North South University

Department of Electrical & Computer Engineering

Project part-2 Paper

Course Code: **CSE332**

Course Name: **Computer Organization & Architecture Lab**.

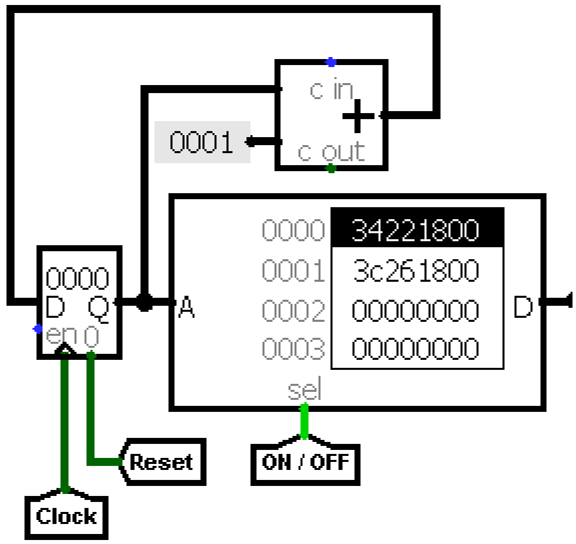
Name: **MD. Muntasir Ahmed**

ID: **1921333042**

Section: **07**

Submission Date: **08-12-2021**

Instruction Fetch: In order to construct instruction-fetch unit, I used a ROM. Its address bit is 16 and data bits is 32. It is also connected with a 16 bit register which actually works as a program counter. It means it counts the number of instructions performed in each clock cycle.



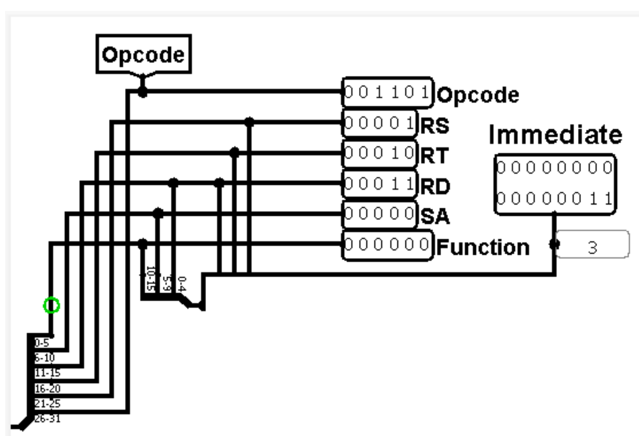
Instruction Decode: In this part of the circuit, it decodes the 32-bit instruction into R-format or I-format for a specific opcode. It also generates the values for registers that will be used in ALU operations.

R-format

Opcode (26-31)	RS (21-25)	RT (16-20)	RD (11-15)	SA (6-10)	Function (0-5)

I-format

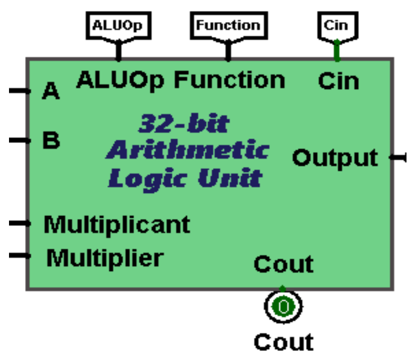
Opcode (26-31)	RS (21-25)	(RT) (16-20)	Immediate Value (0-15)



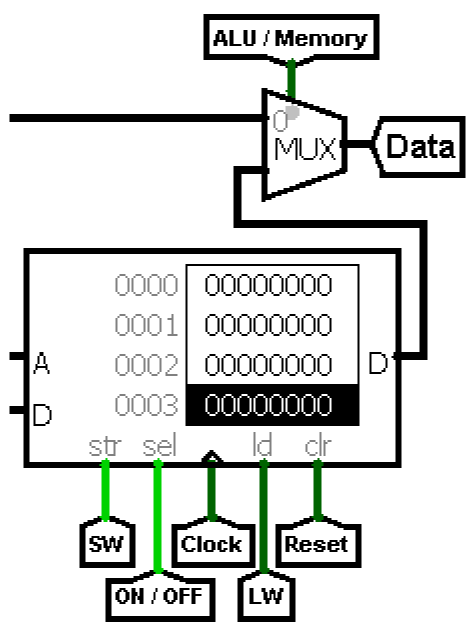
<u>Operand Fetch</u>: In this part the circuit determines the registers that has been selected through the decoded 32-bit instruction. The register will be used in ALU operation. Then the values of that registers have been passed to the ALU. The ALU operation will be determined by the opcode which is the only input of the Main Control Unit.

<u>Execute</u>: Mainly two sub circuits perform most of the execution. One is the ALU another one is the Mani Control Unit.

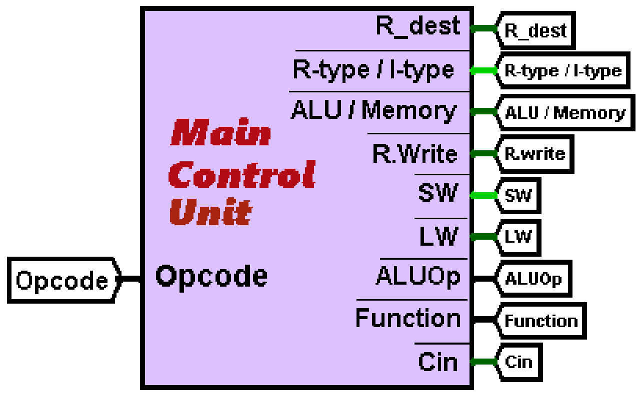
- **ALU:** ALU performs the arithmetic and logical operations. In R-format, it performs XOR, NOR, NAND, ADD, SUB and Mul. In I-format it performs Addi, Subi, Muli, Lw and Sw. While executing load and store operation ALU gives an effective address for memory to store value from register and to load value from memory back to register.



This is the memory that stores and load values.



➢ **Main Control Unit:** It is the circuit which controls the signal flow in the whole circuit. It generates proper signals for ALU, Register, Memory. It also creates signals for the three MUX that determines operation type.



This is the table for the Main Control Unit. I used programmable logic array to find the equation for each signal. I created the meanterms of Opcodes with programmable AND gate. Then I used programmable OR gate to find equation for each signal.

Instructions	Opcode	R_dest	R-type/ I-type	ALU/ Memory	R.Write	SW	LW	ALUop	Function	Cin
NAND	000011	1	0	0	1	0	0	10	10	0
NOR	001000	1	0	0	1	0	0	00	11	0
XOR	000100	1	0	0	1	0	0	11	01	0
Add	000000	1	0	0	1	0	0	11	10	0
Sub	000001	1	0	0	1	0	0	00	01	1
Addi	000110	0	1	0	1	0	0	11	10	0
Subi	000111	0	1	0	1	0	0	00	01	1
Mul	001011	1	0	0	1	0	0	11	11	0
Muli	001100	0	1	0	1	0	0	11	11	0
Sw	001101	0	1	0	0	1	0	11	10	0
Lw	001111	0	1	1	1	0	1	11	10	0

R_dest = $Op_5Op_4Op_3Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2'Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0' + Op_5Op_4Op_3'Op_2Op_1'Op_0'$

R-type / I-type = $Op_5Op_4Op_3Op_2'Op_1'Op_0 + Op_5Op_4Op_3Op_2'Op_1'Op_0' + Op_5Op_4Op_3'Op_2'Op_1Op_0 + Op_5Op_4Op_3'Op_2'Op_1Op_0' + Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

ALU/Memory = $Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

R.write = $Op_5Op_4Op_3Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2'Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0' + Op_5Op_4Op_3Op_2'Op_1'Op_0 + Op_5Op_4Op_3Op_2'Op_1'Op_0' + Op_5Op_4Op_3'Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2'Op_1Op_0 + Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

SW = $Op_5Op_4Op_3'Op_2'Op_1Op_0'$

LW = $Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

O_1 = $Op_5Op_4Op_3Op_2Op_1'Op_0' + Op_5Op_4Op_3Op_2'Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2'Op_1'Op_0 + Op_5Op_4Op_3'Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2'Op_1Op_0 + Op_5Op_4Op_3'Op_2'Op_1Op_0' + Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

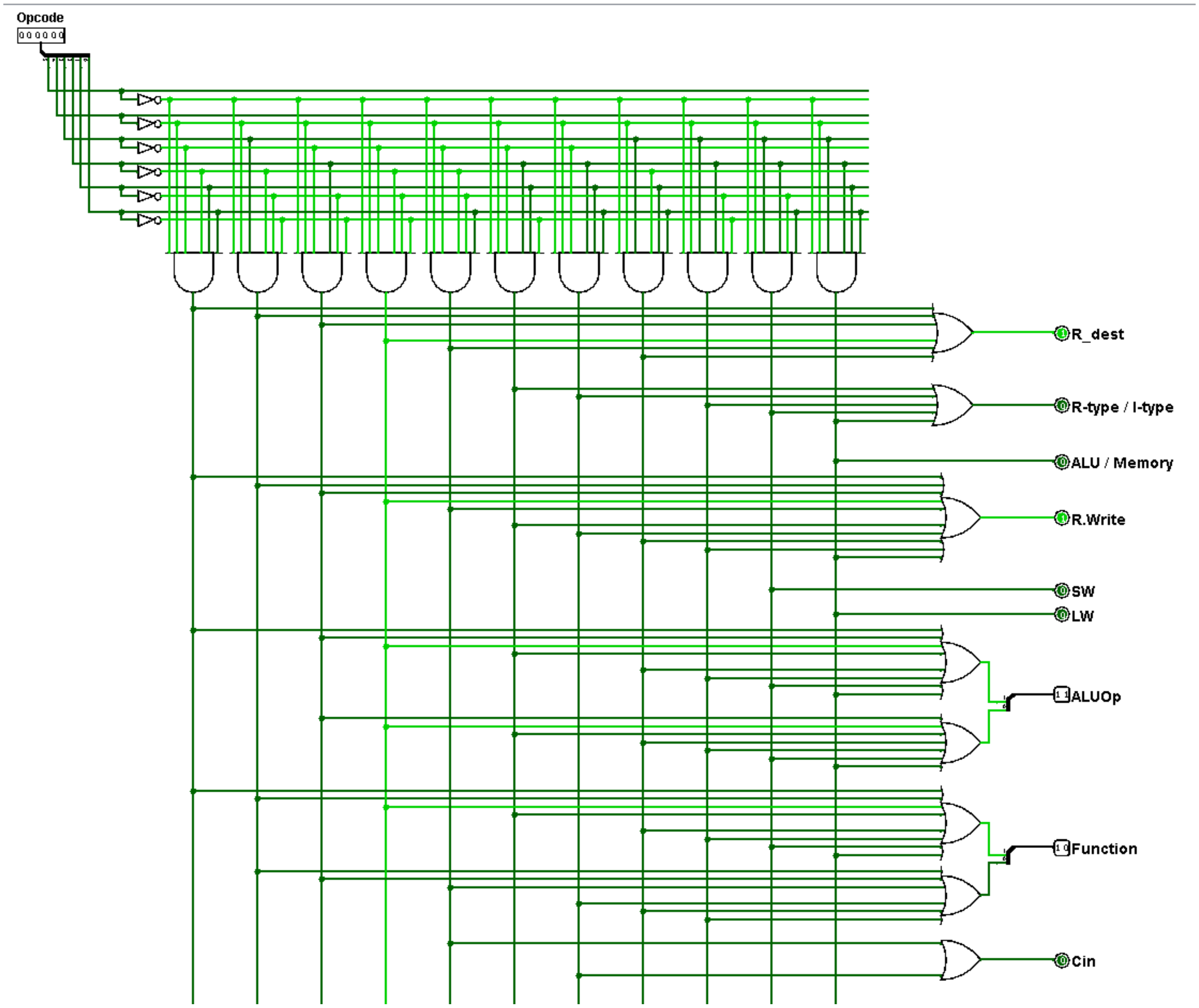
O_0 = $Op_5Op_4Op_3Op_2'Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2'Op_1'Op_0 + Op_5Op_4Op_3'Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2'Op_1Op_0 + Op_5Op_4Op_3'Op_2'Op_1Op_0' + Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

$\mathbf{F_1} = Op_5Op_4Op_3Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2'Op_1'Op_0 + Op_5Op_4Op_3'Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2'Op_1Op_0 + Op_5Op_4Op_3'Op_2'Op_1Op_0' + Op_5Op_4Op_3'Op_2'Op_1'Op_0'$

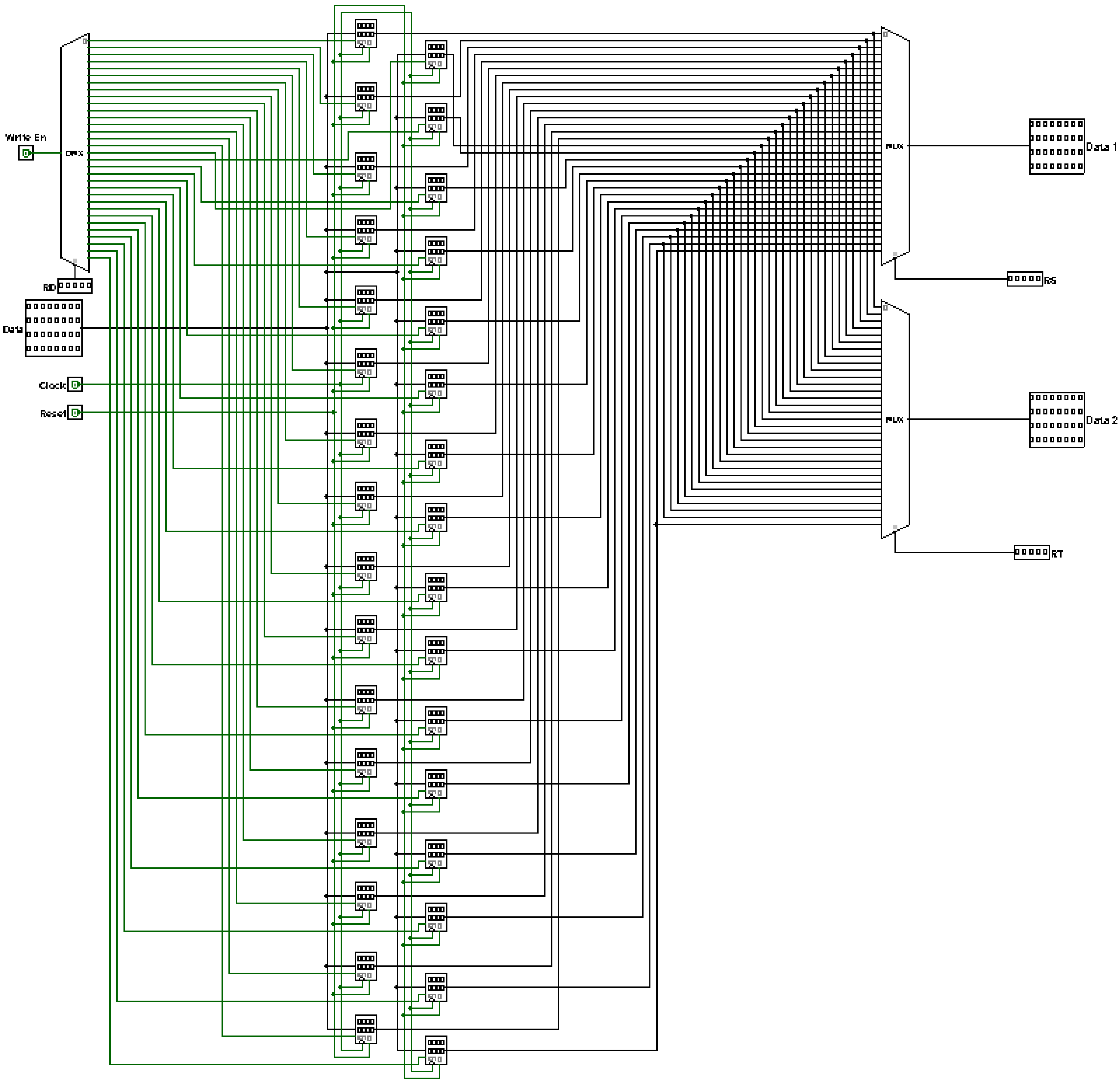
$\mathbf{F_0} = Op_5Op_4Op_3'Op_2Op_1Op_0 + Op_5Op_4Op_3Op_2'Op_1Op_0 + Op_5Op_4Op_3Op_2Op_1Op_0' + Op_5Op_4Op_3Op_2'Op_1'Op_0' + Op_5Op_4Op_3'Op_2Op_1'Op_0' + Op_5Op_4Op_3'Op_2'Op_1Op_0$

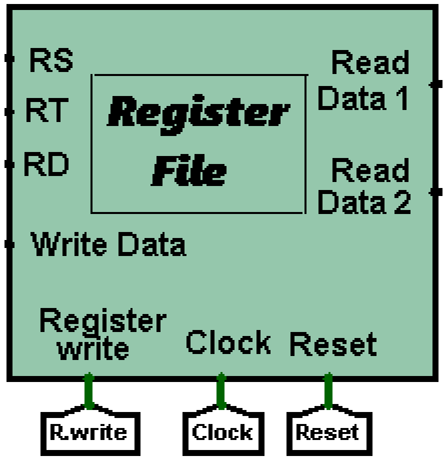
$\mathbf{Cin} = Op_5Op_4Op_3Op_2Op_1Op_0' + Op_5Op_4Op_3Op_2'Op_1'Op_0'$

Circuit of the Main Control Unit.



<u>Result Store</u>: When the result is concerned there are two results has been generated, one by ALU another by Memory. These two results are connected to a MUX who's control signal comes from the Main Control Unit. The output of the MUX is connected to Write Data pin of the Register Circuit. After a clock cycle the expected result will be stored in the desired register.





Next Instruction: After every clock cycle, the program counter changes the address for the ROM to follow the next instruction.

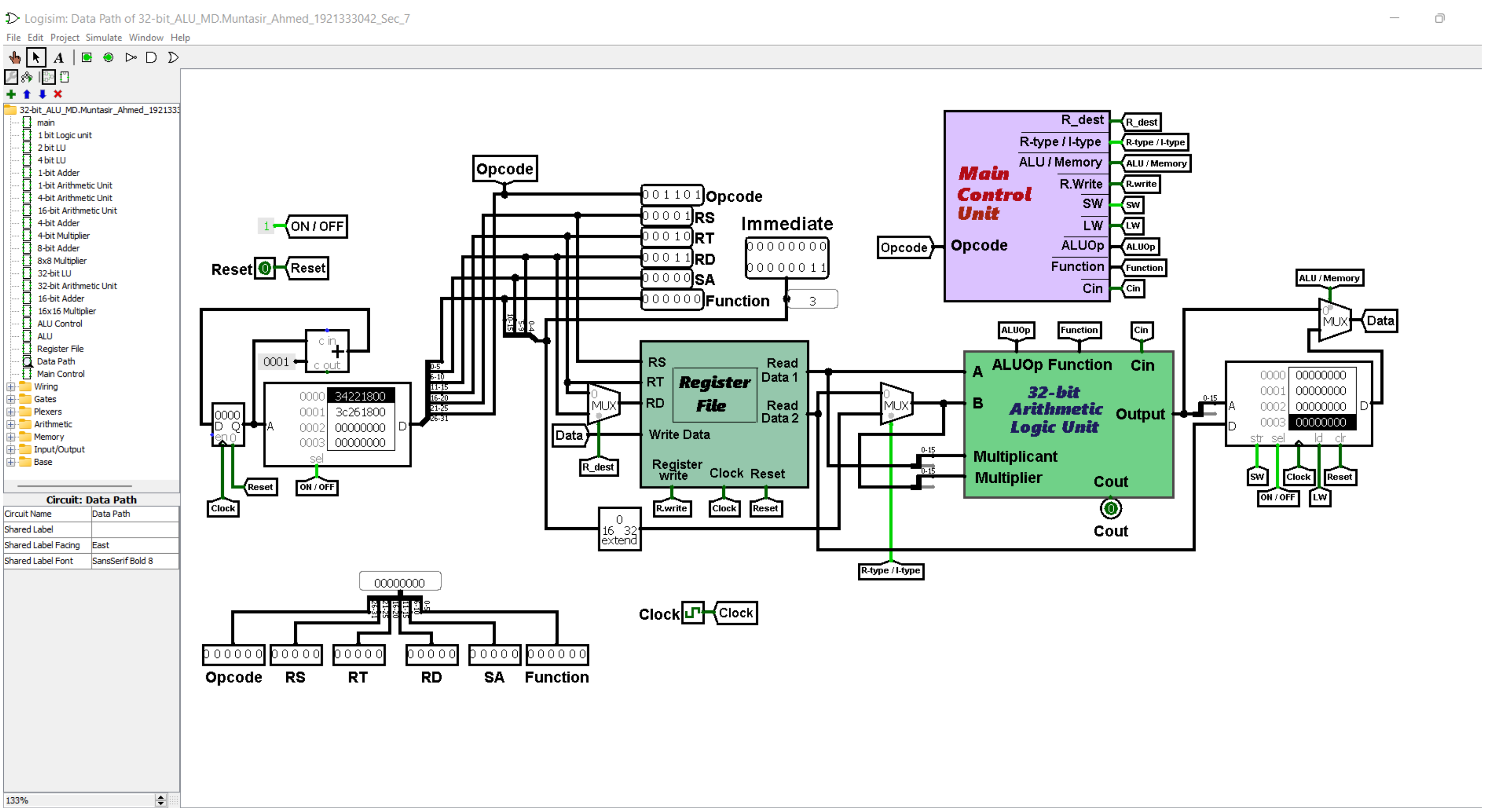


Fig: Complete picture of the circuit.

Discussion: In project part-2 we were asked to design a single cycle CPU which will contain an ISA format Register File, ALU, Memory, ROM, Instruction-fetch and Main Control unit.

In lab 5 I learned how register file works. In lab experiment I designed a 16-bit Register File successfully. When I gained the complete idea of a Register File, I started constructing the 32-bit Register File for project. In lab 6, I learned how to implement a single cycle data path. The lab instructor gave us a brief explanation about instruction format and how the datapath circuit will change according to the instruction format. I also learned a new thing, which is ROM. It is used to pass instructions. I implemented R-format, I-format and load-store datapath successfully in lab 6. The next thing I had to do is to combine the separately designed datapath circuits using MUX. In lab 7, I learned the process of combining the datapath circuits and I did that successfully to my 16-bit single cycle datapath. As the whole concept of datapath is clear to me so, I started to construct datapath for my 32-bit single cycle CUP. I also designed a main control unit which will take 6-bit Opcode as input and will generate accurate control signals for Register File, Memory and the three MUX, which determines the instruction type and operation.

While designing the circuits I faced some problems with the data bits passing from register to ALU and ALU to Memory. For multiplier the two inputs ware in 16-bit but the data from register was 32-bit. The same thing happens reversely for passing data from ALU to Memory. The effective address calculated by the ALU is in 32-bit but Address bit of Memory is 16-bit. I shared this concern with my lab instructor and he advised me to use splitter or bit-extender to fix this issue. I used splitter to divide bits. Now my circuit was ready for simulation test.

For testing the circuit, first I wrote some instructions in MIPS. Then I converted the instruction in HEX to put it in the ROM. After putting proper instruction in the ROM, I gave a clock and the circuit operated as I intended. It is executing all the operations successfully.