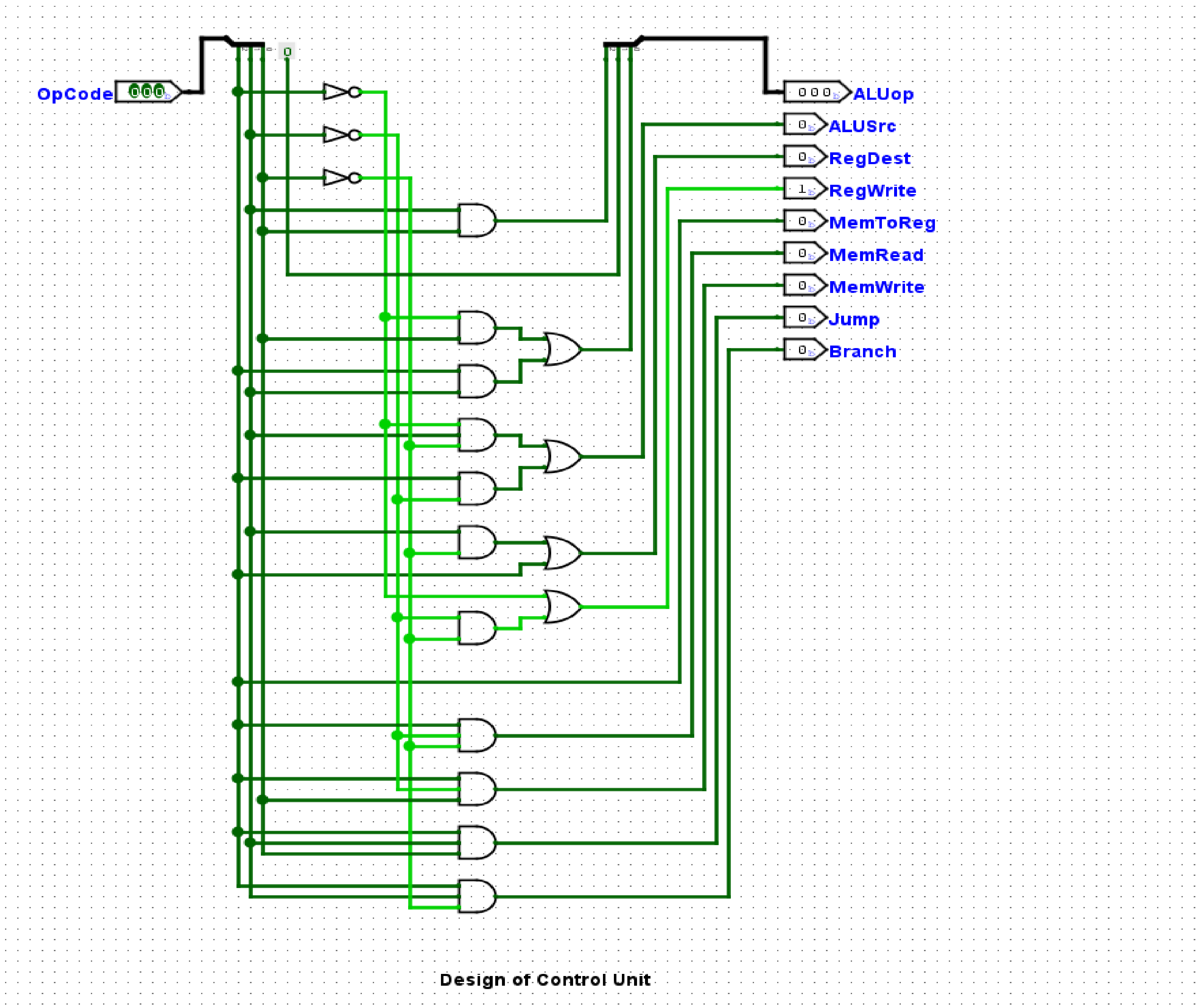
Design of Registers 19 bits each

Control Unit:

Table:

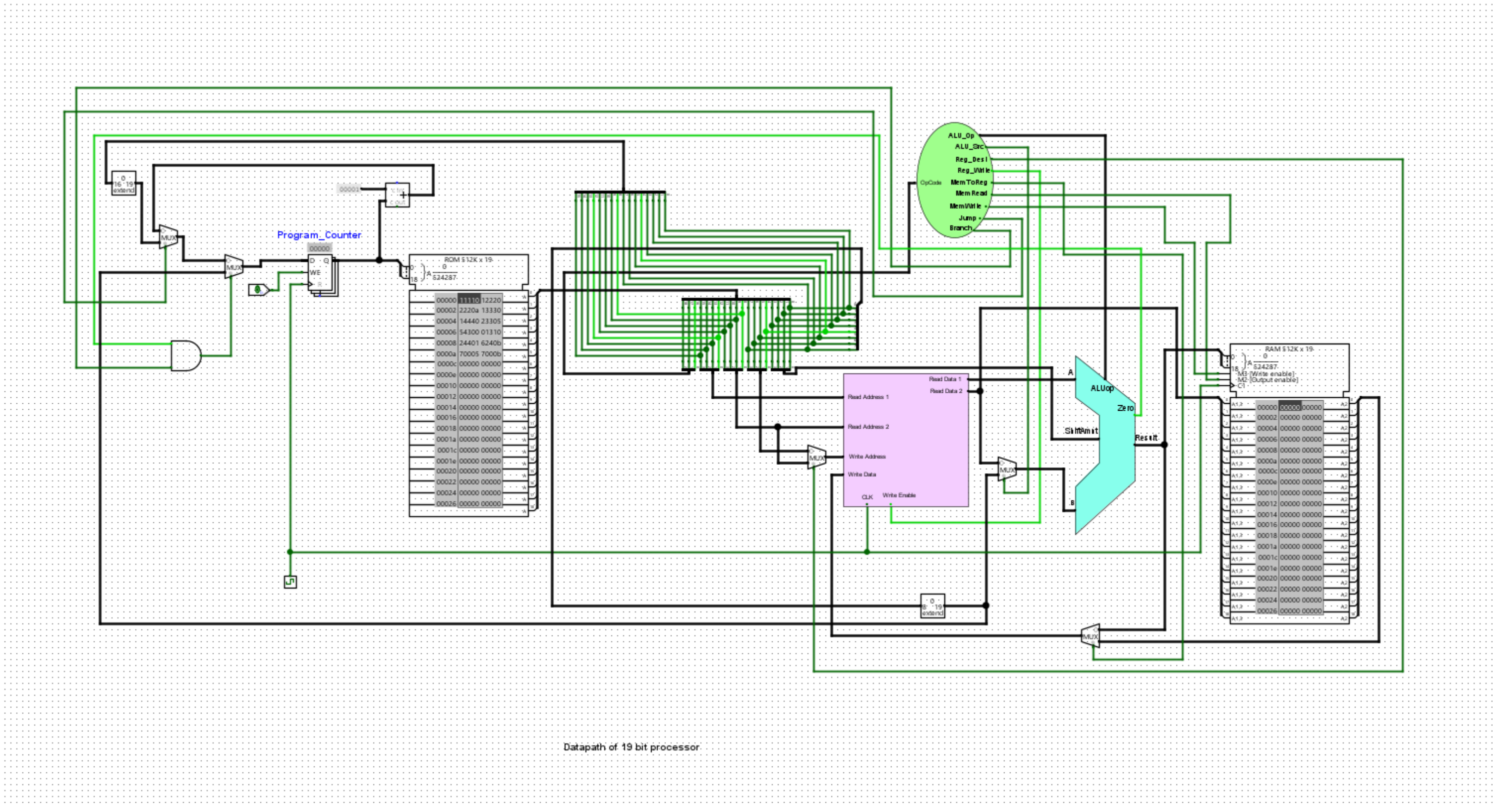
Instructions	Oppcode	ALUop	ALUSrc	RegDest	RegWrite	MEMTO REG	MEMREAD	MEMWRITE	Jump	Branch
ADD	000	000	0	0	1	0	0	0	0	0
SUB	001	001	0	0	1	0	0	0	0	0
ADDi	010	000	1	1	1	0	0	0	0	0
XOR	011	101	0	0	1	0	0	0	0	0
Load	100	000	1	1	1	1	1	0	0	0
Store	101	000	1	X	0	X	0	1	0	0
Branch	110	001	0	X	0	X	0	0	0	1
Jump	111	xxx	x	x	0	x	0	0	1	0

Screenshot:



Design of Control Unit

Final Datapath Design:



Datapath of 19 bit processor

Compiled Codes:

Problem: Store the first 10 numbers of the series 5+10+15+20+25+30+…...+n in memory and their sum on $R1 using a loop.

- Code in C Language:

```
#include <stdio.h>
int main() {
int n = 10;
int start = 5;
int diff = 5;
int sum = 0;
for (int i = 0; i < n; i++) {
    sum += start;
    start += diff;
    }

    printf("sum of the first %d numbers of the series is: %d\n", n, sum);
    return 0;
}
```

- Pseudocode:

  Initialization:

  0. Load R1,0
     Assembly: SUB R1 R1 R1 0   [Initialized the R1 register to be 0]
     Binary: 00010001000100010000
     Hexadecimal: 11110

  1. Load R2,0
     Assembly: SUB R2 R2 R2 0   [Initialized the R2 register to be 0]
     Binary: 00010010001000100000
     Hexadecimal: 12220

  2. Addi R2+10
     Assembly: Addi R2 R2 10   [initialized the R2 register as counter to be 10]
     Binary: 00100010001000001010
     Hexadecimal: 2220A

  3. Sub R3,0
     Assembly: Sub R3 R3 R3 0   [Initialized the R3 register to be 0]
     Binary: 00010011001100110000
     Hexadecimal: 13330

  4. Load 0
     Assembly: Sub R4 R4 R4 0   [Initialized the R4 register to be 0]
     Binary: 00010100010001000000
     Hexadecimal: 14440

Loop:

5. Addi R3,5
   Assembly: Addi R3 R3 5 [store the 5 in Register]
   Binary: 00100011001100000101
   Hexadecimal: 23305

6. Store R4 R3 0
   Assembly: SW R4 R3 0 [store in the memory in 0 address]
   Binary: 01010100001100000000
   Hexadecimal: 54300

7. Add R1, R3, R1
   Assembly: Add R1 R3 R1 0 [Adding to R3]
   Binary: 00000001001100010000
   Hexadecimal: 01310

8. Addi R4, R4,1
   Assembly: Addi R4 R4 1 [Incrementing 1 in the R4]
   Binary: 00100100010000000001
   Hexadecimal: 24401

9. Branch R2 R4 11 [End]
   Assembly: Beq R2 R4 11 [If it is equal, then go to instruction 11, else go to 10]
   Binary: 01100010010000001011
   Hexadecimal: 6240B

10. Assembly: JMP 5 [Again, go back to 5th Instruction]
    Binary: 01110000000000000101
    Hexadecimal: 70005

11. Assembly: JMP 11 [If all ends, then be stuck at 11]
    Binary: 01110000000000001011
    Hexadecimal: 7000B

Discussion:

In this project, I built a Datapath for the 19-bit processor. The main issue I faced during the whole project was mainly the concept. I had to learn everything necessary, especially the I and J-type instruction sets. The instructions for my operation were relatively low, which I realized later while doing the problem I was given. The decision to make the opcode was made very quickly. As a result, a lot of mental toiling was needed to solve the problem. But finally, the result was visible. I observed that even after all the operations ended, it still went to the next step, avoiding the loop condition, so the workaround I did was to jump the instruction to the same line after the loop satisfied the condition. It ensured that the expected result stayed in the same line. Hence, it stored the correct result for the RAM and the register. So, this rules out any error in the Datapath.

Overall, I can say that this project took a lot of effort to accomplish the task it was supposed to do, and it suffice to say that it worked. The testing result that I needed was 275, and as the value stored in the register was 113, converting it to decimal showed 275, which means the Datapath worked successfully.