COL216 Lab2 : Single Cycle Processor

Poojan Shah 2022CS11594 Pritesh Mehta 2022CS11916

We briefly describe our implementation, design choices and the subsequent testing which was done for the complete single cycle datapath and control unit.

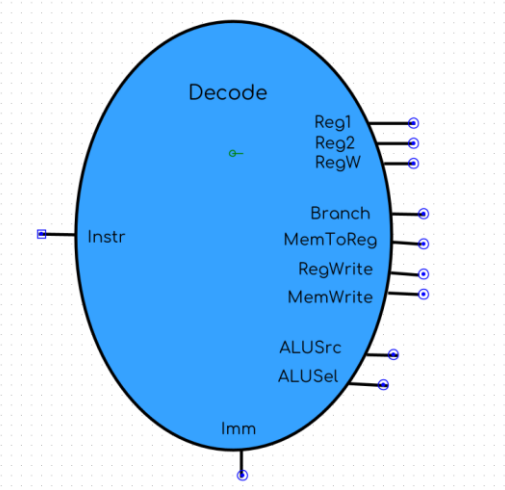
Designing the Decode Module :

Since we are using a single cycle implementation, the decode unit does the job of the control, and is purely combinational in nature. Based on the instruction, we just need to output the control signals. Here is the RISC-V instruction format for reference :

CORE INSTRUCTION FORMATS

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Here is the interface for the decode module :



By observing and understanding the format of the RISC-V ISA, we can derive the following expressions for combinational logic of the control signals. We just have to differentiating opcodes/func3/func7 and utilize don't cares carefully to obtain the following :

$$Reg1 = I[19:15]$$

$$Reg2 = I[24:20]$$

$$RegW = I[11:7]$$

$$MemToReg = I[4]$$

$$RegWrite = I[4] \,\&\&\, !\, I[5]$$

$$MemWrite = !\, I[4] \;\&\&\; I[5] \;\&\&\; !\, I[6]$$

$$ALUSrc = !\,(I[4] \,\&\&\, I[5] \,|\;\; |\, I[5] \,\&\&\, I[6] \,|\;\; |\, I[6] \,\&\&\, I[4])$$

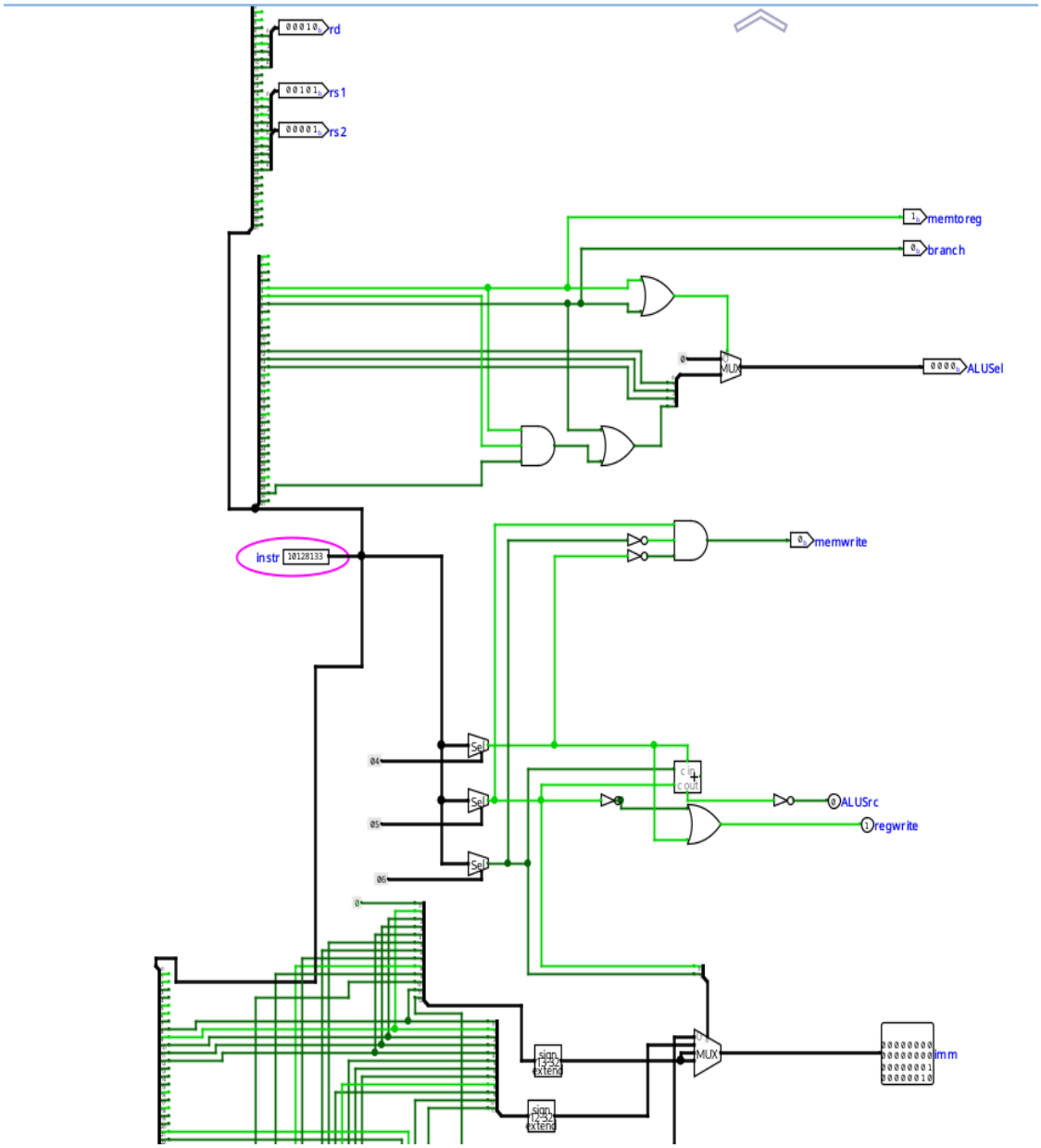
$$ALUSel = if\ (I[4] \,|\;\; |\, I[6])\ then\ 0000\ else\ (I[4] \&\& I[5] \&\& I[30] \,|\;\; |\, I[6]) I[13:11]$$

Similarly, we implemented Imm using a MUX.

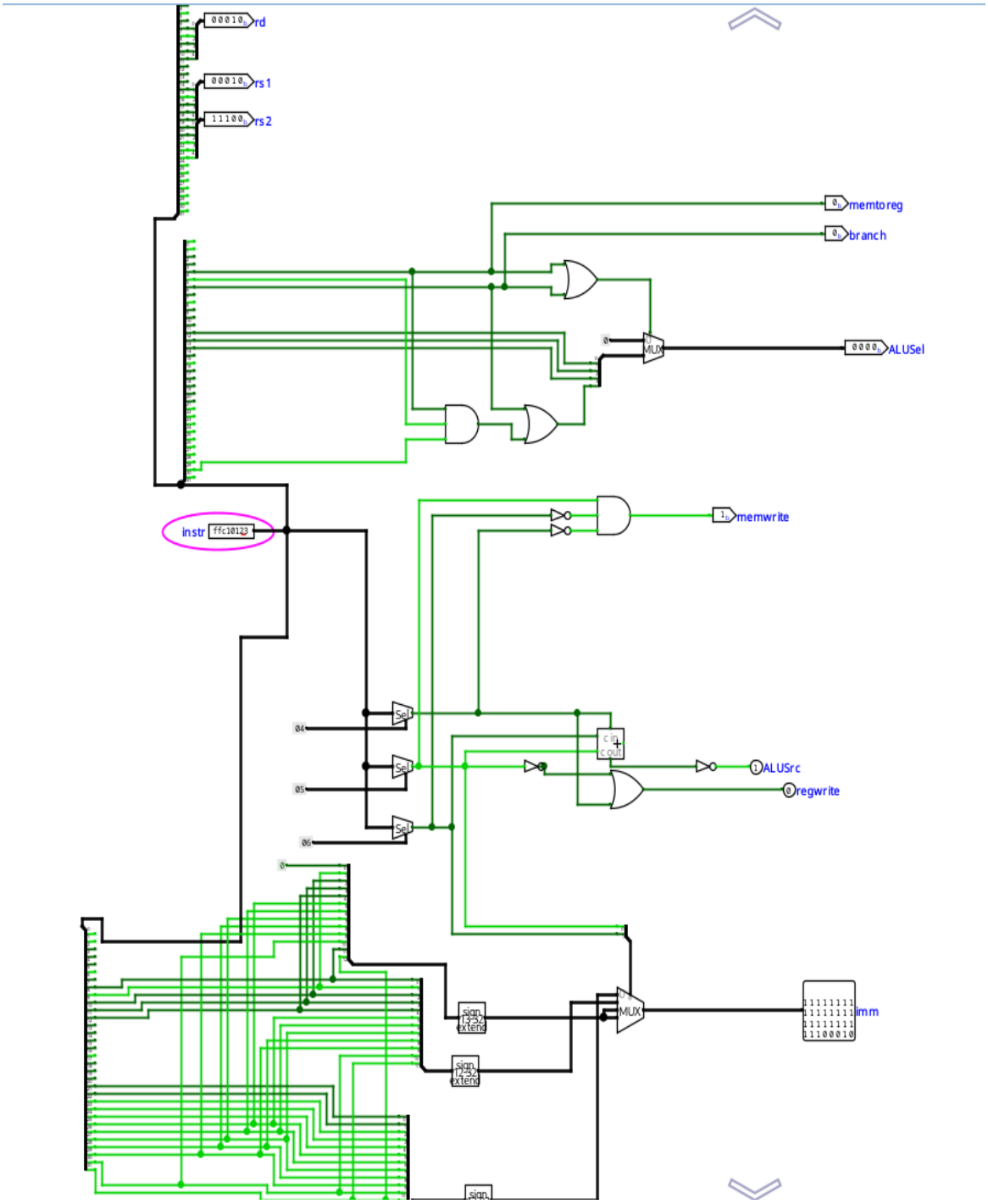
Testing the Decode Module :

For the testing, let us use one instruction of each type :

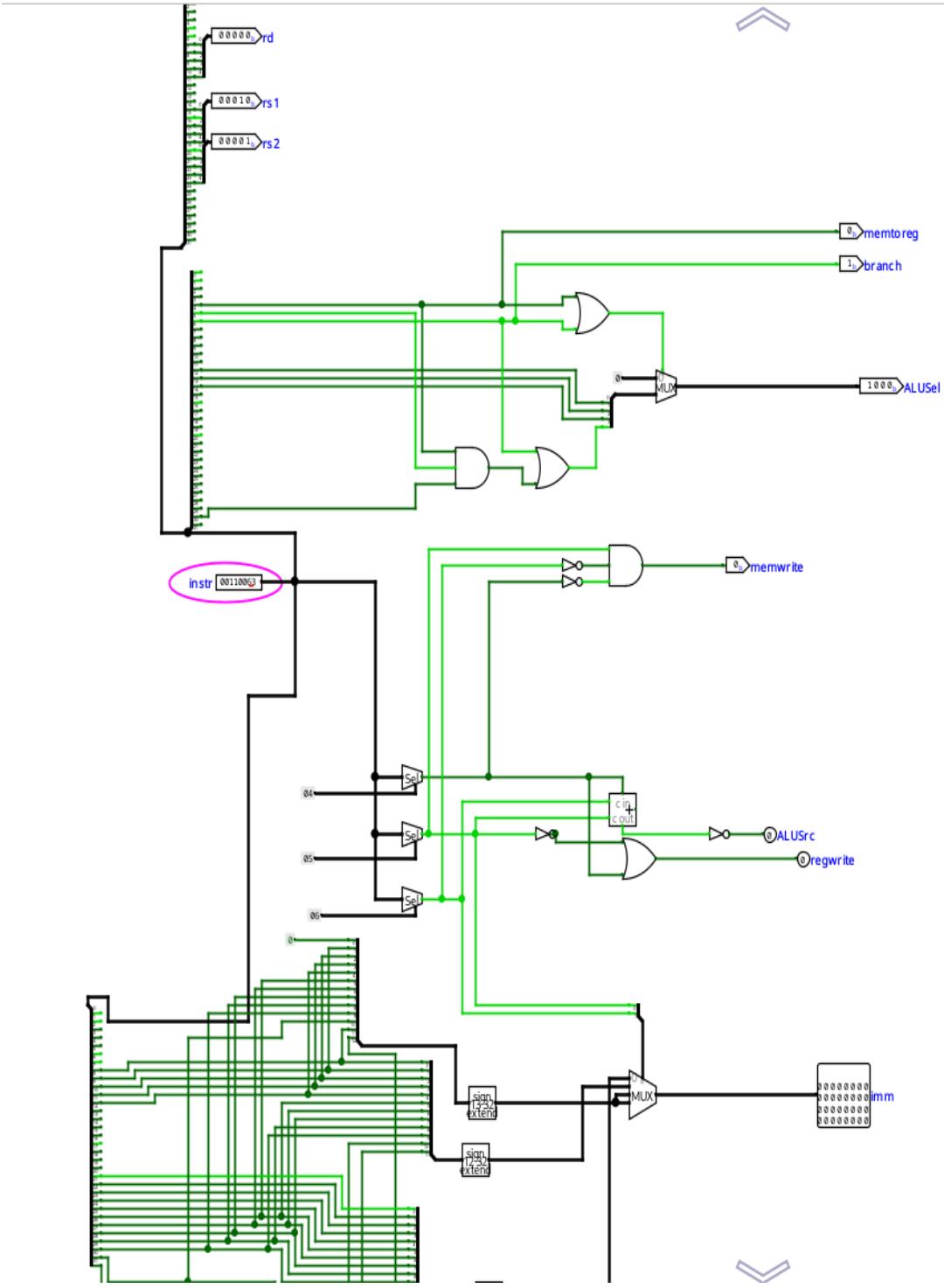
add sp,t0,ra = 00128133



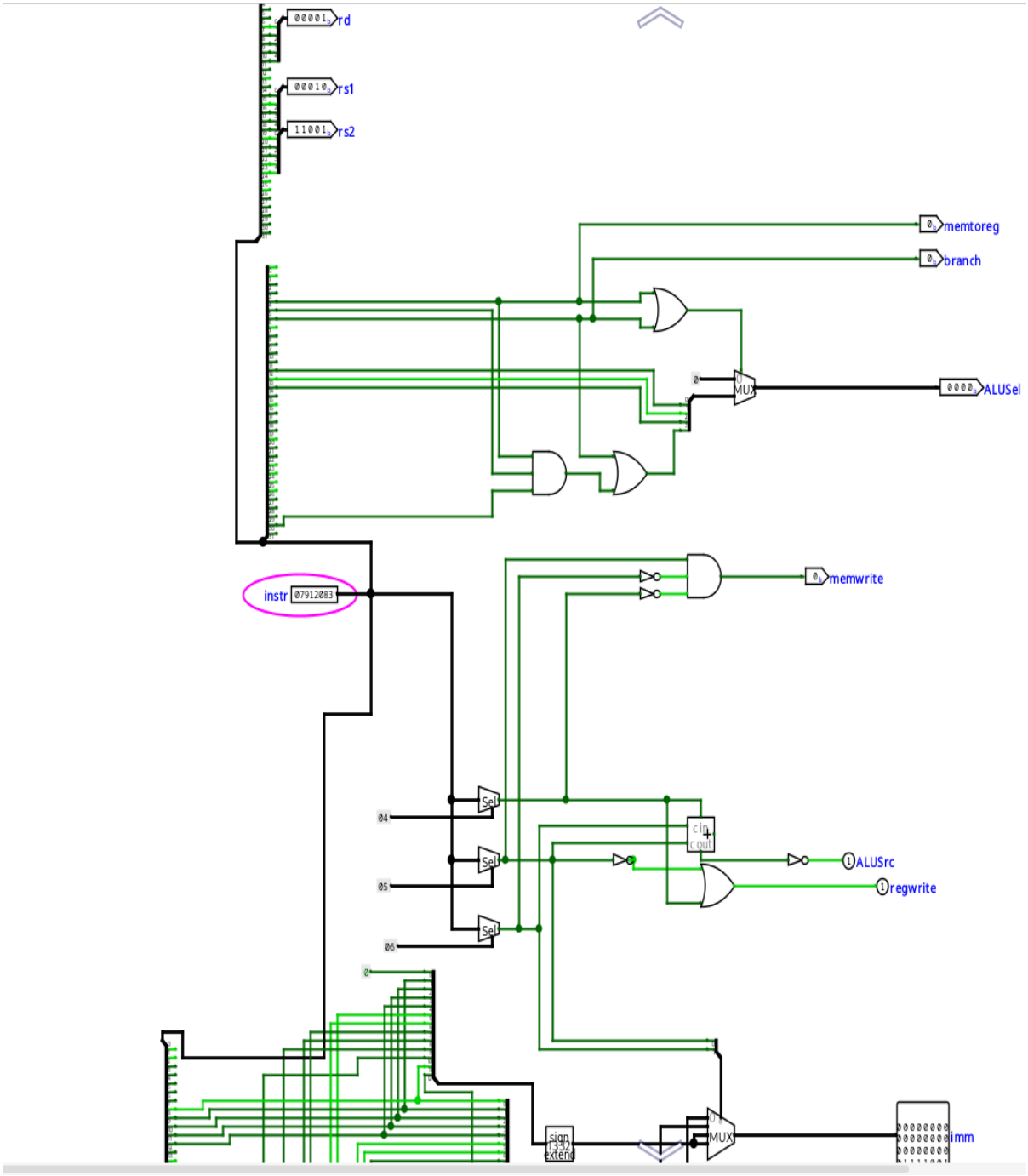
Addi sp,sp,-4 : ffc10113



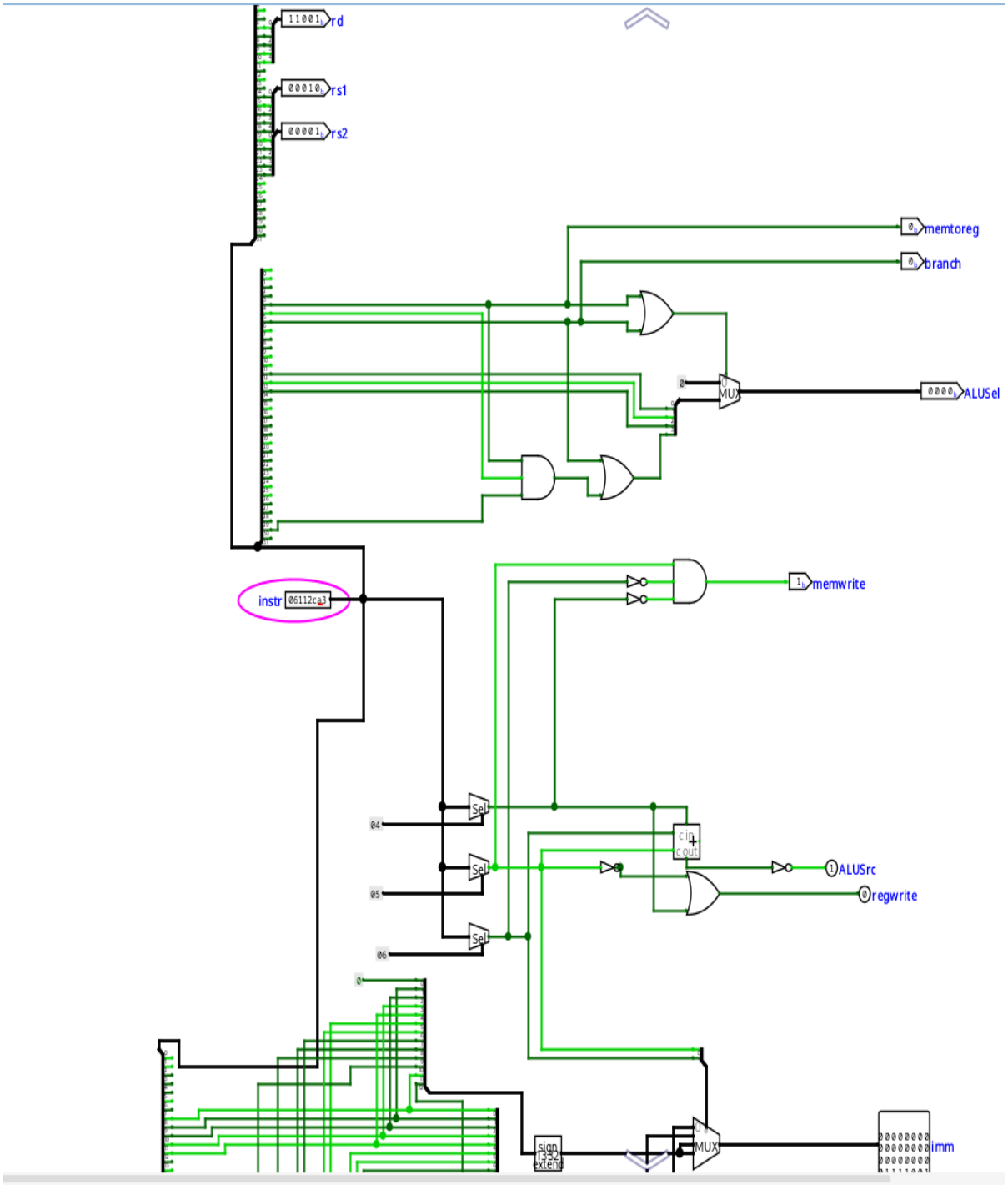
beq sp,ra,start : 00110063



lw ra,121(sp) : 07912083

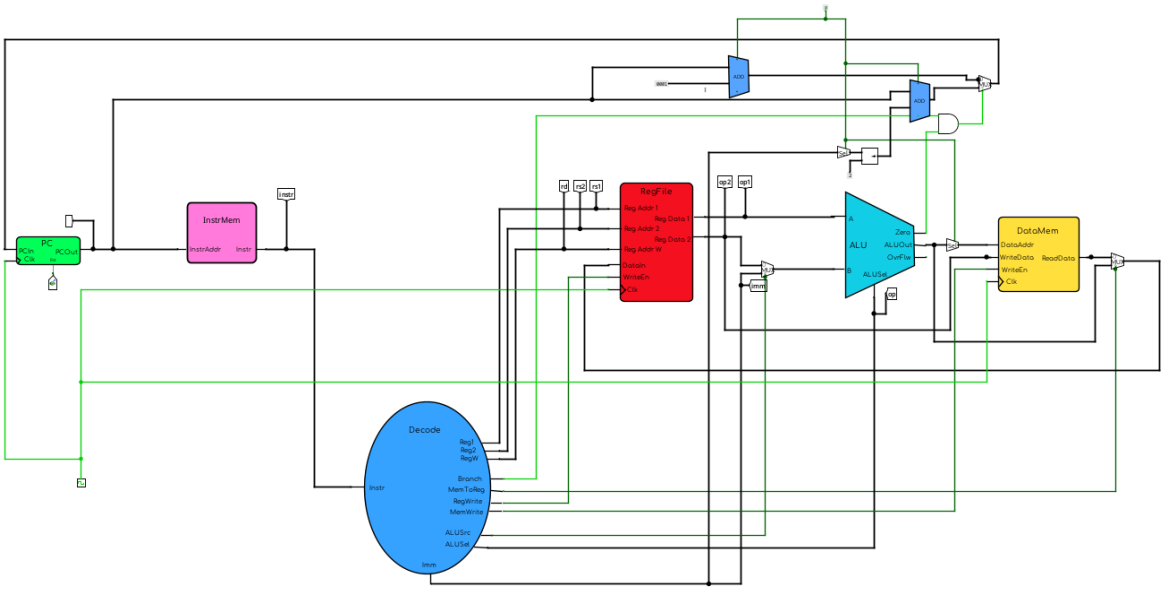


sw ra,121(sp) : 06112ca3

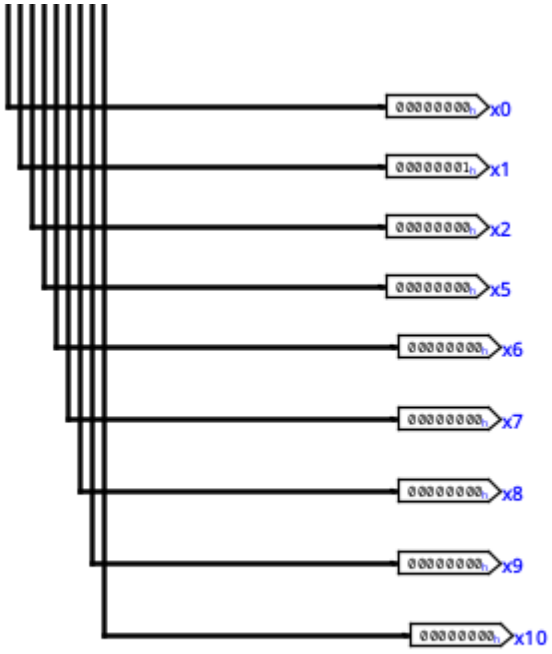


Testing the Complete Design :

Here is the top level design :



To test the design, We need to first assemble the RISC-V code using an online assembler and then store it in the InstrMem and use the inbuilt clock ticking to observe the results. Logisim-evolution also provides VHDL-like waveforms for observation. We provide the test signals as output to the processor. For example, we show the contents of the registers at all the times :



Testing Using Simple Programs :

The following code was used to check R type Instructions :

$$addi\ x1, x0, 3$$

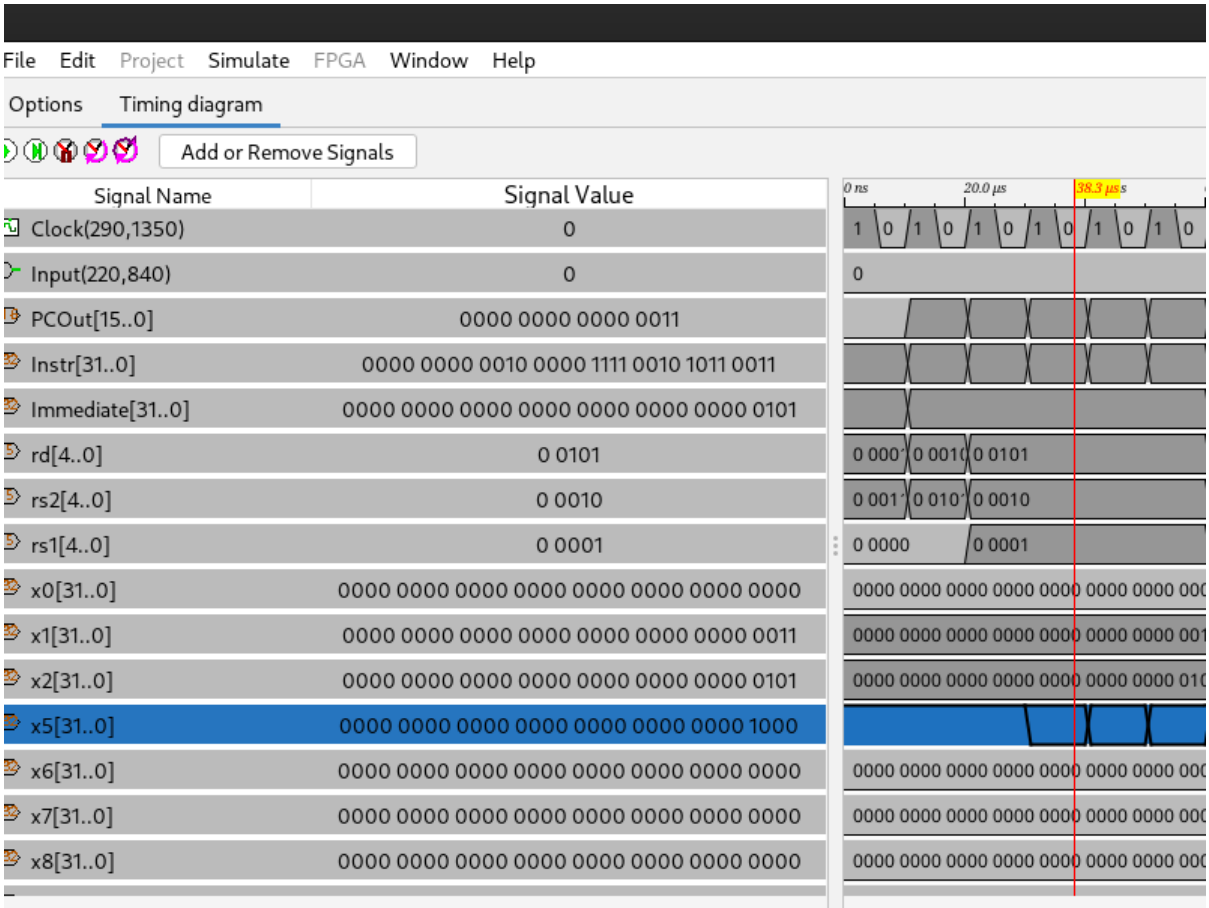
$$addi\ x2, x0, 5$$

$$add\ x5, x1, x2$$

$$and\ x5, x1, x2$$

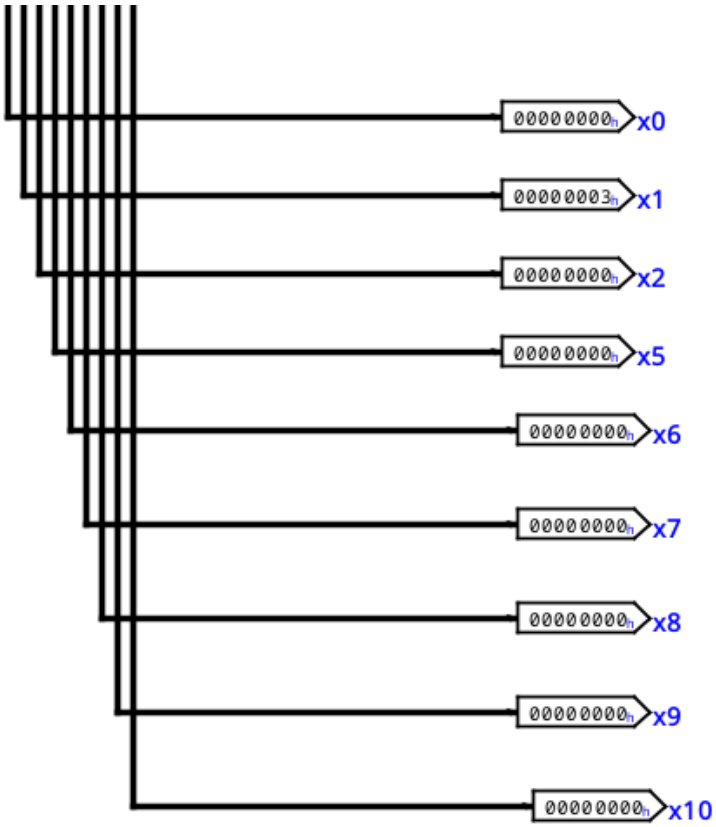
$$slt\ x5, x1, x2$$

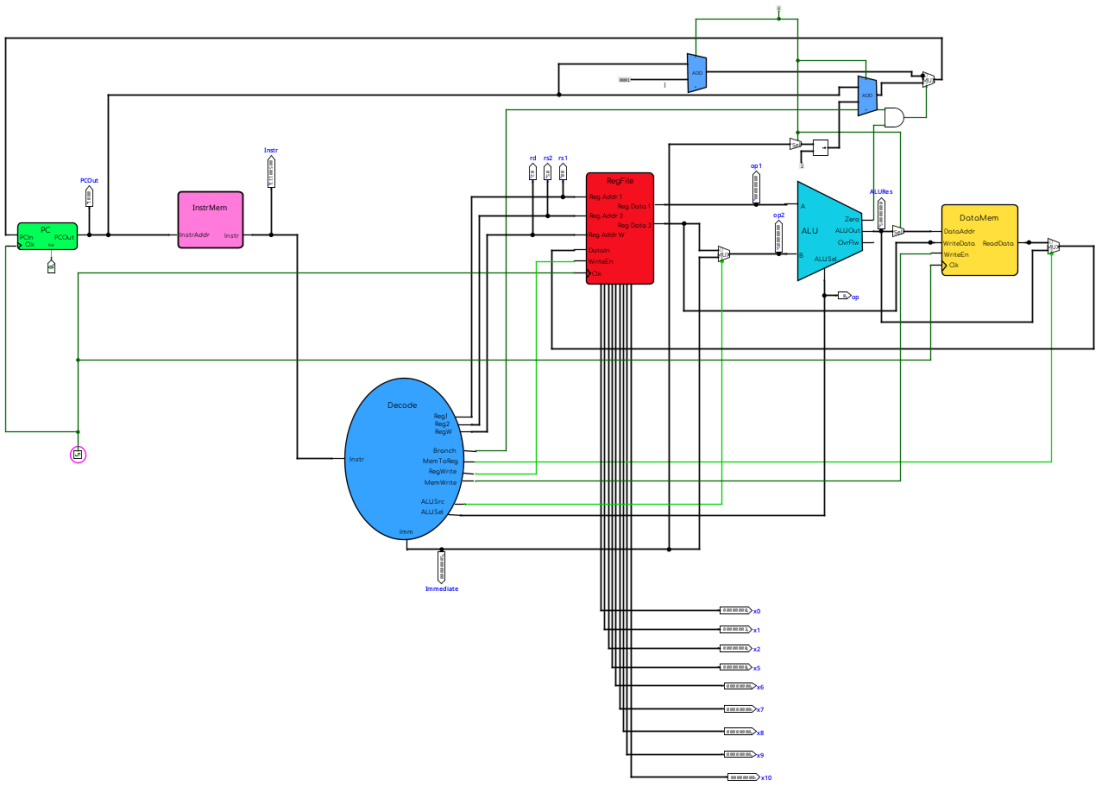
Here are the timing diagram simulations for the same :



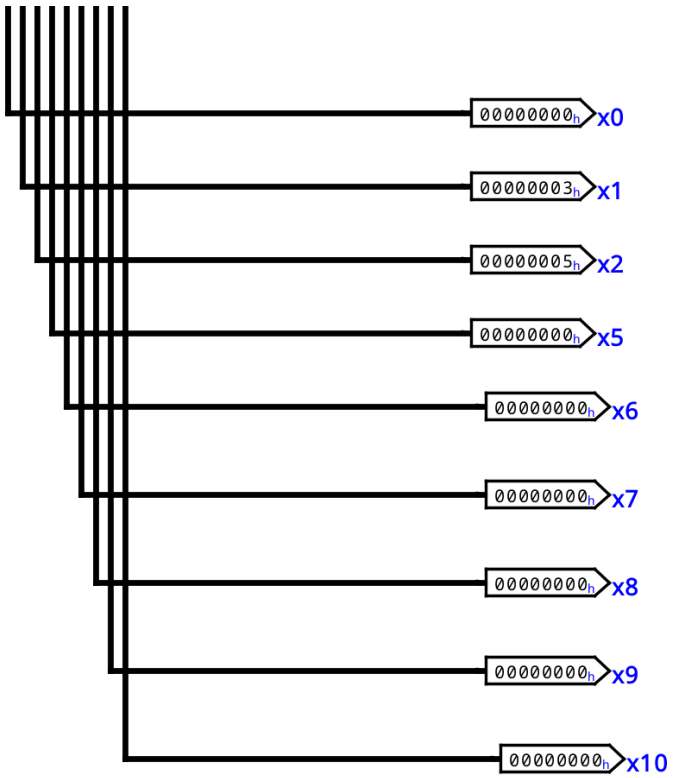
After each instruction is executed we provide the contents of the registers and the complete processor:

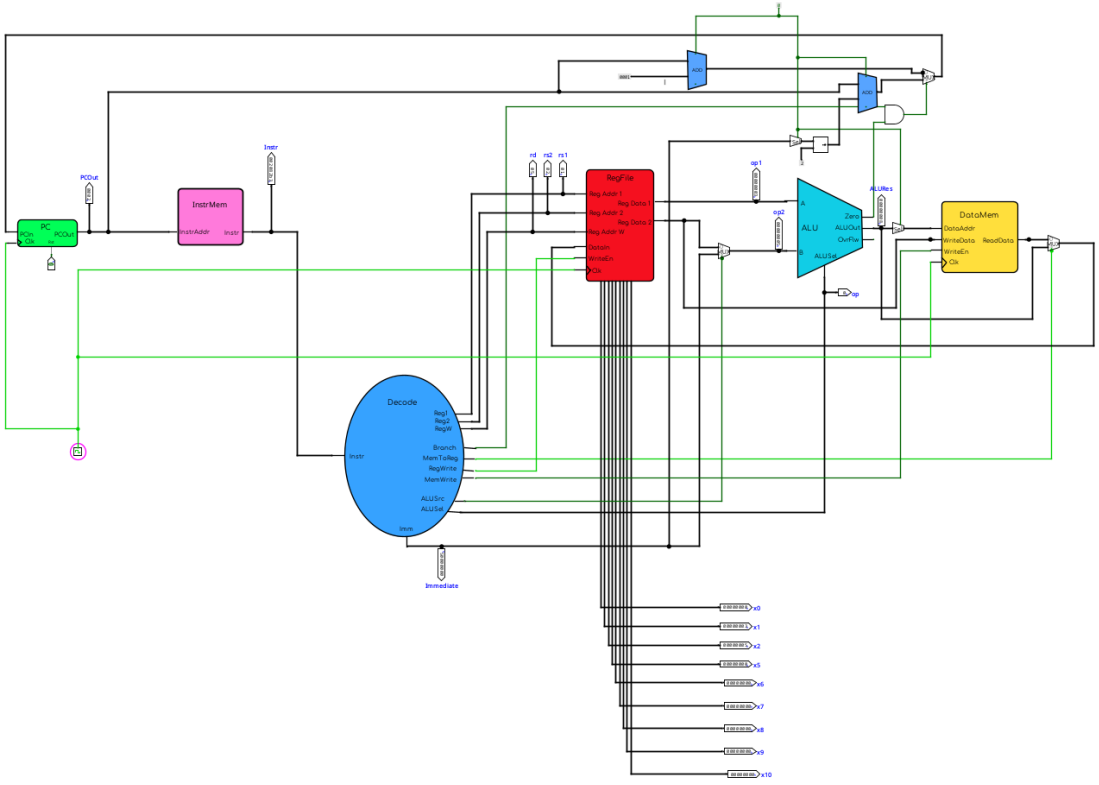
1.


00000000h x0
00000003h x1
00000000h x2
00000000h x5
00000000h x6
00000000h x7
00000000h x8
00000000h x9
00000000h x10

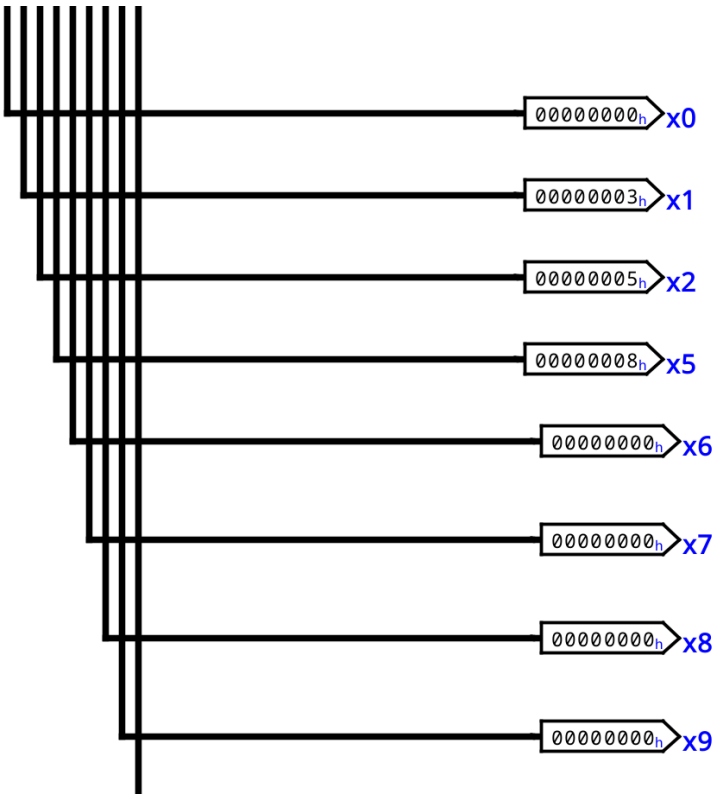

PC
InstrMem
RegFile
ALU
DataMem
Decode
Immediate

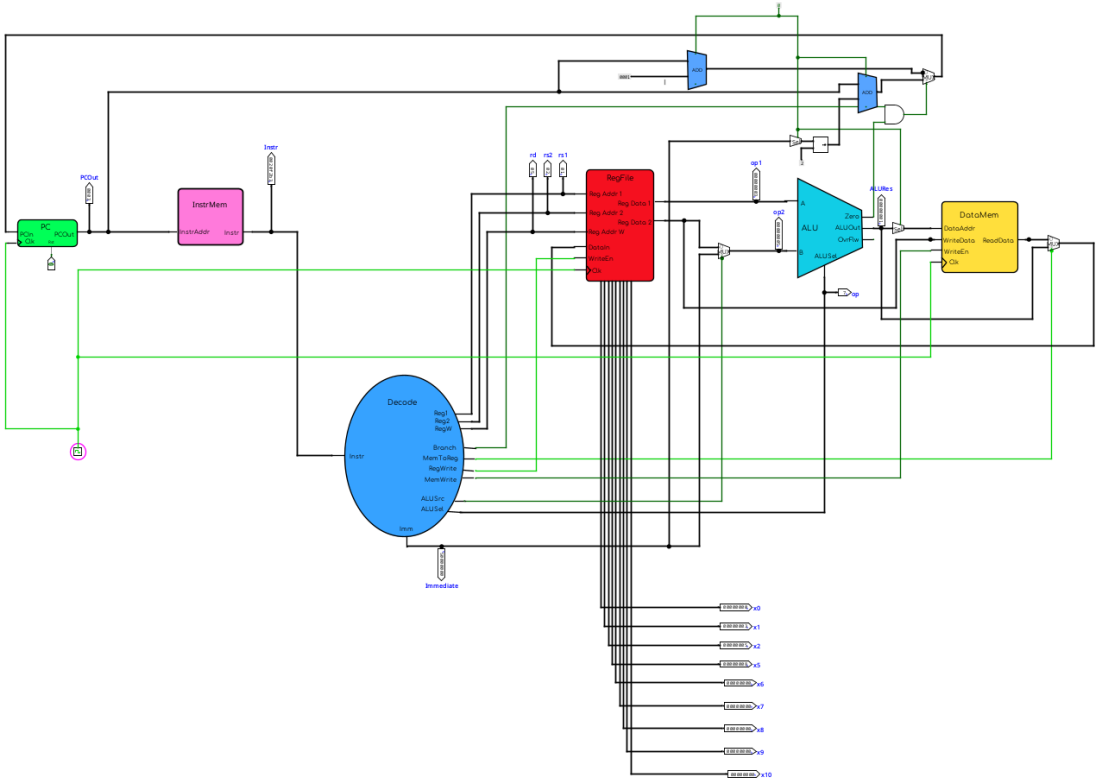
2.


00000000h x0
00000003h x1
00000005h x2
00000000h x5
00000000h x6
00000000h x7
00000000h x8
00000000h x9
00000000h x10

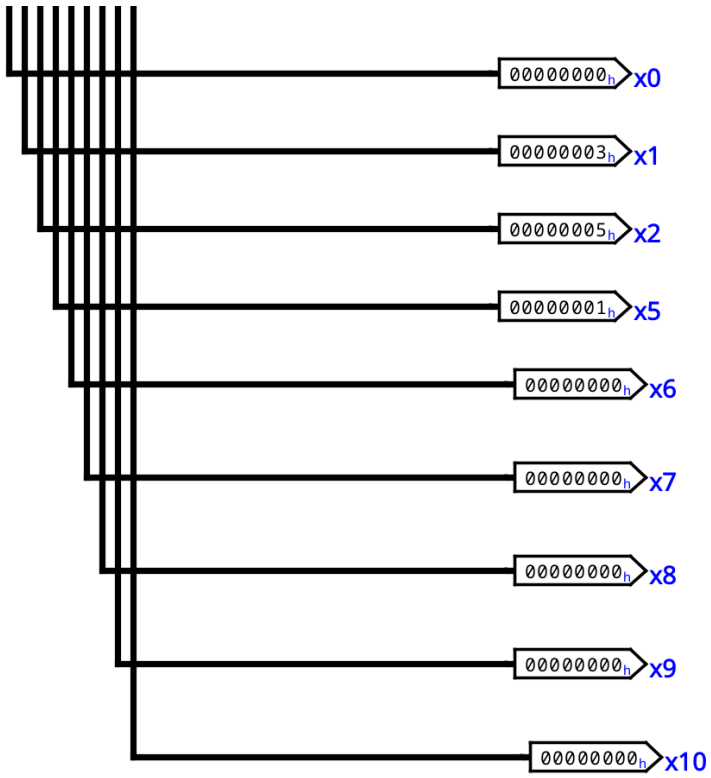

PCOut
Instr
InstrMem
RegFile
ALU
DataMem
Decode
Immediate

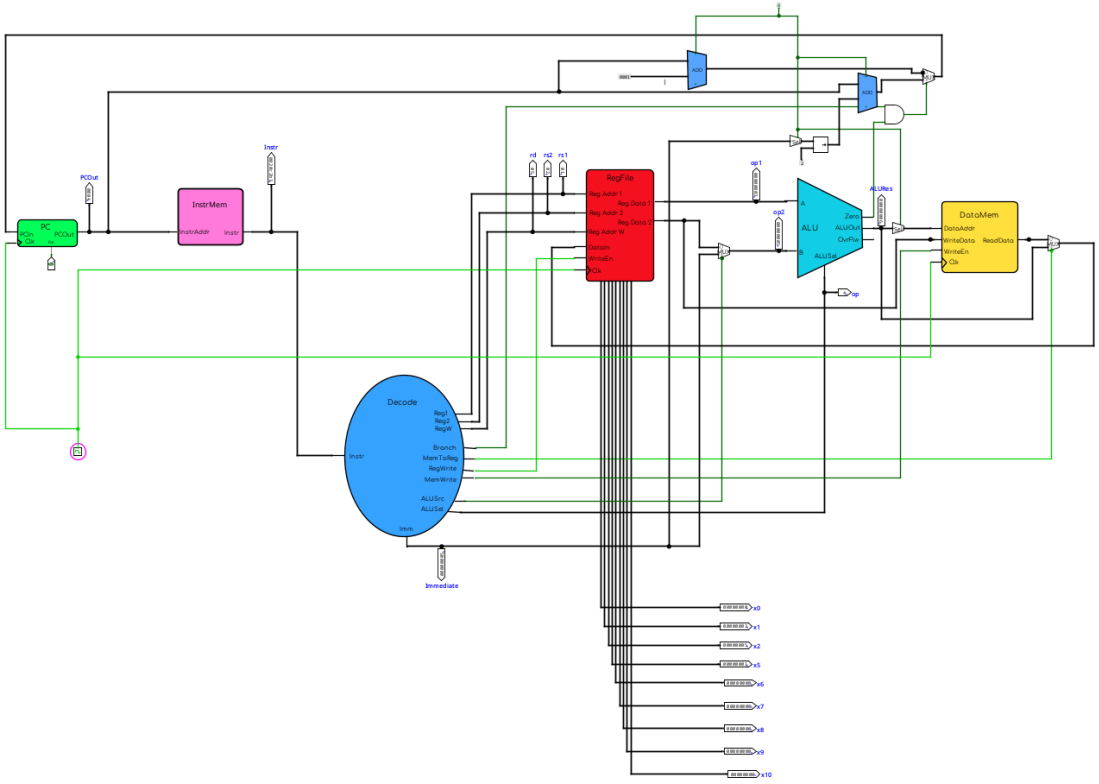
3.


00000000h x0
00000003h x1
00000005h x2
00000008h x5
00000000h x6
00000000h x7
00000000h x8
00000000h x9


PC
InstrMem
RegFile
ALU
DataMem
Decode

4.


00000000h x0
00000003h x1
00000005h x2
00000001h x5
00000000h x6
00000000h x7
00000000h x8
00000000h x9
00000000h x10


PCOut
InstrMem
Instr
RegFile
ALU
DataMem
Decode
Immediate

Now we test using a simple loop using branch :

addi x1,x0,10

addi x2,x0,0

loop:

add x2,x2,x1

addi x1,x1,-1

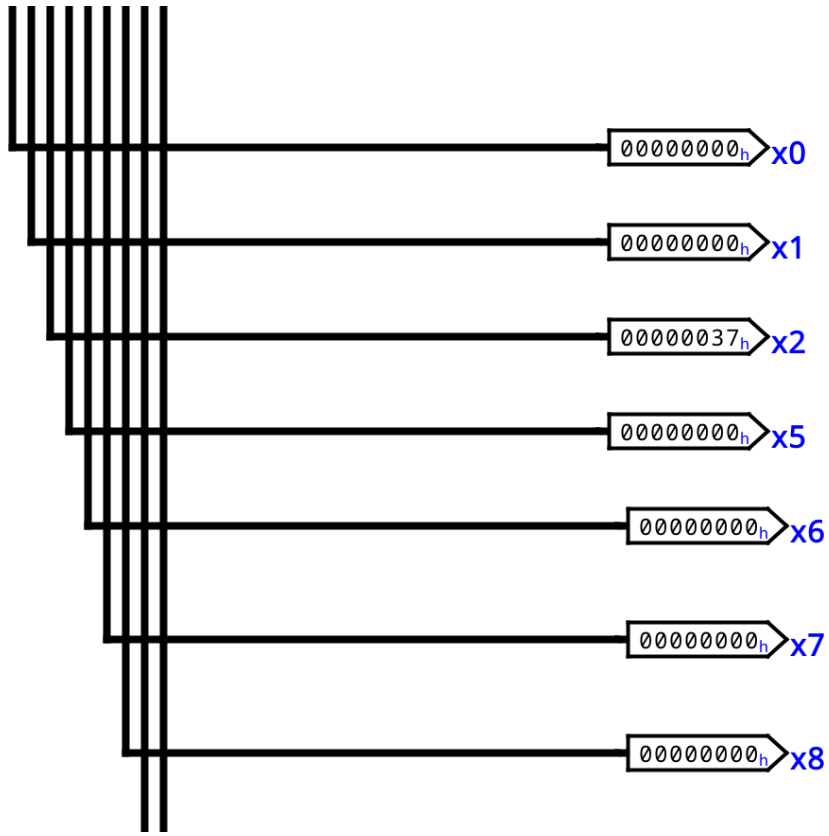
beq x0,x1,exit

beq x0,x0,loop

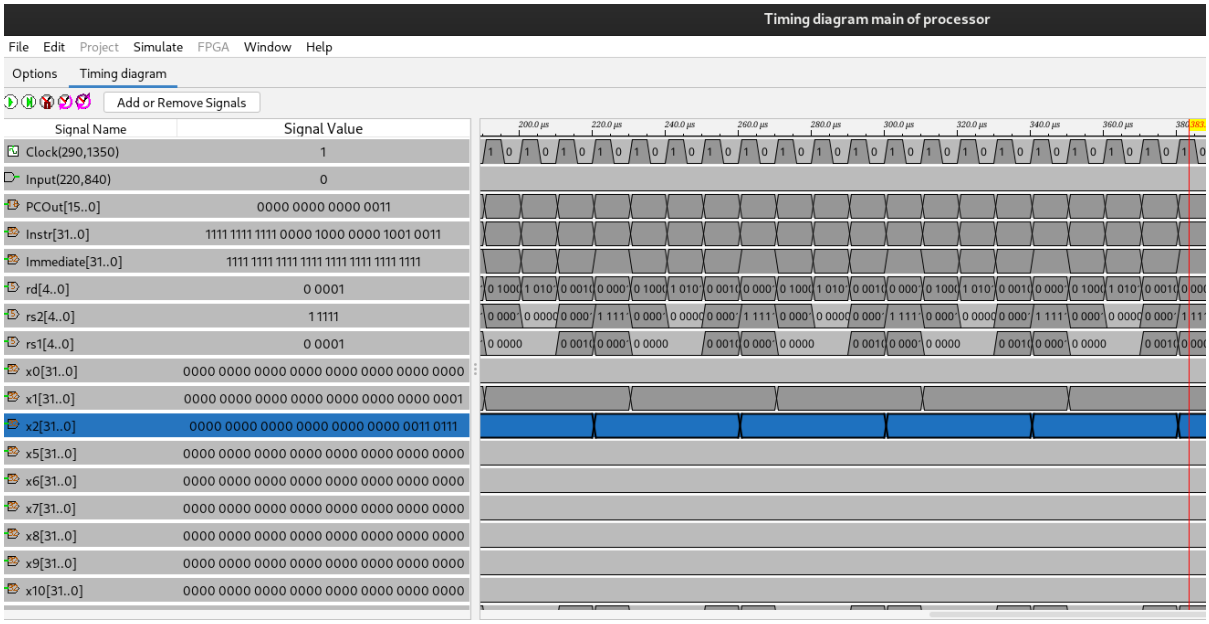
exit:

After the end, here are the results of the registers :

(x2 has sum of first 10 no.s : 55 = 37 (hex))



HERE IS THE COMPLETE SIMULATION :



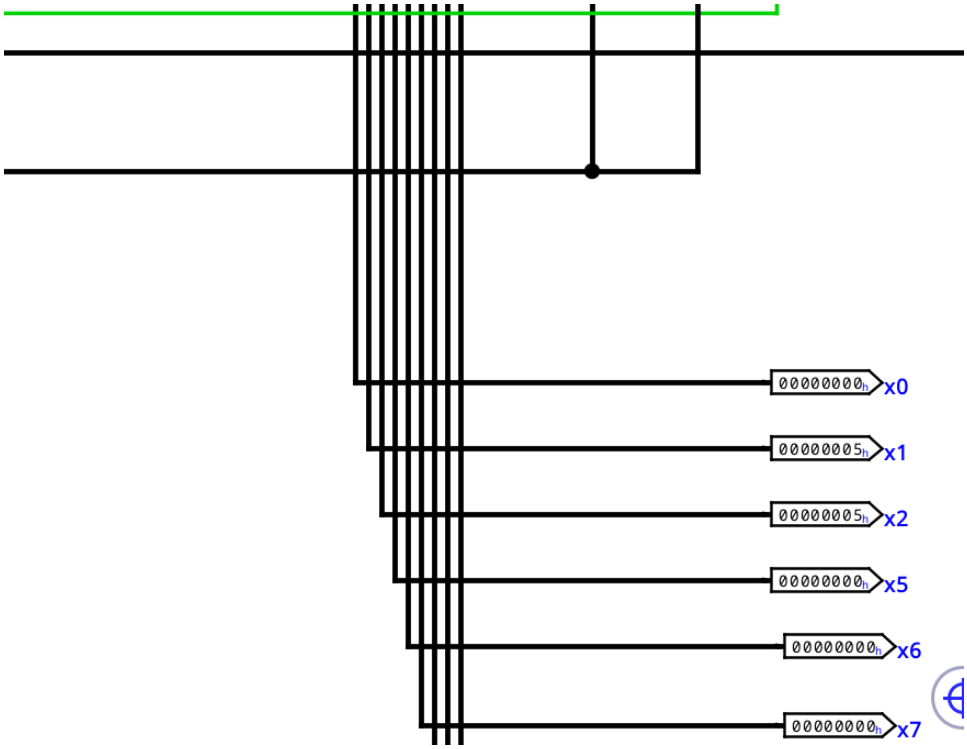
At last, we test using a small program which uses load and stores.

addi x1,x0,5

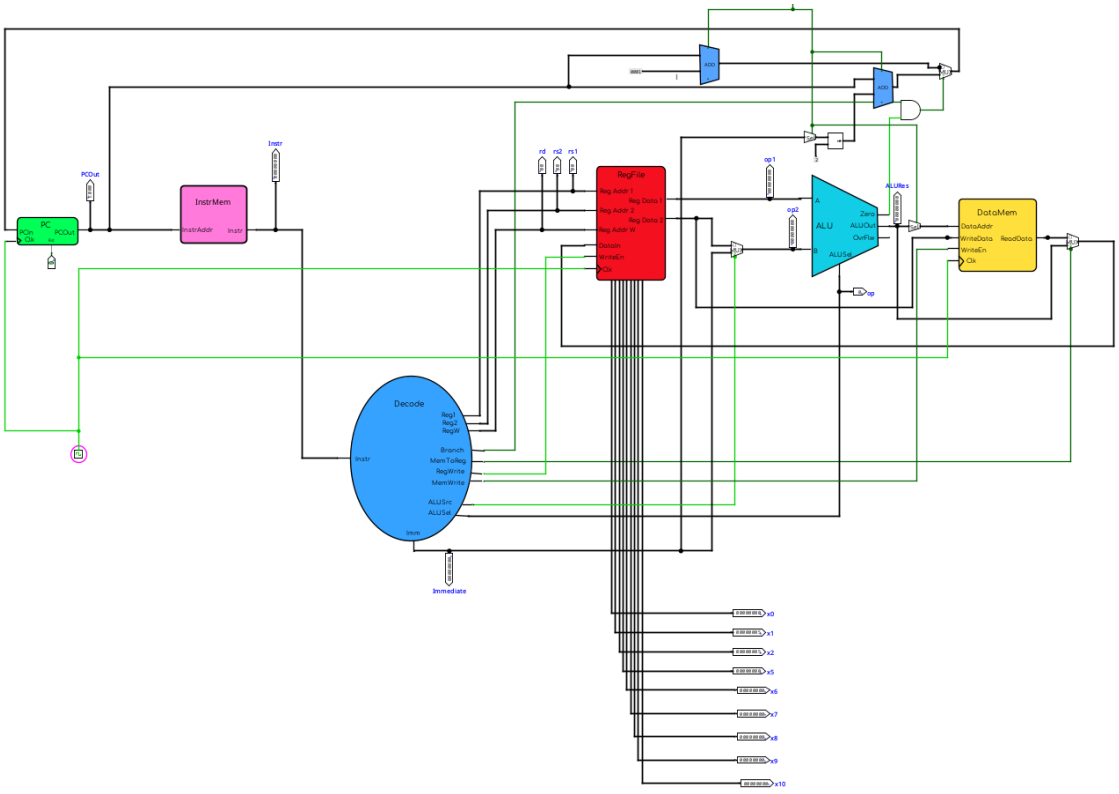
sw x1,4(x0)

lw x2 ,4(x0)

As we can see, the loading and storing occurs correctly.



Here is a picture during the load operation which highlights the datapath :



Conclusion :

We have checked all types of instructions using simple programs . We have checked that the ALU is working correctly, Branching is done correctly through testing a simple loop and that loading and storing works correctly by loading a register and then storing that value in another register.