Bangladesh University of Engineering & Technology

Course no: CSE206

Couse Title: Computer Architecture Sessional

Assignment no.: 03

Section: B1

Group no.: 05

Group Members: 1905071 - Tareq Ahmed

1905078 - Mehreen Tabassum Maliha

1905082 - Kazi Reyazul Hasan

1905084 - Wasif Jalal

1905088 – Mubasshira Musarrat

Date of Submission: 28/02/2023

Introduction:

A processor (CPU) is the logic circuitry that responds to and processes the basic instructions that drive a computer. Processor is capable of doing operations like arithmetic operations, Logic operations and I/O operations etc. MIPS (Microprocessor without Interlocked Pipelined Stages) is a family of reduced instruction set computer (RISC) instruction set architectures (ISA) developed by MIPS Computer Systems, now MIPS Technologies, based in the United States.

Arithmetic Logic Unit (ALU) is the fundamental component of a processor which can perform any instructions like program count, arithmetic and logical operations, address count, finding the jumping address, calculating memory address and so on. The other components needed for a processor are address and data buses, register files, data memory, control unit etc.

In this assignment, we have to implement a 4-bit processor that takes a reduced version of the MIPS instruction set. We need 4-bit ALU for implementing this processor. For each instruction set, one clock cycle is needed and all the instructions are executed on the falling edge of the clock. In this assignment, the following components are designed to implement the processor:

1.4-bit ALU:

This ALU is capable of doing all the arithmetic and logical operations. It is also responsible of doing all the addition and subtraction operations which are required for finding load and store addresses, unconditional jump or branching instructions.

2.Program Counter:

A program counter is a register in a computer processor that contains the address(location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched. The program counter points to the next instruction in the sequence. Here the ROM we have used is word (16 bit) addressable, adding 1 takes us to the next instruction written in the ROM. The stored value indicates the Instruction memory address. When the computer restarts or is reset, the program counter normally reverts to 0.

3. Instruction Memory:

Instruction memory stores all the prefetch instructions which are the actual hex code of the instruction which is converted to binary to run the processor.

4. Data Memory:

Data memory serves for storing 4-bit data and keeping data for required for the proper operation of the programs. It works as the main memory.

5. Register Files:

Temporary registers $zero, $t0, $t1, $t2, $t3, $t4 are used. All of these are of 4-bits. These registers store values.

6. Control Unit:

Control unit is circuitry within a computer's processor that directs operations. It instructs the memory, logic unit, and both output and input devices of the computer on how to respond to the program's instructions. Actually, it gives the selection input to all the MUXs, Data Memory, ALU, Register File and decodes the instructions.

MIPS provides different types of instruction formats for different types of operations:

- **R-Format:**
  To deal with arithmetic operations with registers

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- op: shorthand of opcode, denotes operation type and format type
- rs: Source Register1
- rt: Source Register 2
- rd: Destination Register
- $shamt$: Shorthand of shift amount
- $funct$: shorthand of function code, denotes the specific variant of an operation

- **I-Format:**
  To deal with constants and data transfer operations

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

Here, rs:= source register

rt:= destination register

constant or address := the constant or the address to store/retrieve the value to/from

- **J-Format:**

  To support long jump to a remote address.

op	jumping address
6 bits	26 bits

A datapath is built with elements that process data and addresses in the CPU such as registers, ALUs, MUXs, memories and controls. The MIPS instructions are fetched through a datapath to perform different instructions such as addition, load/store, branching etc. Actually a datapath is a depiction of flow of data through different components of a computer while executing a certain instruction. It can be created by joining more than one together using multiplexers.

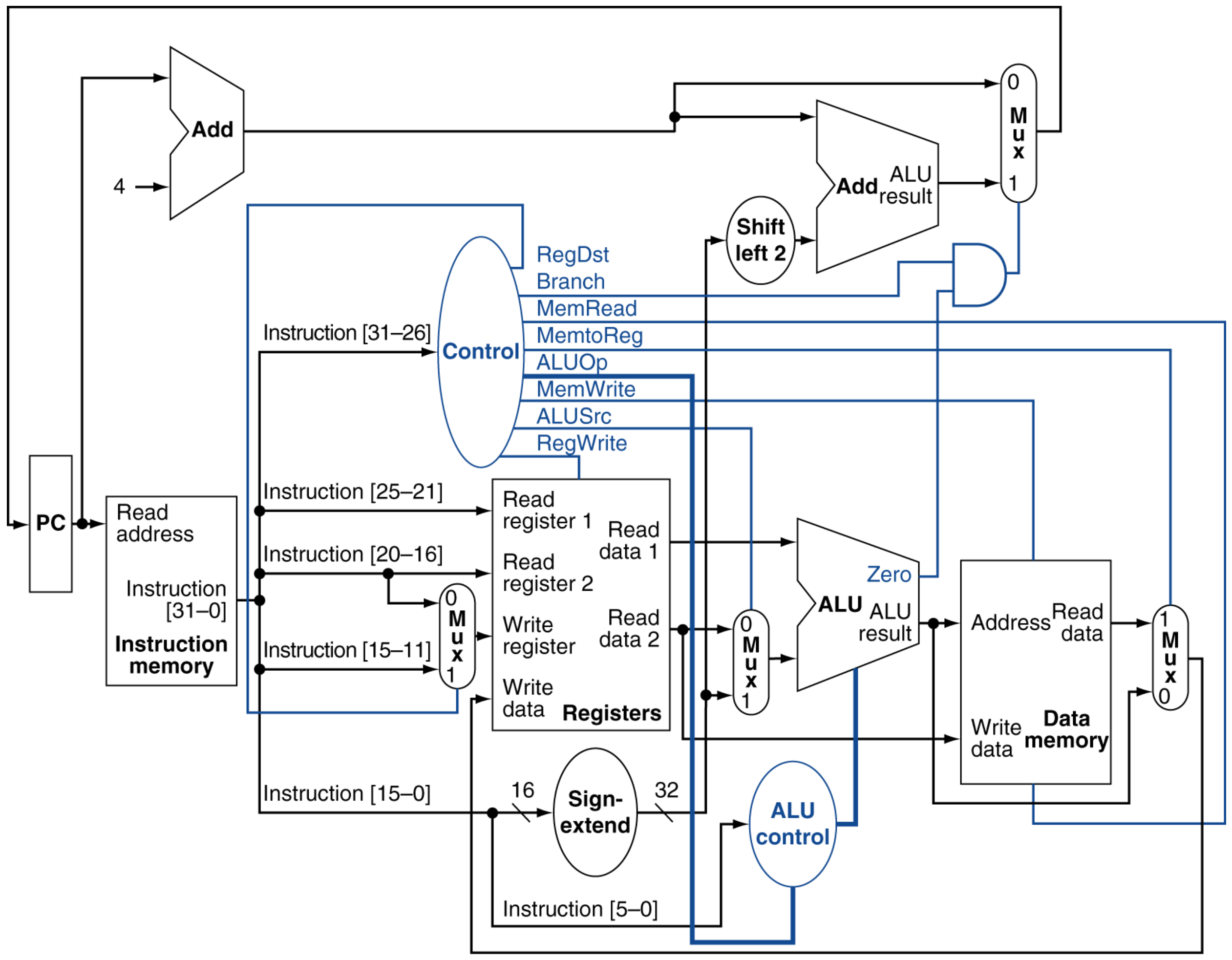


Fig. Datapath with Control

Instruction Set:

The mentioned sequence:

IJPBHDAFGMNECKOL

ID	Instruction	Type	Format	Opcode
I	sll	Logic	S-type	0
J	srl	Logic	S-type	1
P	j	Control	J-type	2
B	addi	Arithmetic	I-type	3
H	ori	Logic	I-type	4
D	subi	Arithmetic	I-type	5
A	add	Arithmetic	R-type	6
F	andi	Logic	I-type	7
G	or	Logic	R-type	8
M	*sw*	*Memory*	*I-type*	*9*
N	*beq*	*Control*	*I-type*	*10*
E	*and*	*Logic*	*R-type*	*11*
C	*sub*	*Arithmetic*	*R-type*	*12*
K	*nor*	*Logic*	*R-type*	*13*
O	*bneq*	*Control*	*I-type*	*14*
L	*lw*	*Memory*	*I-type*	*15*

MIPS Instruction Format

Our MIPS instructions will be 16-bits long with the following four formats

- **R-type**

Opcode	Src reg 1	Src reg 2	Dest reg
4-bits	4-bits	4-bits	4-bits

- **S-type**

Opcode	Src reg 1	Dest reg	Shamt
4-bits	4-bits	4-bits	4-bits

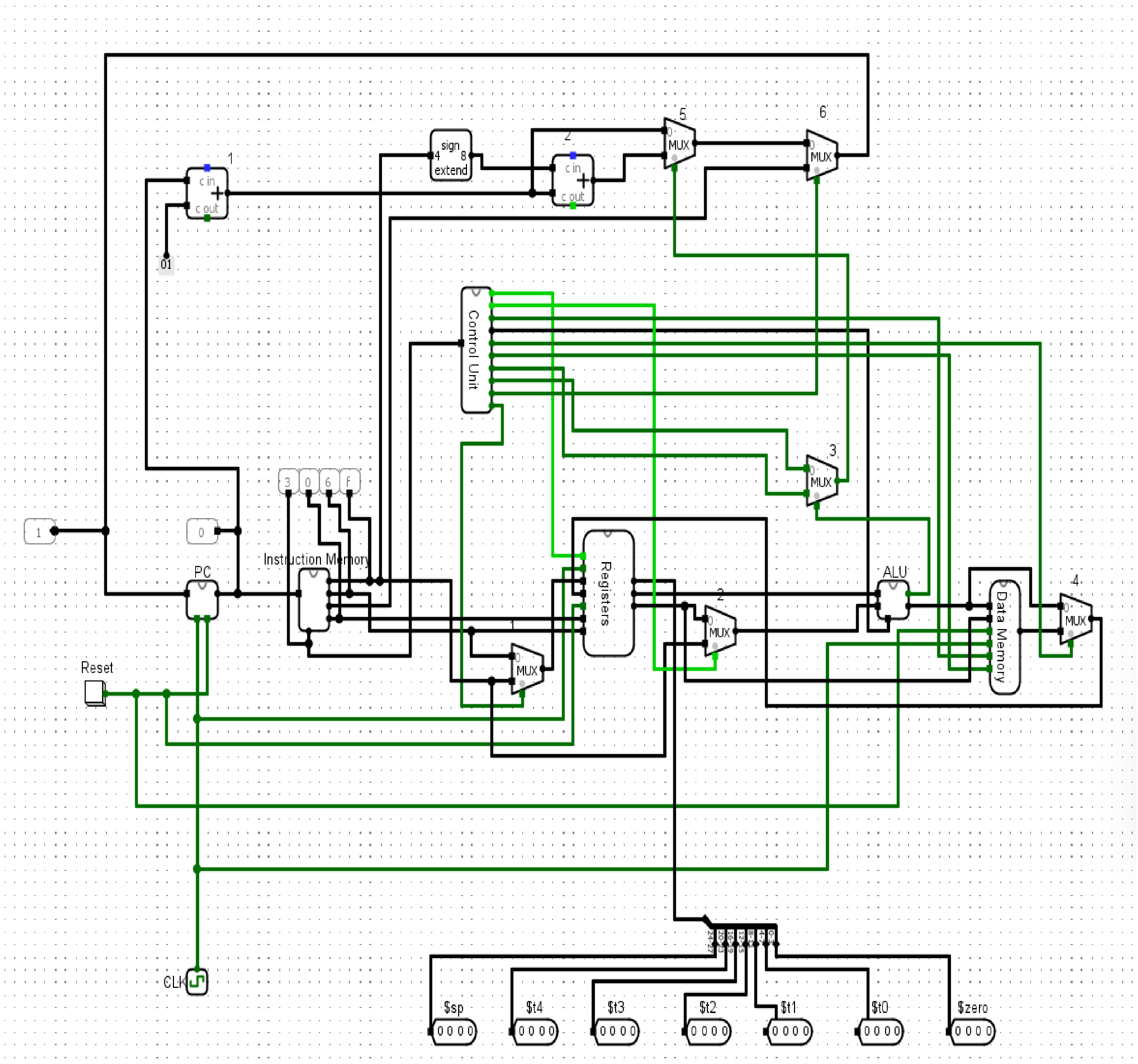
- **I-type**

Opcode	Src reg 1	Src reg 2/Dest reg	Address/Immdt.
4-bits	4-bits	4-bits	4-bits

- **J-type**

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

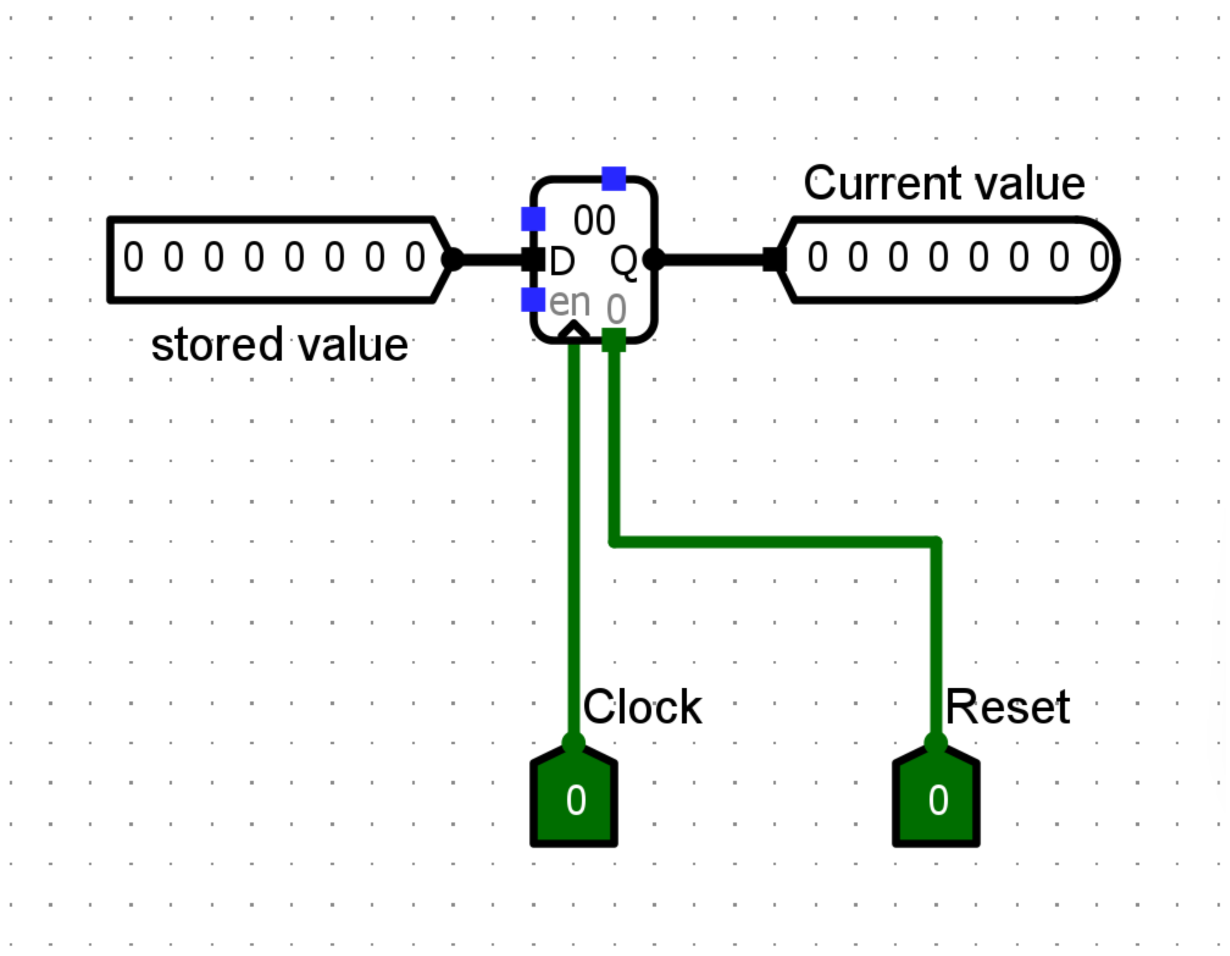
Circuit Diagram with Design steps:



Datapath for 4-bit MIPS

PC:

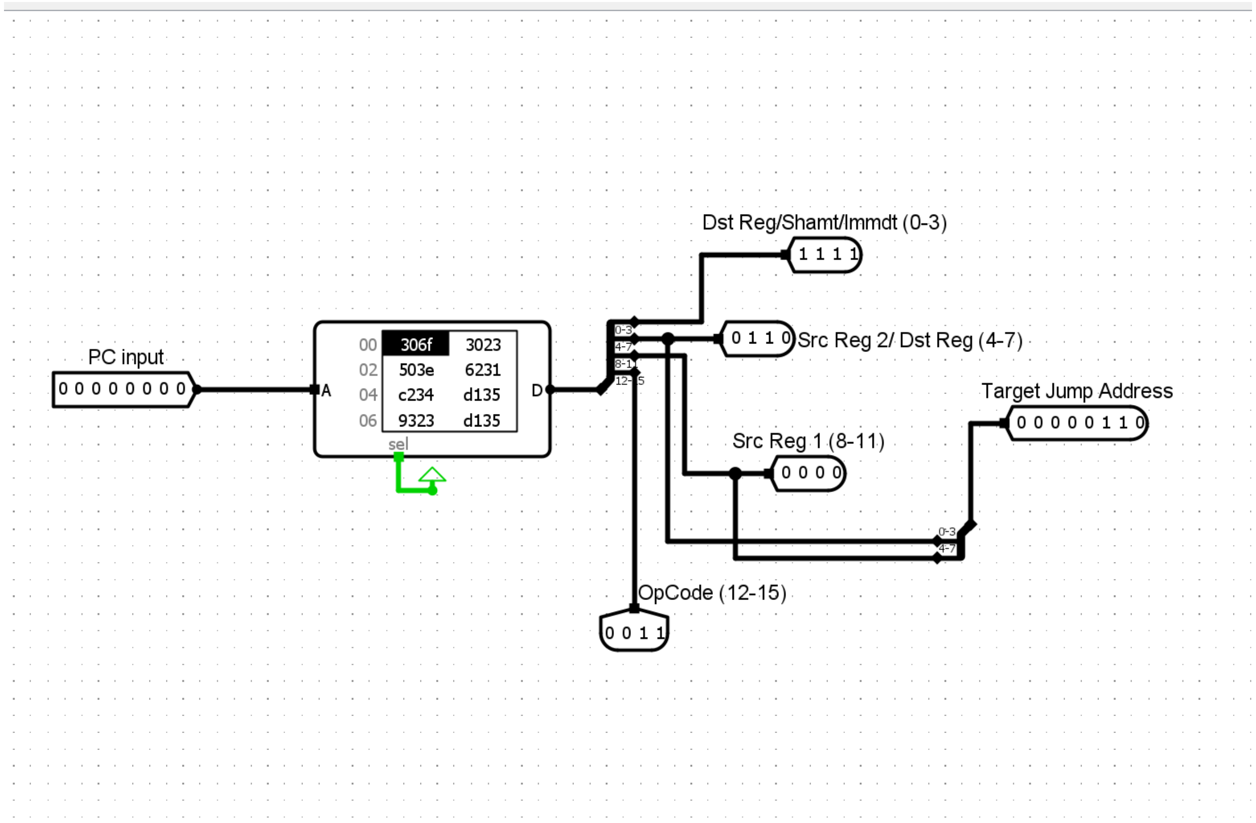
Program counter is a 4-bit D flipflop that holds the address of the current instruction. It is written at the end of every clock cycle and thus does not need a write control signal. PC is incremented by 1 after each instruction in case of normal dataflow.



PC

Instruction Memory:

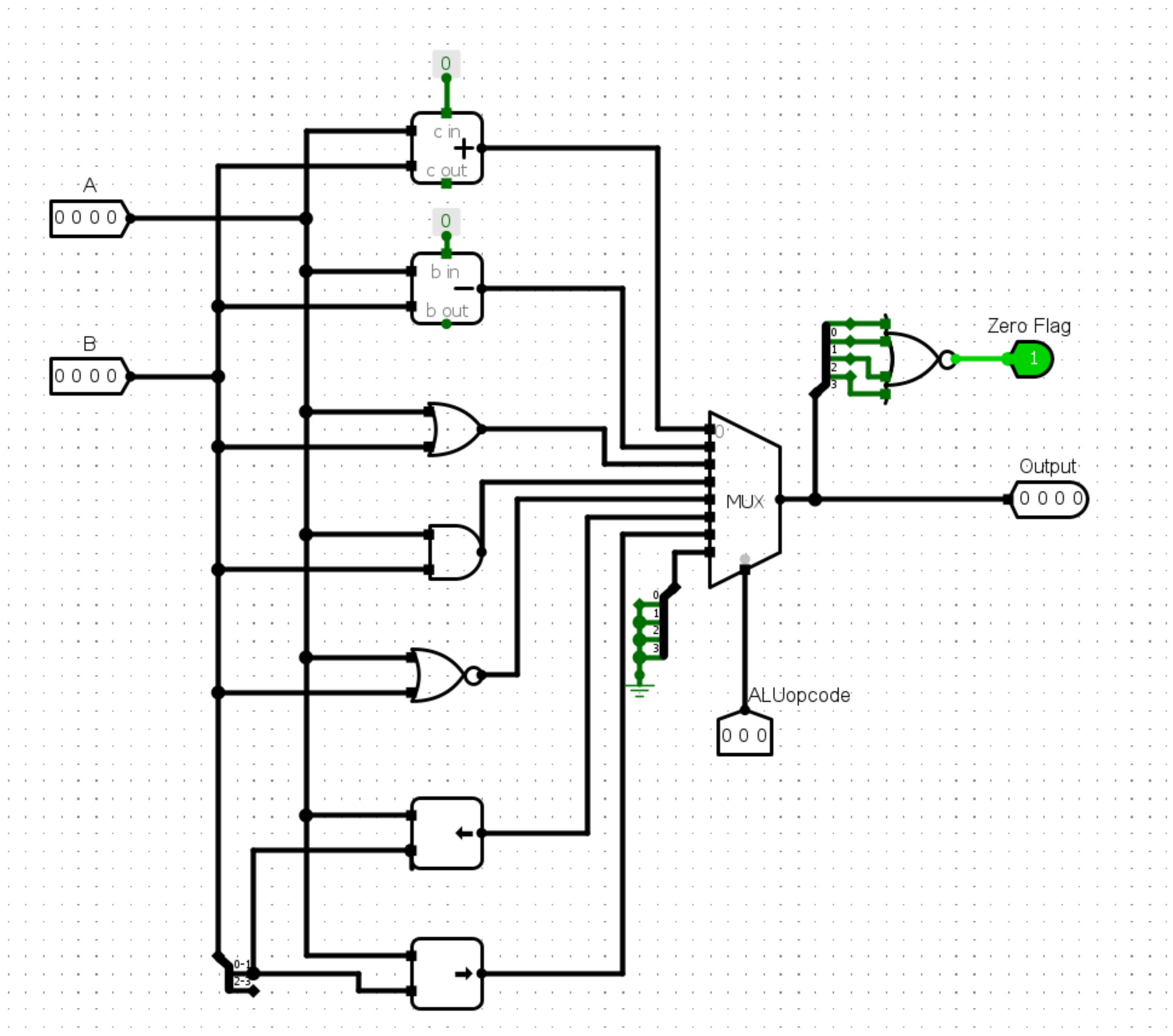
Here an EPROM is used to store the instructions in the instruction memory. The instructions are provided as input. Then these are converted to MIPS 16-bit instructions. After each clock cycle, the PC value is incremented by 1 & the instruction memory gives a new output instruction. These instructions are divided into four 4-bit values. These four parts are classified into the destination register or immediate value, the second source register/destination register, the first source register and the OpCode (type of instructions).



Instruction Memory

4-bit ALU:

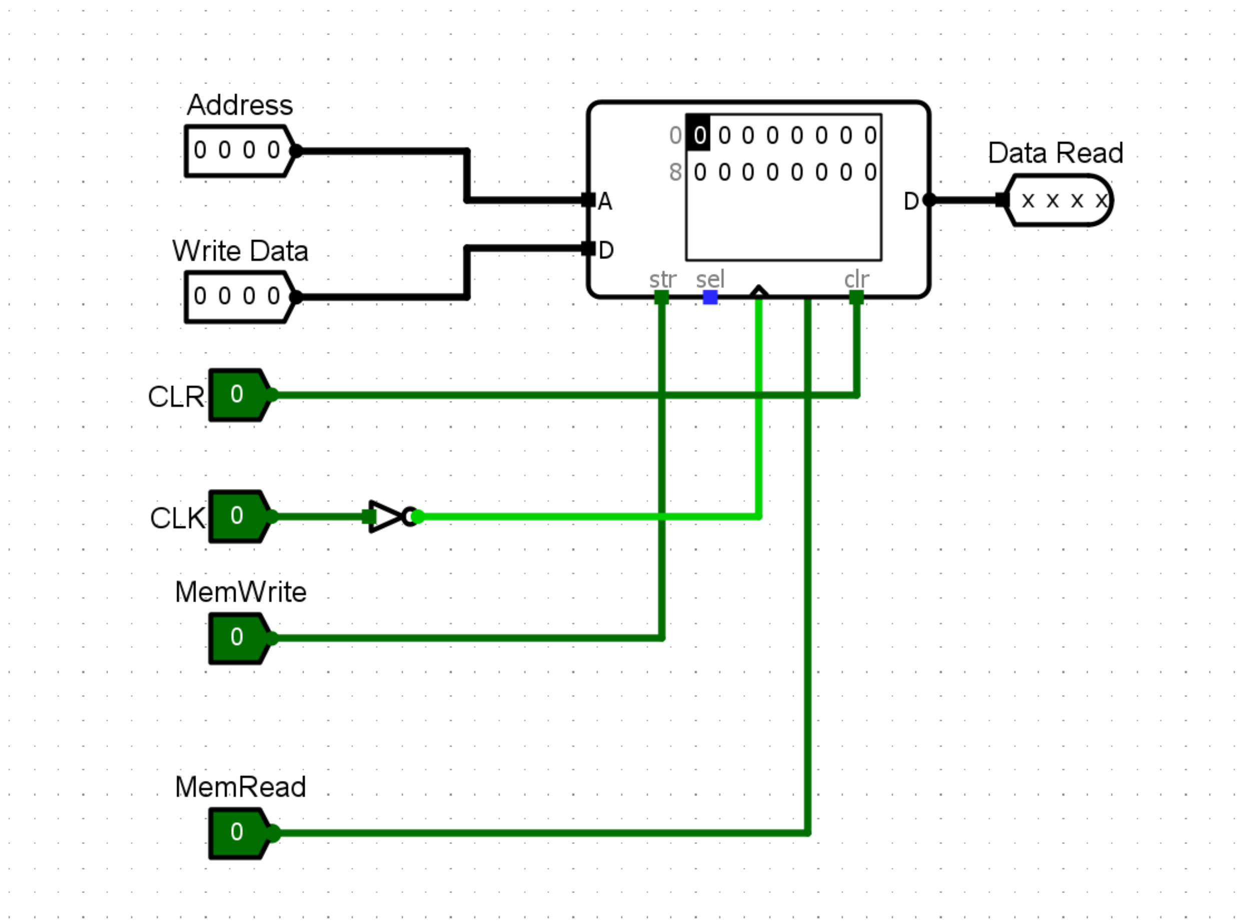
We have designed a 4-bit ALU for handling arithmetic and logical instructions. The ALU takes two 4-bit values, an ALUOp controls as inputs. It produces output values along with the zero flag to indicate whether we need to branch or not.



4-bit ALU

Data Memory:

Data memory is the storage for memory. It takes the memory address to write data in 16-bit RAM. Here, we use 2 selector bits to select our operation either to read or write. In lw instruction, there can be read from memory and in sw instruction, there can be write in memory. For memory read, we use 4-bit memory data as output and for memory write 4-bit write data as input.

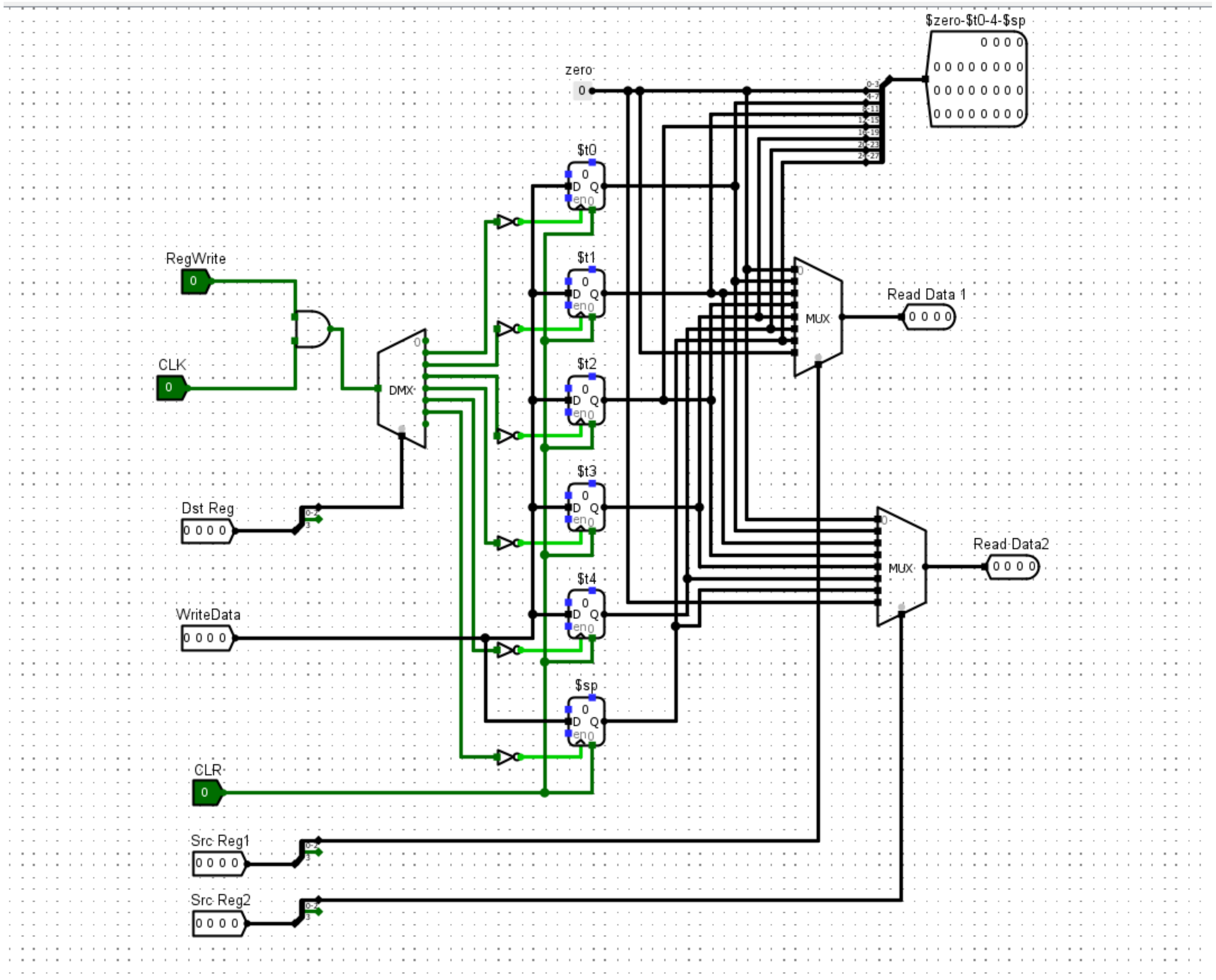


Data Memory

Register:

The processor's 6 general-purpose registers are stored in a structure called a register file. A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the computer. In addition, we will need an ALU to operate on the values read from the registers.

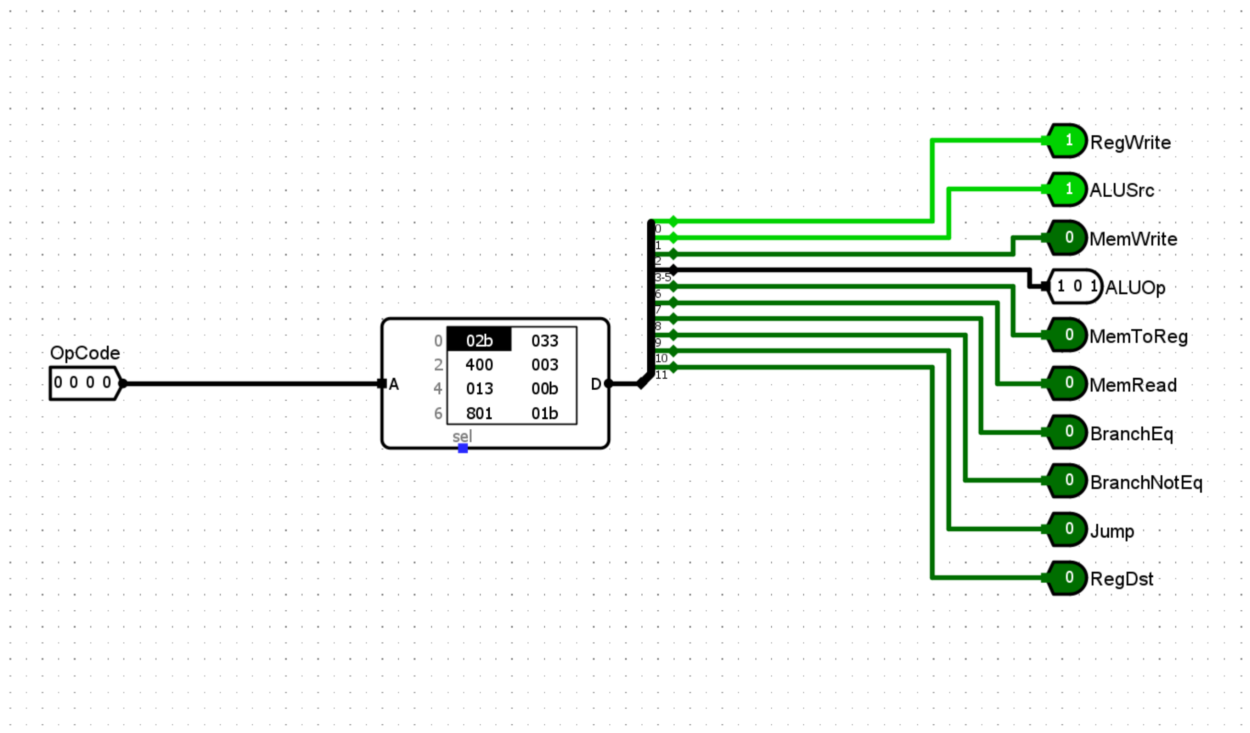
R-format instructions have three register operands, so we will need to read two 4 bit data from the register file and write one data bit into the register file for each instruction. For each data word to be read from the registers, we need an input to the register file that specifies the register number to be read and an output from the register file that will carry the value that has been read from the registers. To write a data word, we will need two inputs: one to specify the register number to be written and one to supply the data to be written into the register. The register f le always outputs the contents of whatever register numbers are on the Read register inputs. Writes, however, are controlled by the write control signal, which must be asserted for a write to occur at the clock edge.



Registers

Control Unit:

In the control unit, we produce the selector bits for our MUX operations which are based on the provided instructions. That’s why, we use a 16-bit ROM. Here we store the hexadecimal values for different operations. After receiving 4-bit OpCode, we use it to select the corresponding operation from the ROM. The ROM produces 9 selector bits and a 3-bit ALUOp for controlling operations of a 4-bit ALU.



Control Unit

Implementing PUSH & POP Instruction:

push and pop instructions are converted into several MIPS instructions (lw, sw, addi, subi).

For push $tx type instructions, first the value of $tx register is stored in the head of the stack memory. Then the value of $sp is decreased by 1-

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

For pop $tx type instructions, the reverse of push is done. First the value of $sp register is increased by 1. Then the value is loaded from the stack into $tx register-

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

Program Execution Instructions:

- ***Generating Machine Code:***
  - Write the assembly code in assembly1.txt
  - Run the provided .c file to generate a output.hex and a output.bin file

- ***Copying Code Contents:***

  - *Software Simulation:*

    Open the provided .circ file in logisim-ITA and copy the contents of the output.hex file into ROM contents into the instruction block

  - *Hardware Execution:*

    Copy the contents of the output.hex file into the provided .c file and burn the ATMEGA32 with the instruction.hex file generated by Atmel Studio

- ***Run/Execution:***
  - *Software Simulation:*

    Go to the main circuit (main.circ) and enter clock pulse to simulate

  - *Hardware Execution:*

    Provide clock pulse with a switch to start execution

IC count:

Components	Details	Count
IC74157	Quad 2 to 1 line Mux	8
IC7483	4 bit Adder	4
IC74273	8 bit D flip-flop	1
ATMEGA32	Micro-controller	5

Simulator:

Software: Logisim-ITA

Discussion:

In this assignment, we were instructed to design, simulate and implement a modified and reduced version of the MIPS instruction set. The instructions were of 16 bit instead of 32 bit. The data address bus is of 8 bit & registers are 4 bits. The instructions are divided into four equal sections, excluding jump only. The opcode is of 4 bits instead of 6 bits. The original MIPS does not have any S-format instructions, but in our reduced MIPS, a S-format instruction was implemented, which indicates shift. Additionally, other formats of instructions were also modified. For example, in the reduced R-format, there are no bits reserved for shamt or funct. The Instruction Memory, Registers, Data Memory, Control and ALU were replaced by ATMEGA32. IC 74273 was used as the Program Counter (pc). The .c file serves as a manual assembler. Stack Pointer ($sp) was implemented as a bonus.

In the hardware implementation, care was taken to ensure that all the connections in the circuit were properly attached, through repeated continuity tests with a multimeter. During construction and testing phases, the instruments used were fortunately mostly functional, thus enabling us to build a functional MIPs with minimal debugging.

We have invested quite some time to make the design as efficient as possible. We tried to minimize the number of ICs. We have tested the given instructions as well as the corner cases. The experiment helped us better understand the overall MIPS instruction set and the data path as a whole.