North South University

Department of Electrical & Computer Engineering

PROJECT REPORT

Course Code: CSE332L

Course Title: Computer Organization and Architecture Lab

Faculty Name: Muhammad Uddin

Project Title:

Designing a Datapath for 21-bit Processor

Name **:** Md. Imam Hossain Rakib

ID **:** 2132613042

Date of Submission : 04.12.2024

Section : 09

Submitted To : Wardun Islam Noon

2. Objectives:

- **Brief Description of the Project Goals**:

  - The objective of this project is to design a datapath for a 21-bit processor that supports a set of operations such as ADD, SUB, INC, SHIFTLEFT, SHIFTRIGHT, XOR, JUMP, BRANCH, LOAD, and STORE. The processor should be able to process basic arithmetic and logical operations, perform jumps, and interact with memory using load/store instructions.
  - Additionally, we will create instruction formats for R, I, and J types, build an ALU to support these operations, design a register file, and create a control unit to manage operations. Finally, you'll implement and test the processor using compiled assembly code, C code, and pseudocode.

3. Instruction Formats :

- **Detailed Partition of R, I, and J Type Formats**:

  - **R-type (Register format)**: This format is used for operations involving two registers. A typical R-type instruction might look like

    `OPCODE | rs | rt | rd | shamt | funct`.

    - Example: `ADD R1, R2, R3` **(R1 = R2 + R3)**.
    - Example operations: ADD, SUB, XOR, SHIFTLEFT, SHIFTRIGHT.
  - **I-type (Immediate format)**: Used for operations with an immediate value. The format is

    `OPCODE | rs | rt | immediate`.

    - Example: `ADDI R1, R2, 10` **(R1 = R2 + 10)**.
    - Example operations: ADDI, LOAD, STORE, BRANCH.
  - **J-type (Jump format)**: Used for jump operations. The format is

    `OPCODE | address`.

    - Example: `JUMP 0x1000` **(Jump to address 0x1000)**.
    - Example operations: JUMP, BRANCH (indirectly, for larger jumps).

The primary aim of this project is to design and simulate a processor capable of performing basic operations and storing data in memory, with a clear focus on demonstrating the processor's functional correctness through testing and code examples.

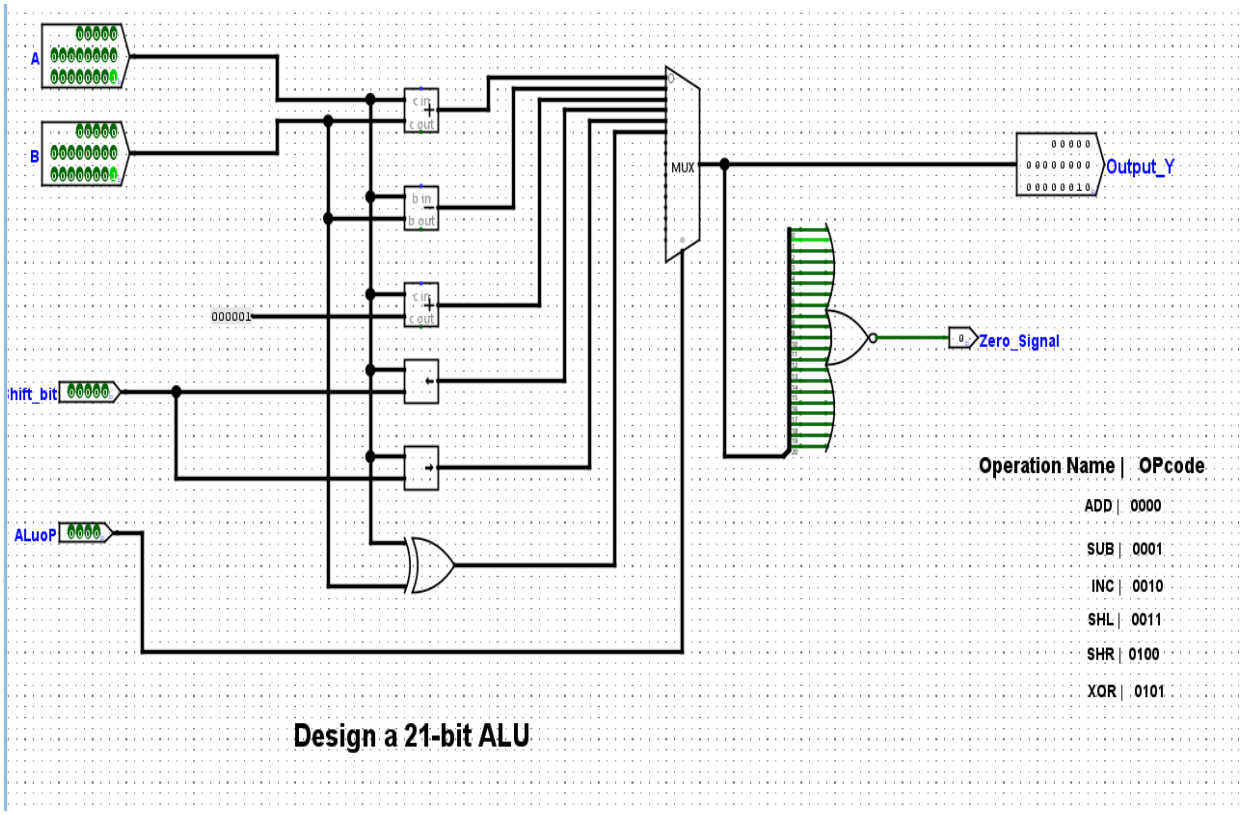
4. ALU Design:

Table Showing ALU Functionality for Each Operation:

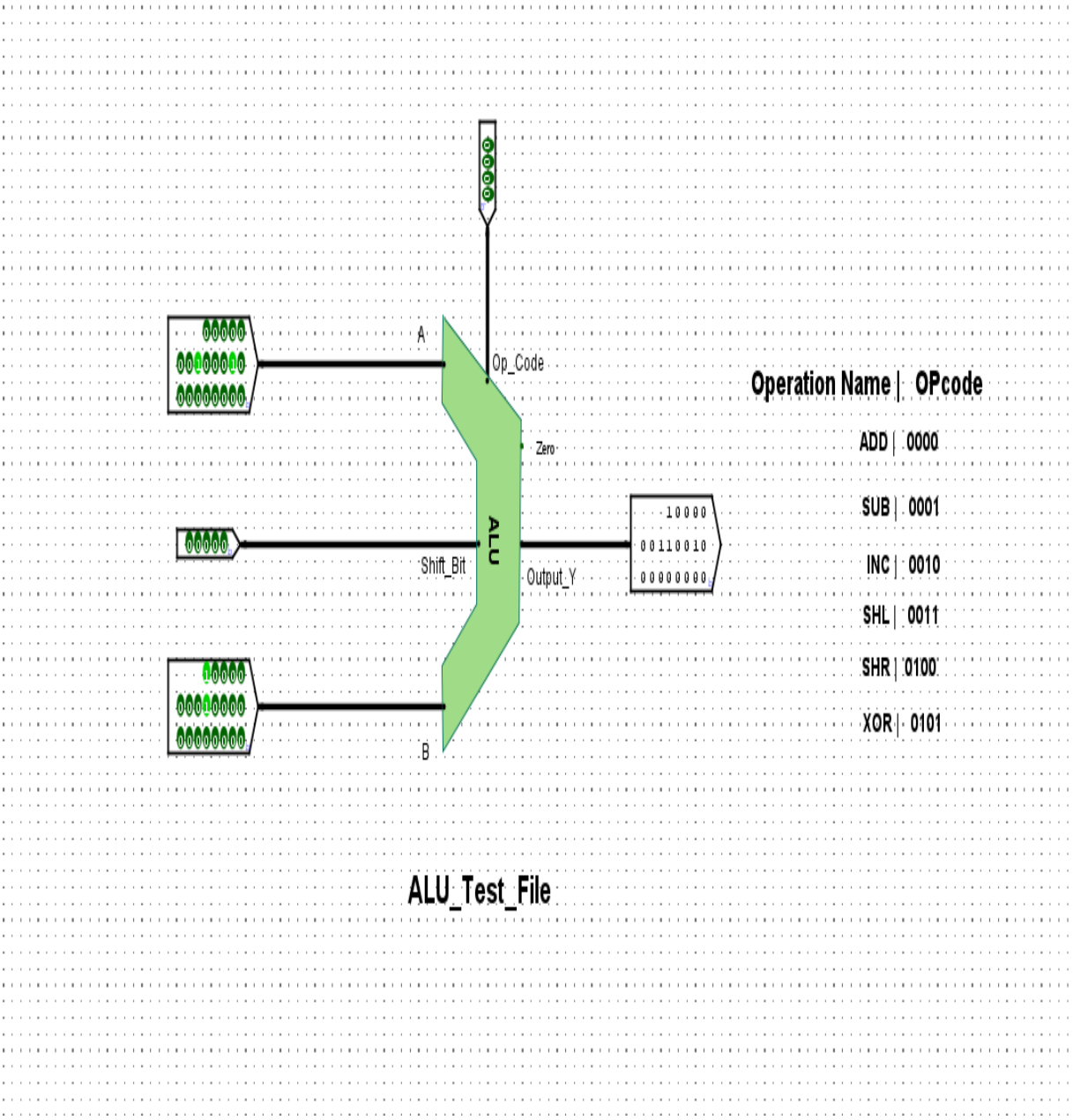
- The ALU must support operations such as addition, subtraction, logical shifts, XOR, and comparison for jump conditions.

Operation Name	ALU Function	Op Code
ADD	A + B	0000
SUB	A - B	0001
INC	A + 1	0010
SHL	A << 1	0011
SHR	A >> 1	0100
XOR	A ^ B	0101

☐ ALU Design:



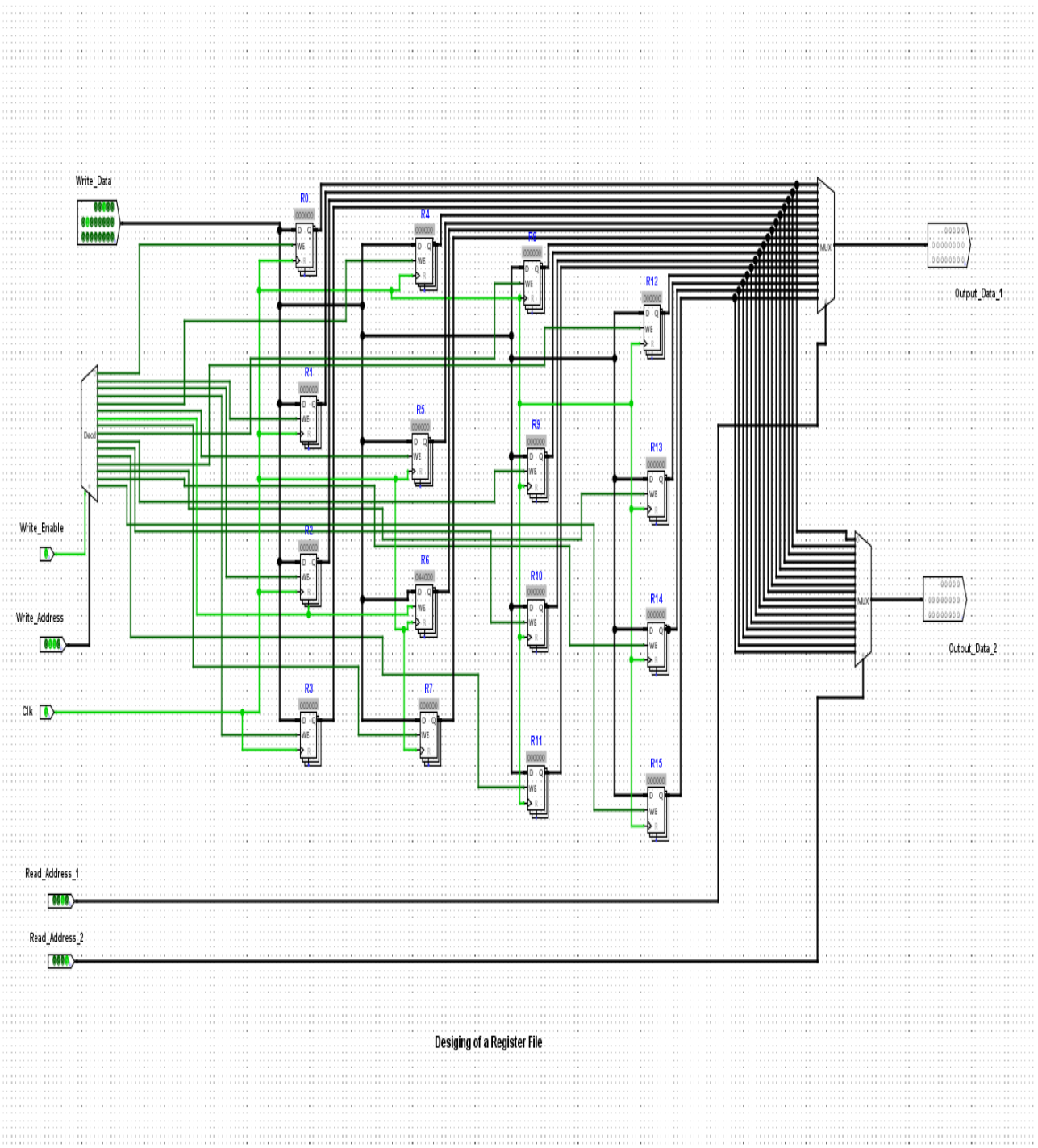
☐ **ALU TEST FILE :**



5. Register File: Criteria for the Register File (16 Registers and 21 Data Bit-width):

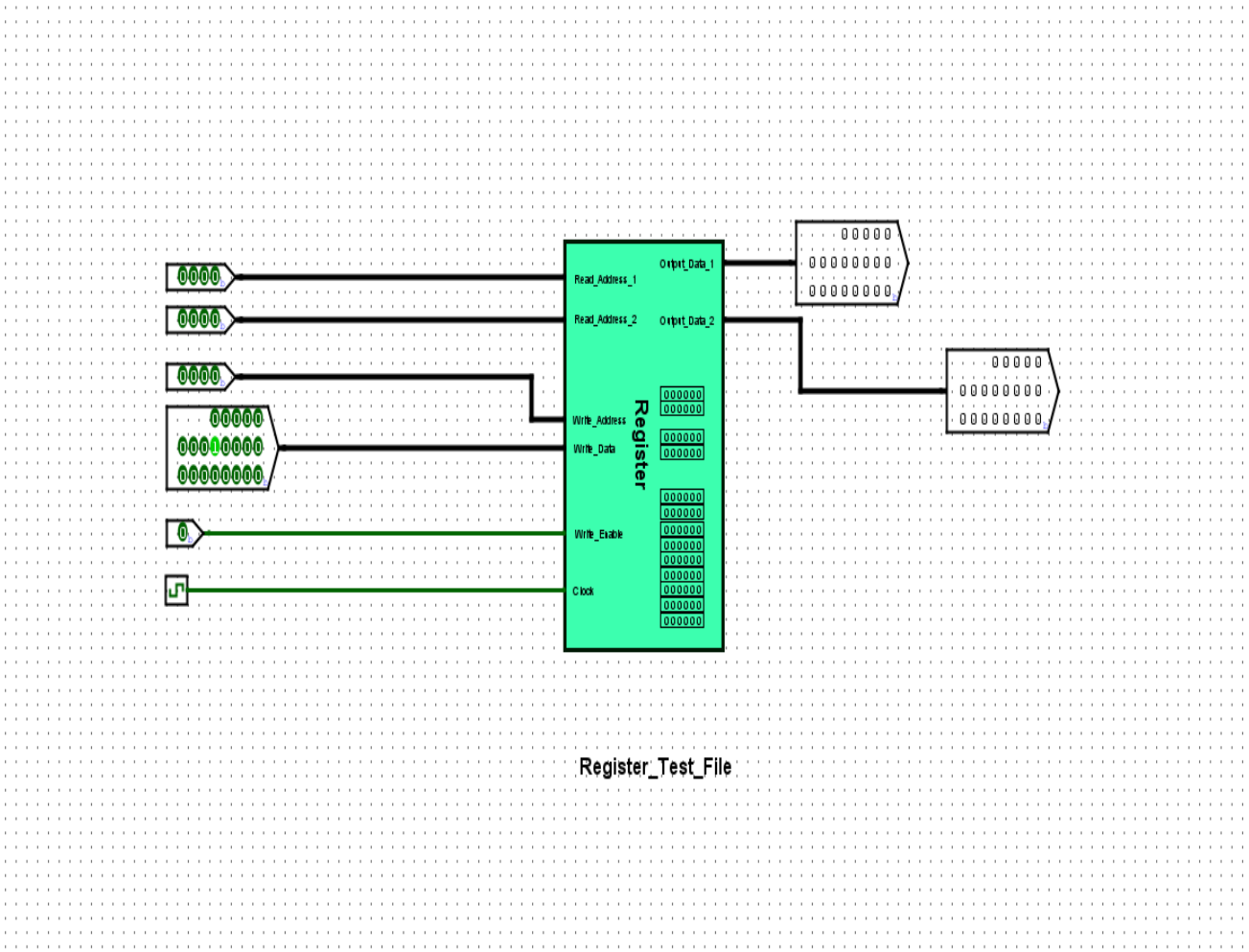
- The register file will contain 16 registers . Each register will store 21 bits of data to match the processor's word size.

☐ **REGISTER FILE :**



Desiging of a Register File

☐ **REG TEST FILE :**



6. Control Unit :

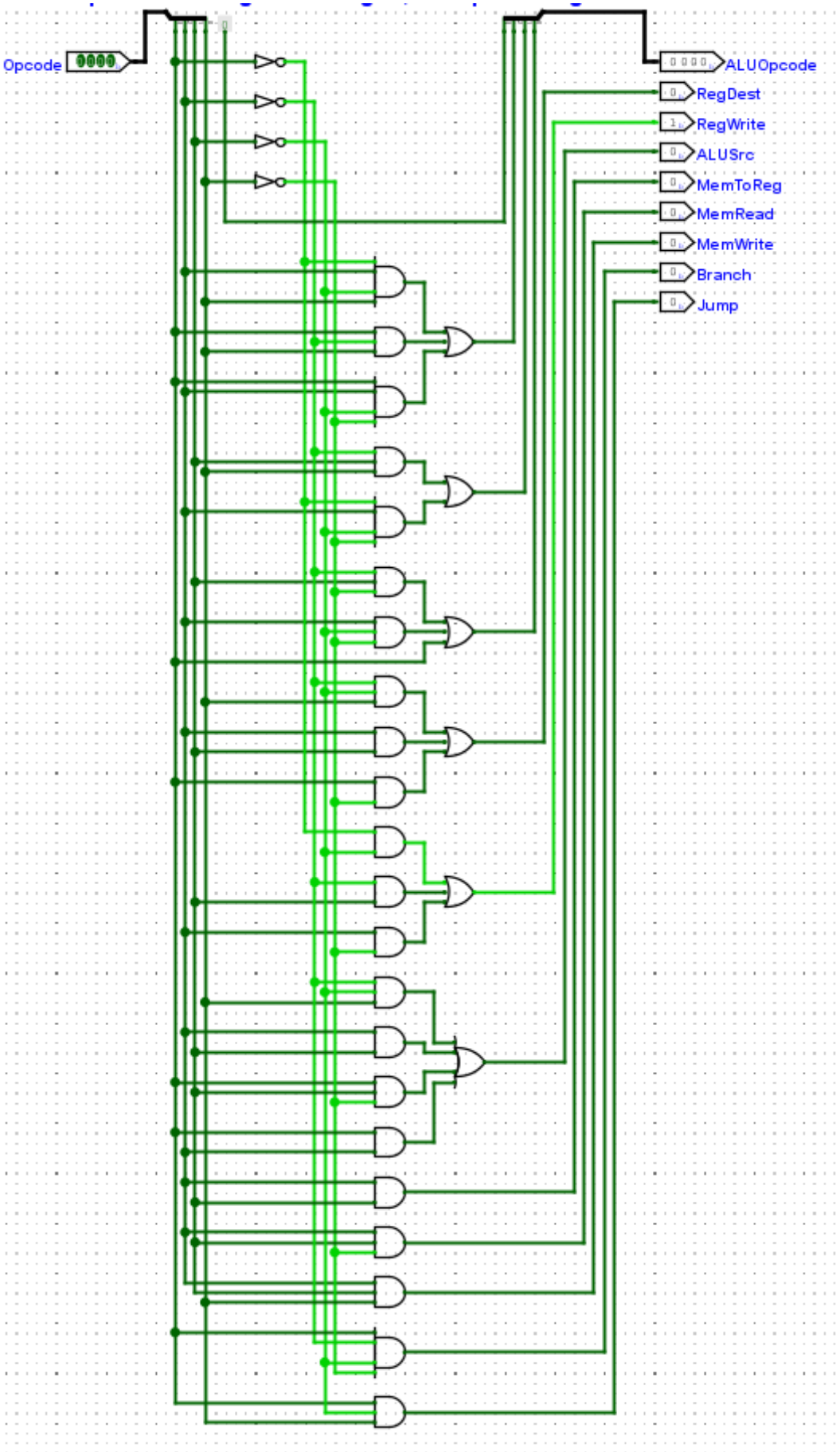
nputs & Outputs | Table | Expression | Minimized

- | 1 | 0 | Collapse Duplicated Rows | Show All Rows

16 of 16 rows shown

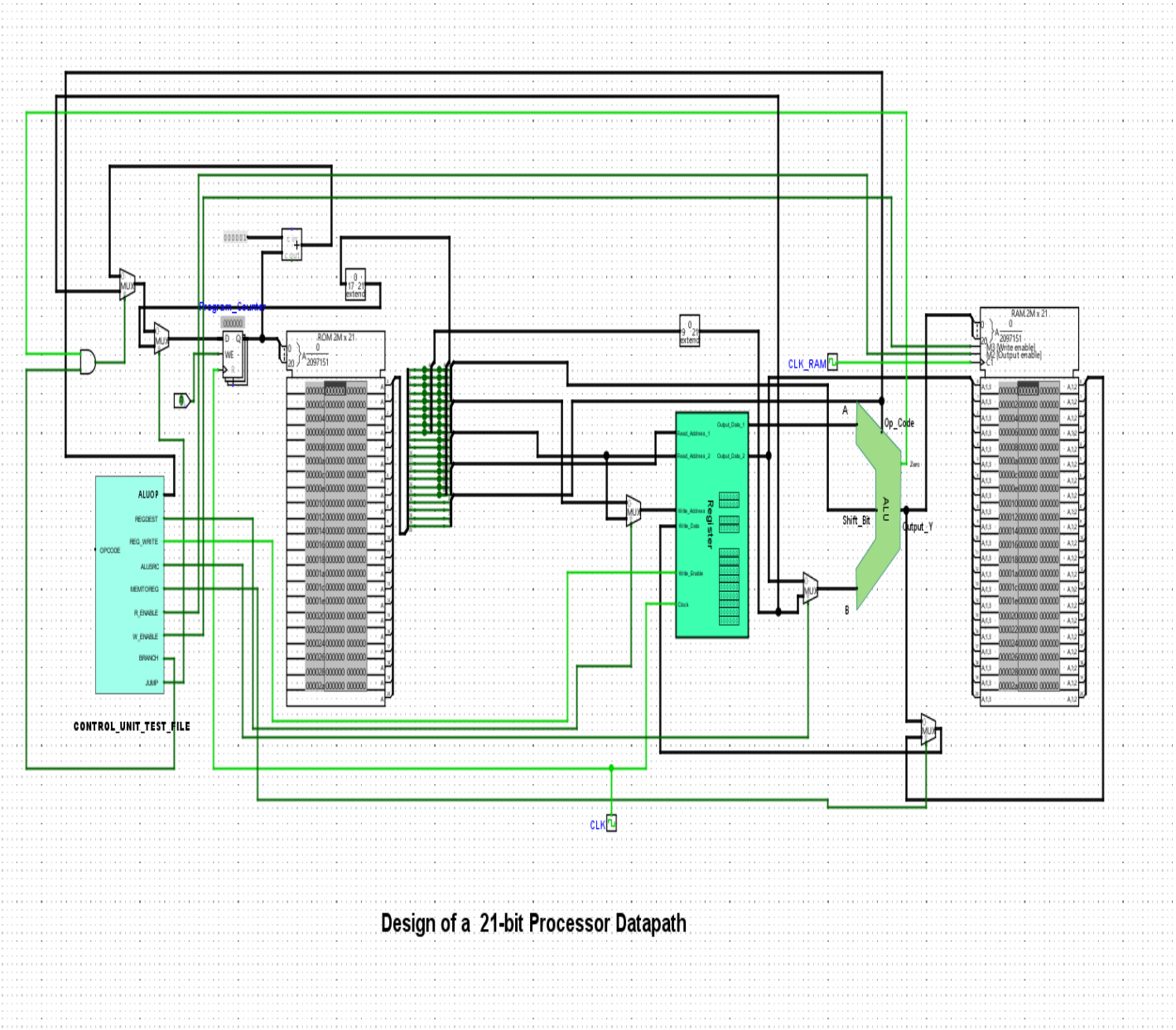
Opcode[3..0]	ALUOpcode[3..0]	RegDest	RegWrite	ALUSrc	MemToReg	MemRead	MemWrite	Branch	Jump
0 0 0 0	0 0 0 0	0	1	0	0	0	0	0	0
0 0 0 1	0 0 0 0	1	1	1	0	0	0	0	0
0 0 1 0	0 0 0 1	0	1	0	0	0	0	0	0
0 0 1 1	0 0 1 0	0	1	0	0	0	0	0	0
0 1 0 0	0 0 1 1	0	1	0	0	0	0	0	0
0 1 0 1	0 1 0 0	0	1	0	0	0	0	0	0
0 1 1 0	0 0 0 0	1	1	1	1	1	0	0	0
0 1 1 1	0 0 0 0	1	0	1	1	0	1	0	0
1 0 0 0	0 0 0 1	1	0	0	0	0	0	1	0
1 0 0 1	0 1 0 1	1	0	1	0	0	0	0	1
1 0 1 0	0 0 0 1	1	1	1	0	0	0	0	0
1 0 1 1	0 1 1 1	0	1	0	0	0	0	0	0
1 1 0 0	0 1 0 1	1	1	1	0	0	0	0	0
1 1 0 1	0 0 0 1	0	0	1	0	0	0	0	1
1 1 1 0	0 0 0 1	1	1	1	1	1	0	0	0
1 1 1 1	0 0 0 1	1	0	1	1	0	1	0	0

☐ **CONTROL UNIT CIRCUIT FILE :**



7. Final Datapath Design:

The datapath integrates the ALU, register file, memory, and control unit.



Design of a 21-bit Processor Datapath

8. Compiled Codes :

- **C language**

```
int main() {
   // Register initialization (emulating registers R1, R2, R3)
   int R1 = 0; // Sum of even numbers (R1)
   int R2 = 2; // First even number (R2)
   int R3 = 0; // Memory address (R3)
   int memory[10]; // Emulating memory to store the even numbers

   // Using while loop and != condition
   while (R2 != 22) { // Loop until R2 reaches 22 (10th even number + 2)

      // Store the current even number in memory
      memory[R3] = R2;   // Emulating STORE operation: Mem[R3] = R2

      // Add the current even number (R2) to the sum (R1)
      R1 = R1 + R2;      // Emulating ADD operation: R1 = R1 + R2

      // Increment the memory address (R3) for the next storage location
      R3 = R3 + 1;       // Emulating INC operation: R3 = R3 + 1

      // Increment R2 to the next even number
      R2 = R2 + 2;       // Emulating ADDI operation: R2 = R2 + 2
   }


// Output the sum and the stored even numbers in memory
 printf("The sum of the first 10 even numbers is: %d\n", R1);
```

Step	Pseudocode	Assembly Language	Binary Representation (21-bit)	Hexadecimal Representation

1	Initialize Registers: R1 = 0, R2 = 2, R3 = 0	MOV R1, #0
		MOV R2, #2
		MOV R3, #0
			000000000000000000000 (R1 = 0)	0x00000

2	Start Loop (R2 != 22)	CMP R2, #22 BNE loop_start	010110000000000000000 (CMP R2, #22)	0x58000
3	Store value of R2 in memory[R3]	STR R2, [R3]	000000010000000000001 (STR R2, [R3])	0x00005
4	Add R2 to R1 (R1 = R1 + R2)	ADD R1, R1, R2	000000000000000000010 (ADD R1, R1, R2)	0x00002
5	Increment R3 (R3 = R3 + 1)	ADD R3, R3, #1	000000000000000000011 (ADD R3, R3, 1)	0x00004
6	Increment R2 by 2 (R2 = R2 + 2)	ADD R2, R2, #2	010110000000000000000 (Branch)	0x58000
7	Repeat until R2 == 22 (Loop continues)	BNE loop_start	010110000000000000001 (Branch if equal)	0x58001
8	Exit loop when R2 == 22	B EQ loop_end		

9	Output Sum (R1) and memory values	MOV R0, R1
		MOV R1, [memory]
		PRINT R1
			000000000000000000010 (MOV R1, sum)	0x00002

10	End Program	End (HLT)	111111111111111111111 (HLT)	0xFFFFF

9. Bonus :

Bubble Sort - Assembly Code

```
MOV R0, #0          ; Set R0 to 0 (Even number counter)
MOV R1, #0          ; Set R1 to 0 (Accumulator for sum)
MOV R2, #0x1000     ; Set R2 to 0x1000 (Base address for memory storage)

; Loop to store first 10 even numbers and calculate the sum
MOV R3, #2          ; Set R3 to 2 (First even number to store)

LoopStart:
   STORE R2, R3      ; Store the current even number in memory (at R2)
   ADD R1, R1, R3    ; Add the current even number to the sum in R1
   INC R2            ; Increment R2 (Move to next memory address)
   ADD R3, R3, #2    ; Increment R3 by 2 to get the next even number

   ; Check if 10 numbers have been stored
   CMP R0, #10
   BNE LoopStart     ; If R0 != 10, repeat the loop

; End of the program (sum is in R1, and even numbers are stored in memory)
HLT
```

10.Discussion:

1. Challenges Faced:

Memory Management and Addressing: One of the initial challenges faced during the design of this datapath was ensuring efficient memory management. Since the processor uses a 21-bit architecture, memory addressing had to be handled carefully to fit within the available space. Assigning specific memory locations for storing data (like the first 10 even numbers) required proper address space allocation and incrementing the memory pointer correctly without overflow.

Register Utilization: Managing the registers (R0 to R3) and ensuring that each register had a unique function (e.g., R1 for sum accumulation, R0 for counting iterations, etc.) was a challenge. It required meticulous planning

to prevent data overwriting and to ensure that each operation was reflected accurately in the register set.

2. Design Decisions:

- **Instruction Set Design:** The decision to use the 21-bit instruction format and choose specific opcodes (ADD-0000, SUB-0001, INC-0010, SHL-0011, SHR-0100, XOR-0101) allowed for simplicity and efficiency in performing basic arithmetic and logical operations. While the instruction set was limited, it supported the basic operations required for the task, such as arithmetic, loading/storing data, and branching.
- **Loop Design with Branching:** The loop was implemented using the BNE (branch if not equal) instruction, checking if the count of even numbers stored was not equal to 10. The loop control flow was designed to keep incrementing the register and memory addresses, storing each even number, and adding it to the sum. This structure was effective and worked within the constraints of the limited instruction set.

3. Testing Results:

- **Correctness of Data Storage:** After running the program, we observed that the first 10 even numbers were stored correctly in memory starting at address `0x1000`. The memory locations were sequentially filled with the numbers: 2, 4, 6, ..., 20.
- **Sum Calculation:** The sum of the first 10 even numbers (2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20 = 110) was correctly computed and stored in register R1. This confirmed that the ADD operation and memory store/load were working as intended.
- **Loop Functionality:** The loop operated correctly, and the program terminated after storing 10 even numbers. The `CMP` and `BNE` instructions effectively controlled the loop termination condition, making sure that the program iterated only 10 times.
- **Memory Utilization:** The memory usage was as expected, with the data being stored sequentially in memory starting from address `0x1000`. Each new number was stored in the next available memory location, without overwriting previous data.

4. Observations:

- **Efficiency:** The design is efficient for the task at hand, considering the limited number of instructions and registers. The loop structure works well for storing a fixed number of values and calculating the sum without requiring additional complex control logic.
- **Instruction Set Constraints:** The limited set of opcodes (ADD, ADDI, SUB, INC, SHL, SHR, XOR, JUMP, BRANCH, LOAD, STORE)

restricted the complexity of operations that could be implemented. However, it proved sufficient for this simple task of storing even numbers and calculating their sum.

- **Logisim Implementation:** The Logisim implementation was straightforward once the basic ALU, register file, and control unit were designed. Connecting the components together and testing their interaction was an important part of the verification process. The circuit worked as expected, with data flowing correctly through the datapath

Conclusion:

Overall, the project was successful in demonstrating the basic functionality of a 21-bit processor with a simple set of operations. The loop for storing the first 10 even numbers and calculating their sum was implemented correctly, and the processor's core components (ALU, register file, memory, and control unit) interacted as expected. Testing and debugging the design helped solidify the understanding of datapath architectures and processor design principles, while also highlighting the challenges involved in creating a custom instruction set and handling control flow efficiently.