Course ID: CSE 306

Course Title: Computer Architecture Sessional

Assignment 3: Assignment on 4-bit MIPS Design, Simulation and implementation

Section : A2

Group : 03

Group Members:

1) 1905034
2) 1905036
3) 1905038
4) 1905054
5) 1905059

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 unit a computer program. The CPU performs basic arithmetic, logic controlling and input/output (I/O) operations specified by the instructions in the program.

Principle 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 instruction 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 activateds 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 5 ~~seven~~ Resgisters. They are denoted as $zero, $t0, $t1, $t2, $t3, $t4, $t5, . $zero register stores 00H. Other Registers are used as General Purpose

Reginters.

3. ALU: The ALU does all the calculations. It is controlled by 3-bit ALU op 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, Reginter 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.

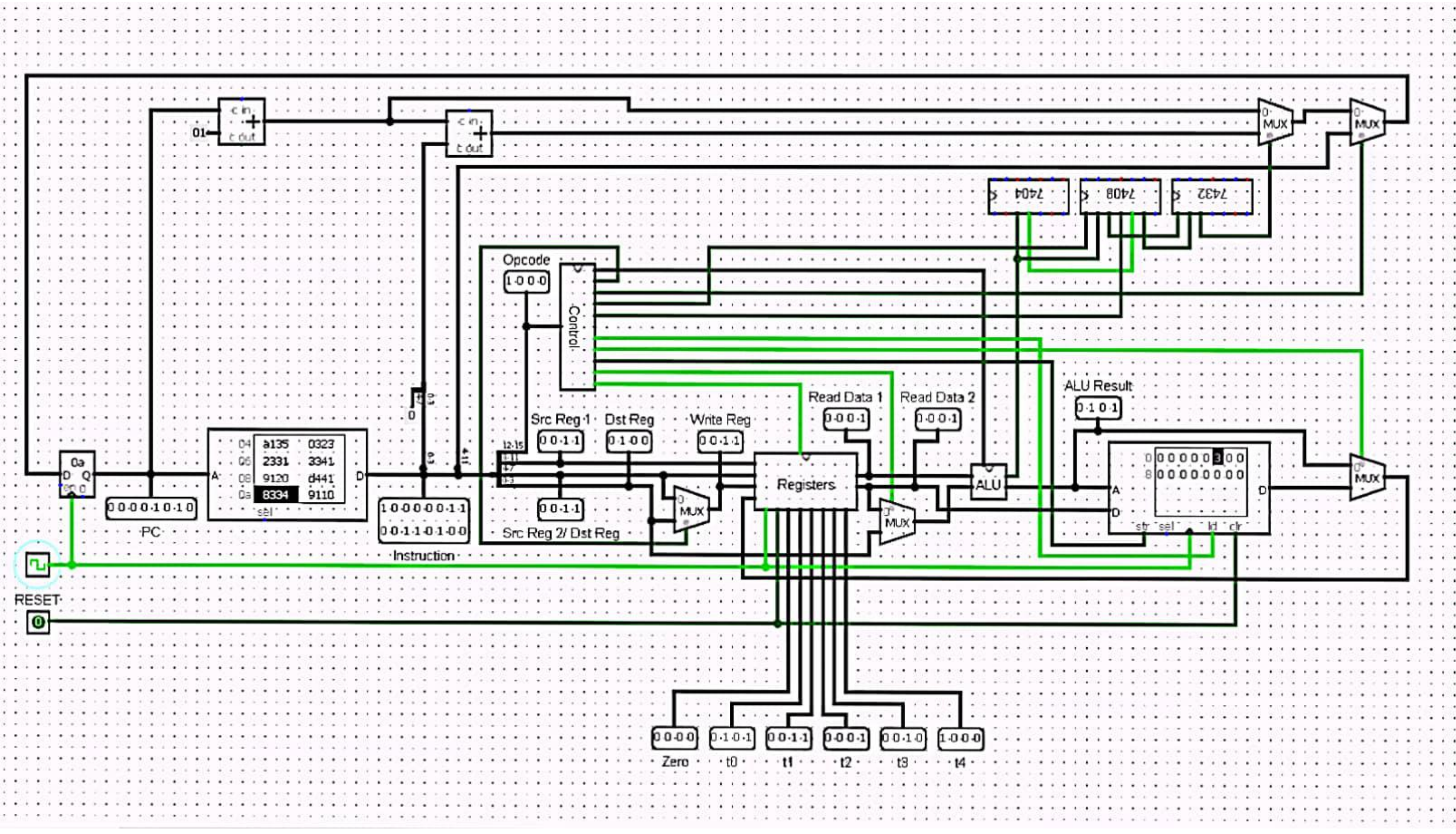
Instruction Set:

Instruction Set with Instruction ID

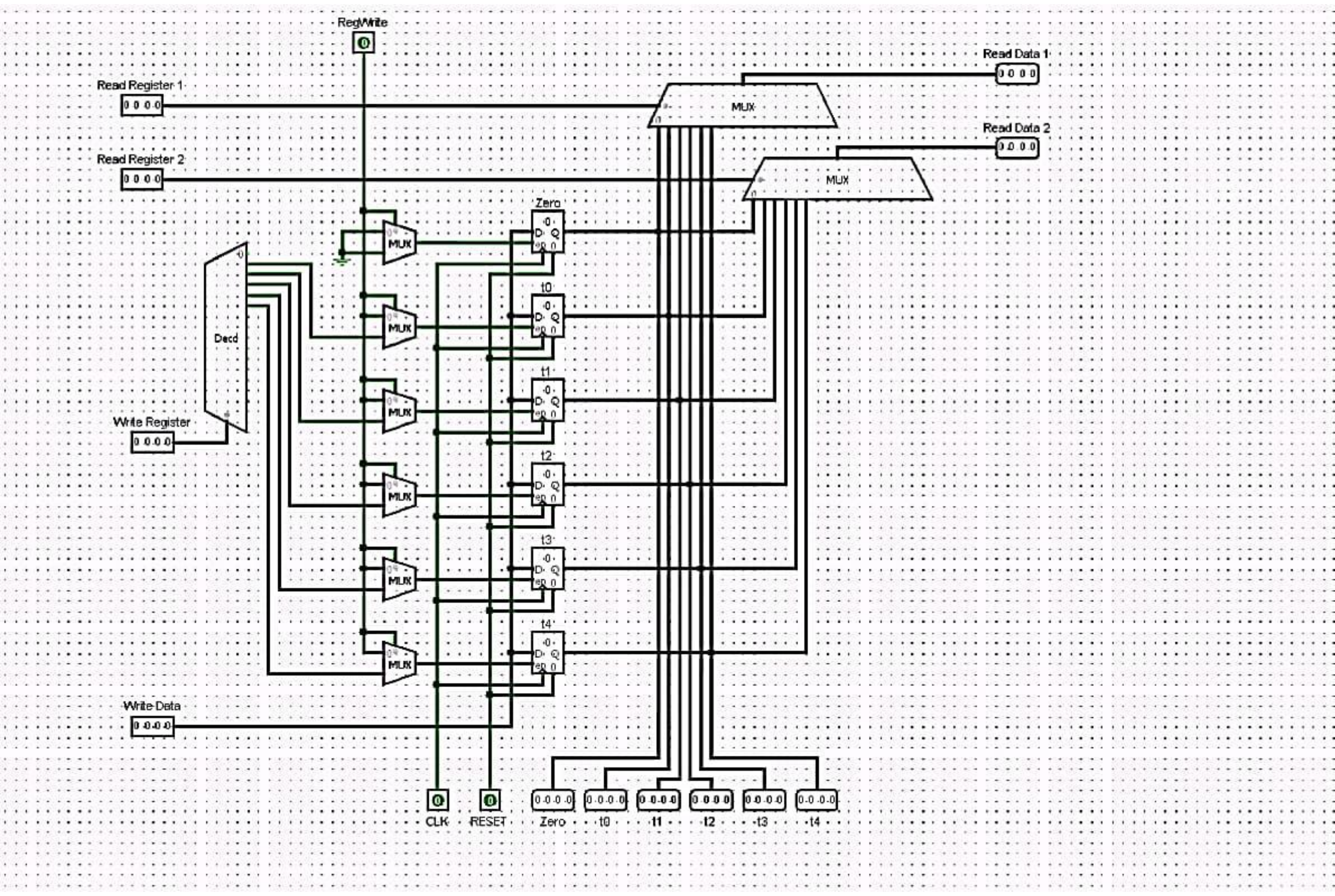
Instruction ID	Category	Type	Instruction
A	Arithmetic	R	add
B	Arithmetic	I	addi
C	Arithmetic	R	sub
D	Arithmetic	I	subi
E	Logic	R	and
F	Logic	I	andi
G	Logic	R	or
H	Logic	I	ori
I	Logic	R	sll
J	Logic	R	srl
K	Logic	R	nor
L	Memory	I	sw
M	Memory	I	lw
N	Control - conditional	I	beq
O	Control- conditional	I	bneq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

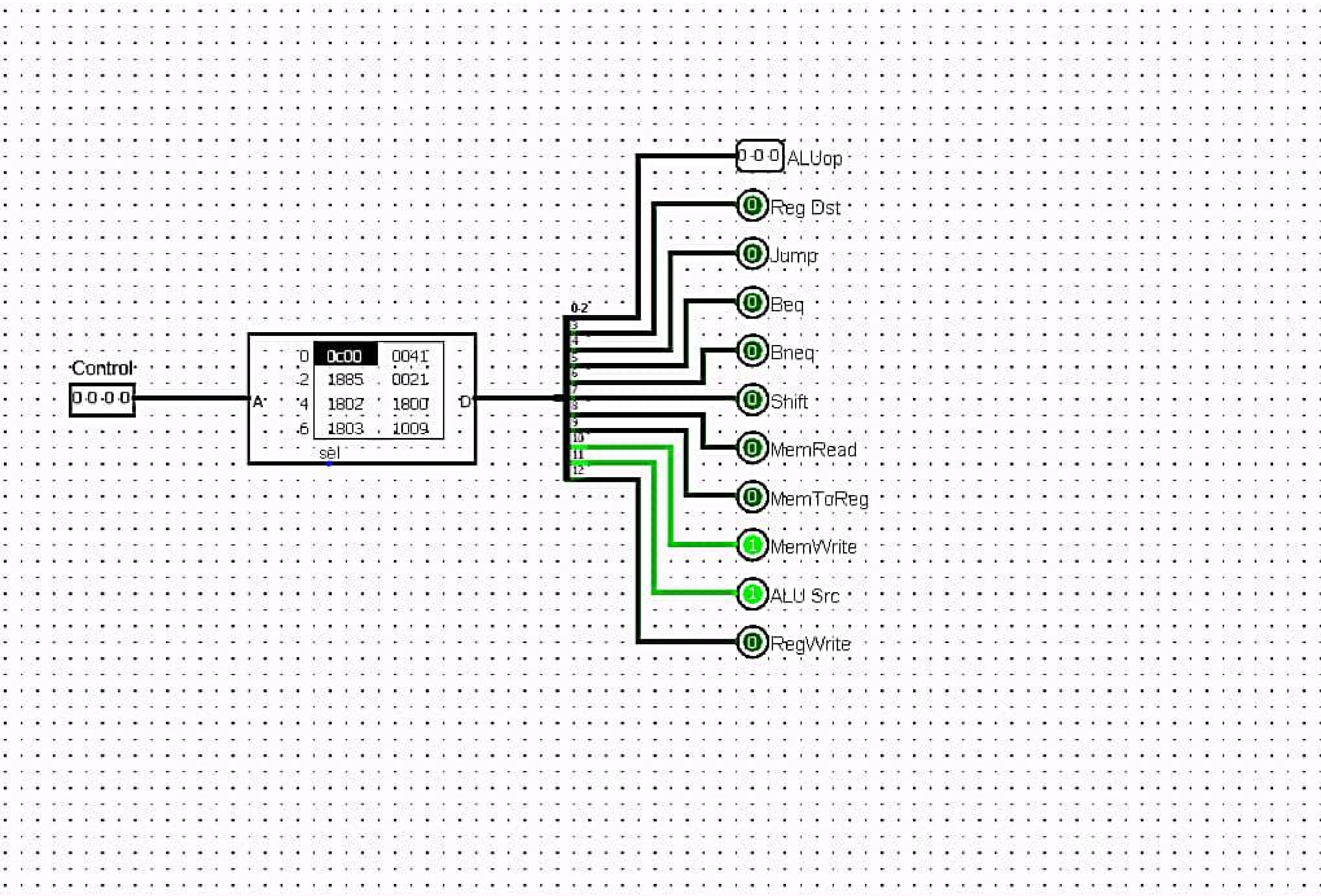
Instruction Set with Op-code:

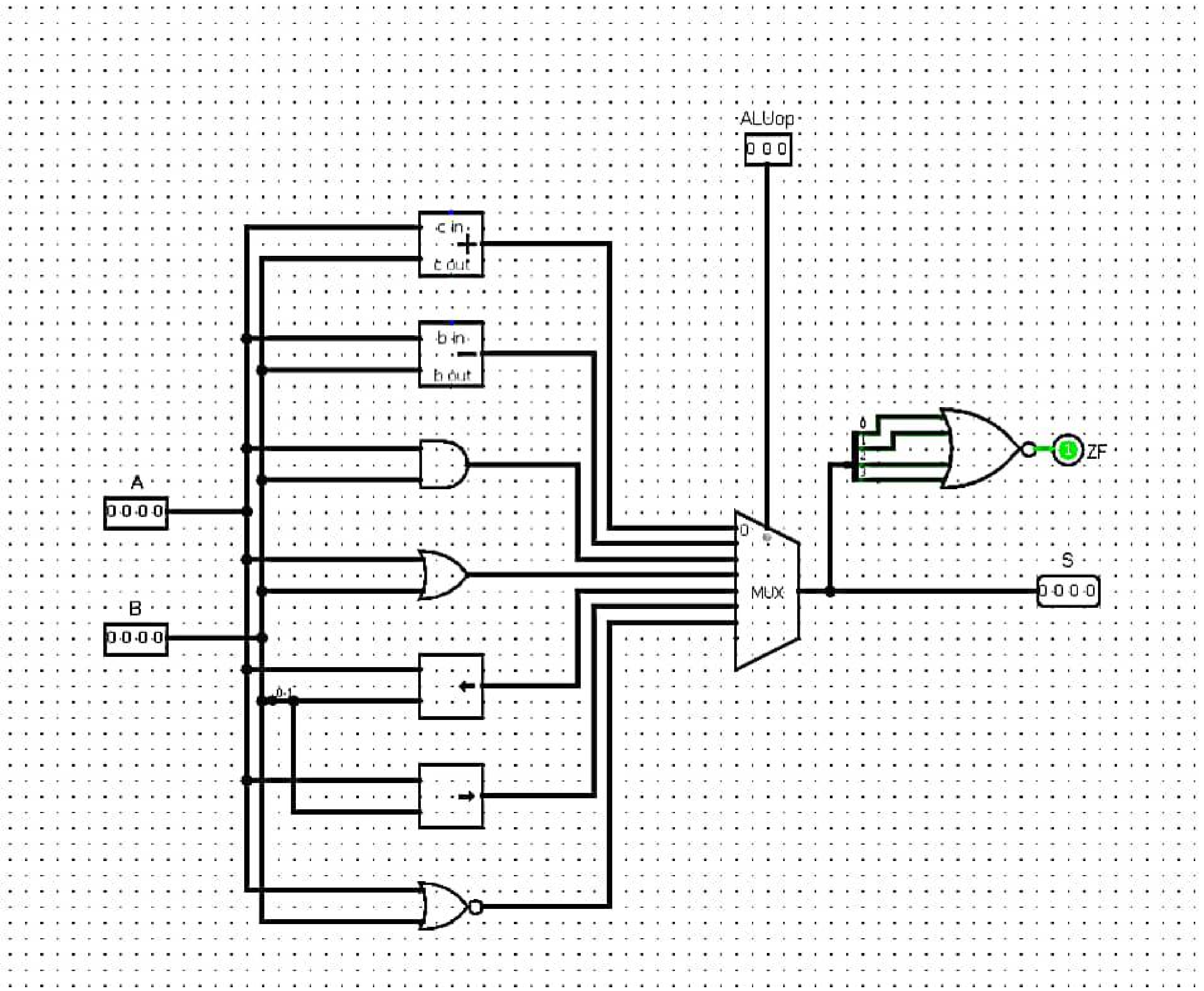
Op-code	Category	Type	Instruction
0000	Memory	I	lw
0001	Control Conditional	I	bneq
0010	Logic	R	srl
0011	Control Conditional	I	beq
0100	Logic	I	~~addi~~ andi
0101	Logic	I	addi
0110	Logic	I	ori
0111	Arithmetic	R	sub
1000	Memory	I	sw
1001	Control-unconditional	J	j
1010	Logic	R	nor
1011	Logic	R	or
1100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



01
c in
c out
MUX
7404
7408
7432
Opcode
1 0 0 0
Control
Src Reg 1
0 0 1 1
Dst Reg
0 1 0 0
Write Reg
0 0 1 1
Read Data 1
0 0 0 1
Read Data 2
0 0 0 1
ALU Result
0 1 0 1
Registers
ALU
0 0 0 1
Src Reg 2/ Dst Reg
0a
D Q
a135 0323
2331 3341
9120 d441
8334 9110
sel
PC
0 0 0 0 1 0 1 0
1 0 0 0 0 0 1 1
0 0 1 1 0 1 0 0
Instruction
RESET
0
Zero
0 0 0 0
t0
0 1 0 1
t1
0 0 1 1
t2
0 0 0 1
t3
0 0 1 0
t4
1 0 0 0
str sel ld clr


RegWrite
0
Read Register 1
0 0 0 0
Read Register 2
0 0 0 0
Read Data 1
0 0 0 0
Read Data 2
0 0 0 0
MUX
MUX
Decd
Write Register
0 0 0 0
Zero
t0
t1
t2
t3
t4
Write Data
0 0 0 0
0
CLK
0
RESET
0 0 0 0
Zero
0 0 0 0
t0
0 0 0 0
t1
0 0 0 0
t2
0 0 0 0
t3
0 0 0 0
t4


Control
0·0·0·0
A
0 0c00 0041
2 1885 0021
4 1802 1800
6 1803 1009
sel
D
0-2
3
4
5
6
7
8
9
10
11
12
0·0·0
ALUop
0
Reg Dst
0
Jump
0
Beq
0
Bneq
0
Shift
0
MemRead
0
MemToReg
1
MemWrite
1
ALU Src
0
RegWrite



ALUop
0 0 0
c in
c out
b in
b out
A
0 0 0 0
B
0 0 0 0
0-1
0
MUX
S
0 0 0 0
0
1
2
3
1
ZF

How to write and execute a program in this machine:

At first, we use a C++ program which takes the given assemble code as an input file, generate an output file which contain 16 bit hexadecimal code for each instruction. Then, we load those hexadecimal codes in our another program, then run this program and burn our Instruction Memory which is a AtMega 32 chip. Then we active PC (a D-Flip Flop) at the falling edge of a clock signal. The output of PC goes to Instruction Memory and fetch 16 bit code from the specific address. Then the datapath goes through Register, Control Unit, ALU, Data Memory and execute the instruction. So, only one instruction executes at every clock. After every clock, PC adds 1 to its previous address. However, we see the output of different components in our hand made circuit which builds with LED light, breadboard and jumper wires. Thus, this MIPS executes Instructions in this way.

ICs used with count as a chart

Gate	Count
At Mega 32 A	5
ic 74157	9
ic 7483	9
ic 74273	1

Simulator: Logisim Version 2.7.1

Discussion:

While implementing the circuit, we had to change our design several times. Some designs requires a lot of ICs. To optimize number of ICs in our design, we had to discard those. Sometimes, our wire connection were loosed, we did wrong connection at Atmega32, we forgot to connect the strobe of ~~8~~ 4-bit MUX with ground, sometimes, we forgot to connect ~~at~~ all ground pin with ~~gro~~ common ground bus. So, for our mistake, we needed to invest more extra time to solve problems. Besides, 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.