EEE 445 COMPUTER ARCHITECTURE I

CNG 331 COMPUTER ORGANISATION

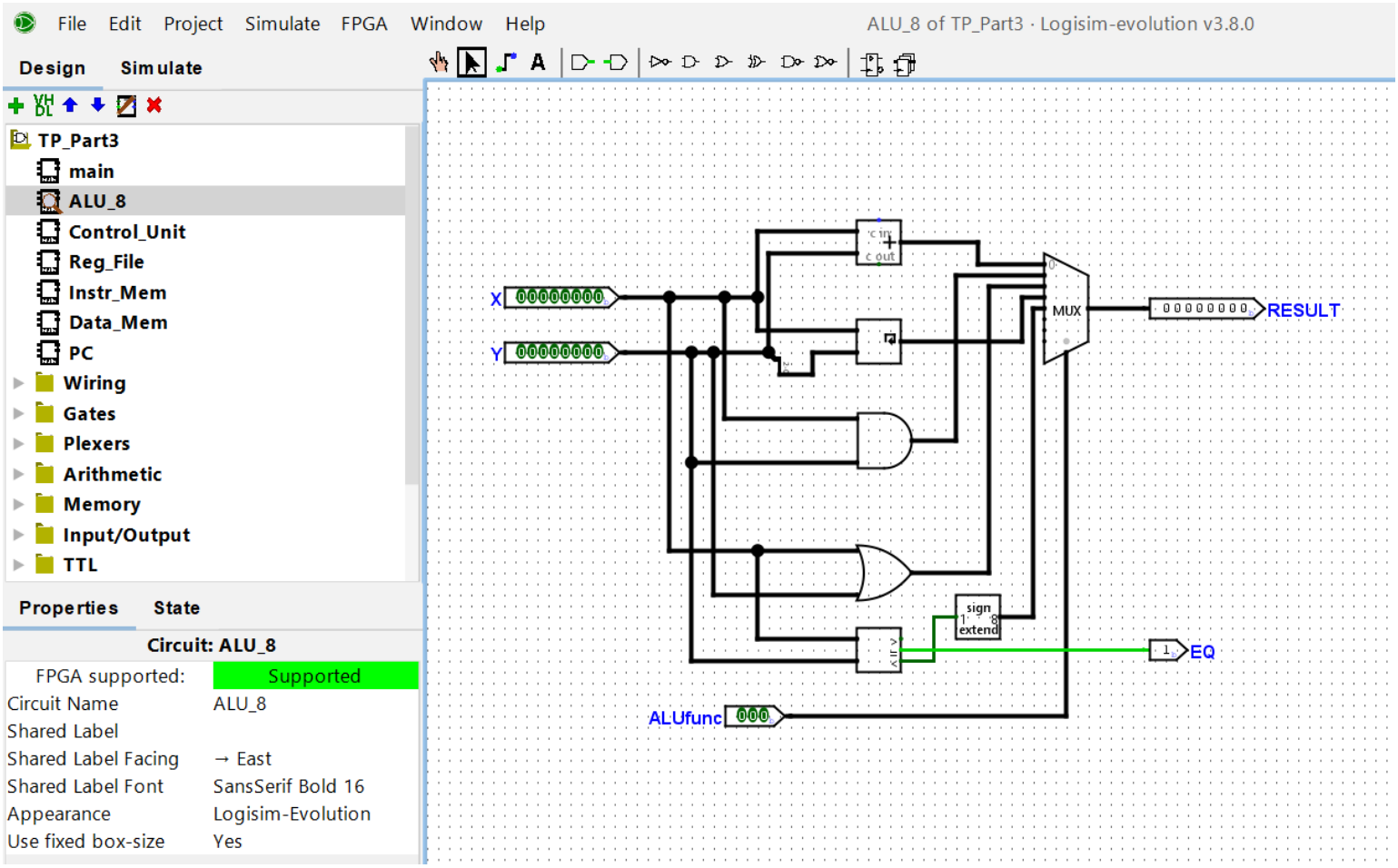
TERM PROJECT PART 3 REPORT

Fatma Erem Aksoy
2315075

The screenshots of each component designed in this project can be found below and a description is provided for the components that have changed after the previous (Part 2) version.

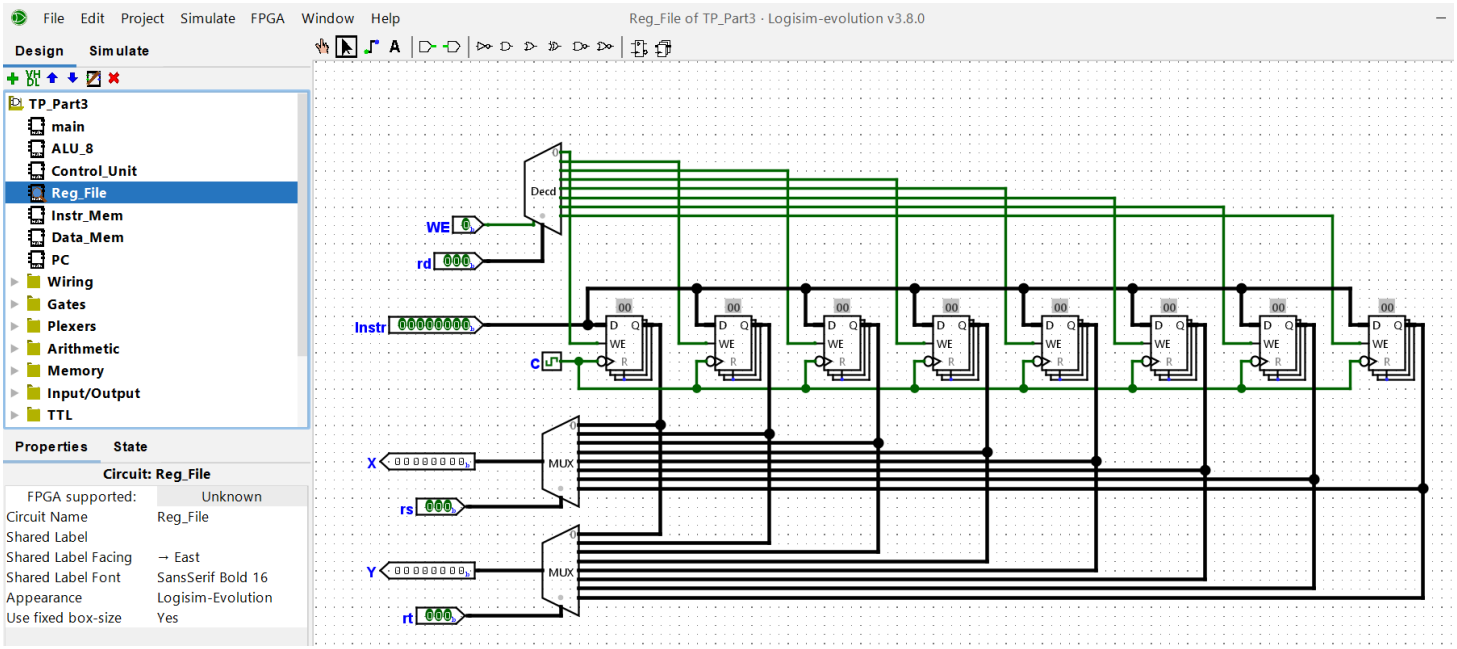
ALU_8:

ALU unit has changed a bit. Two extra operations are added to the previous design and now it is also capable of doing SLT operation which is selected by the MUX shown when the selector bit (ALUfunc) is 100. A sign extension is done for the output of the less than comparison as the MUX uses 8-bits data but the comparison result is 1 bit. Also a comparison will be made between X and Y inputs to check if these two values are equal to each other as the zero flag output, which is shown with EQ value in the figure. So EQ value will be 1 when X and Y inputs are equal to each other.



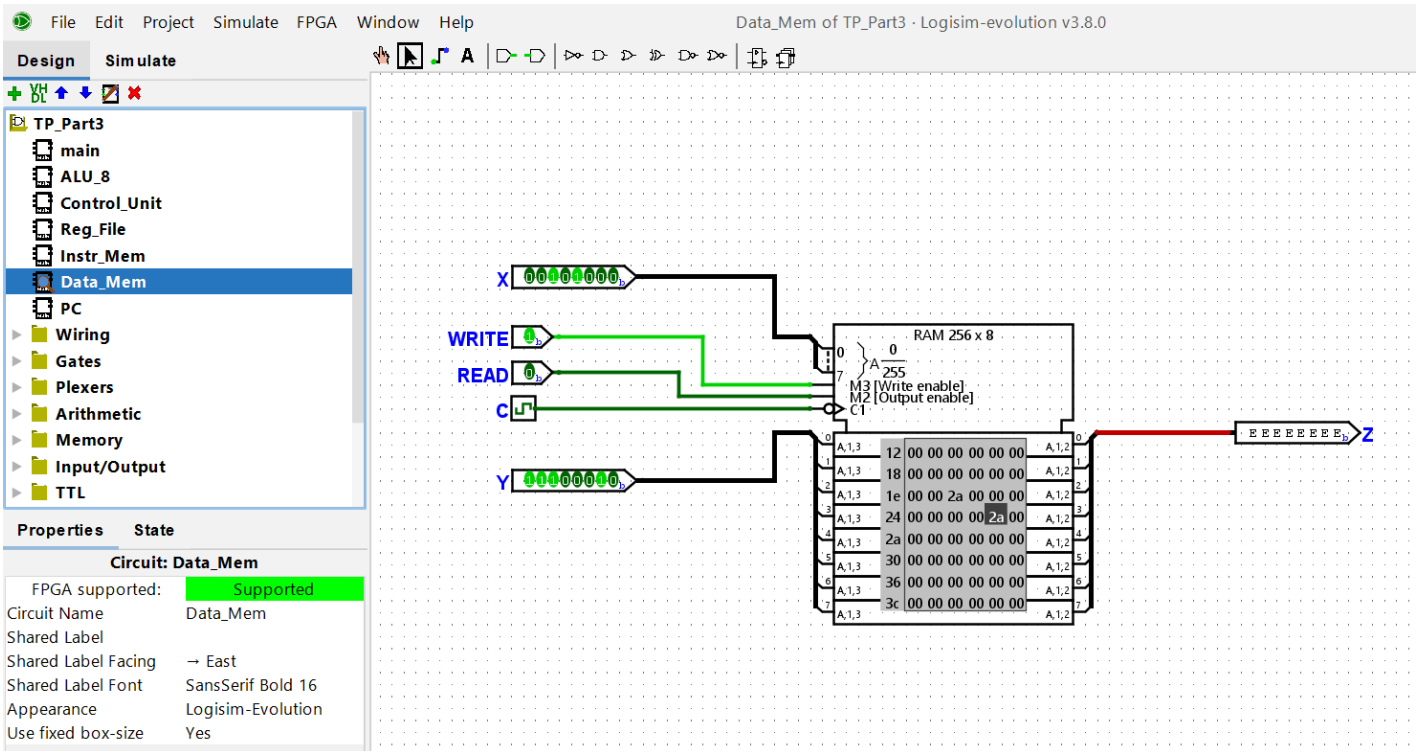
Register File:

This part of the design is same as the previous design, no change has applied.



Data Memory:

This part of the design is same as the previous design, no change has applied.



Control Unit:

The control signals for the outputs of this unit are calculated by using the K-maps and then finding the necessary equations, the signals are created with the proper gates. The expressions of each signal after all the calculations done for the K-maps, are as follows:

ALUfunc[2] = opcode[3].opcode[0]

ALUfunc[1] = opcode[3].Not(opcode[1])

ALUfunc[0] = opcode[2].Not(opcode[1])

MUX1[1] = Not(opcode[2]).opcode[1]

MUX1[0] = Not(opcode[1]).opcode[0]

MUX2 = (Not(opcode[2]).opcode[0]) + (Not(opcode[2]).opcode[1])

WE = (Not(opcode[3]).Not(opcode[2]).opcode[0]) + (Not(opcode[1]).Not(opcode[0])) +
(opcode[3].opcode[2])

WRITE = opcode[1].Not(opcode[0])

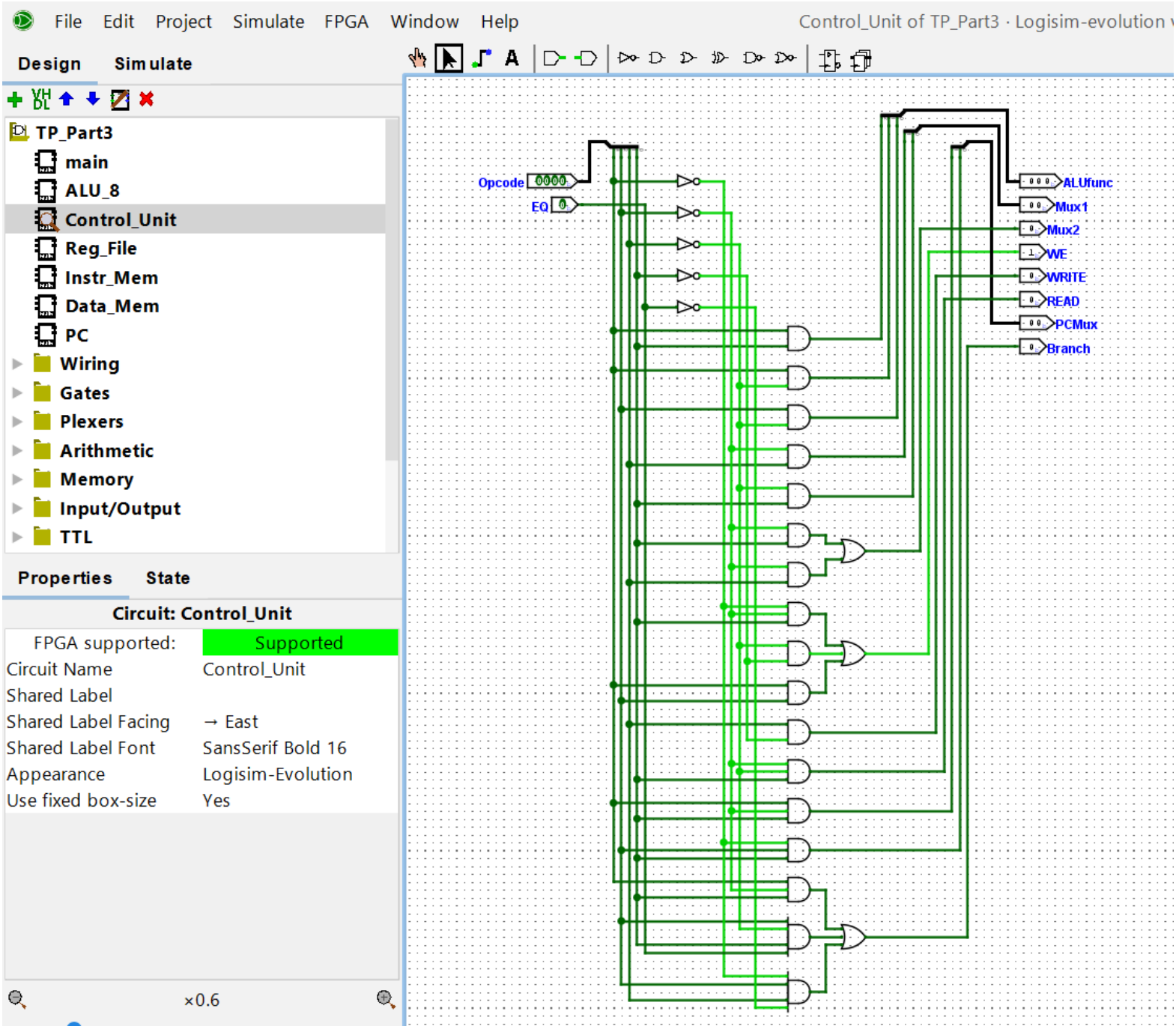
READ = Not(opcode[2]).Not(opcode[1]).opcode[0]

PCMux[1] = opcode[3].Not(opcode[2]).opcode[0]

PCMux[0] = Not(opcode[3]).opcode[2].opcode[0]

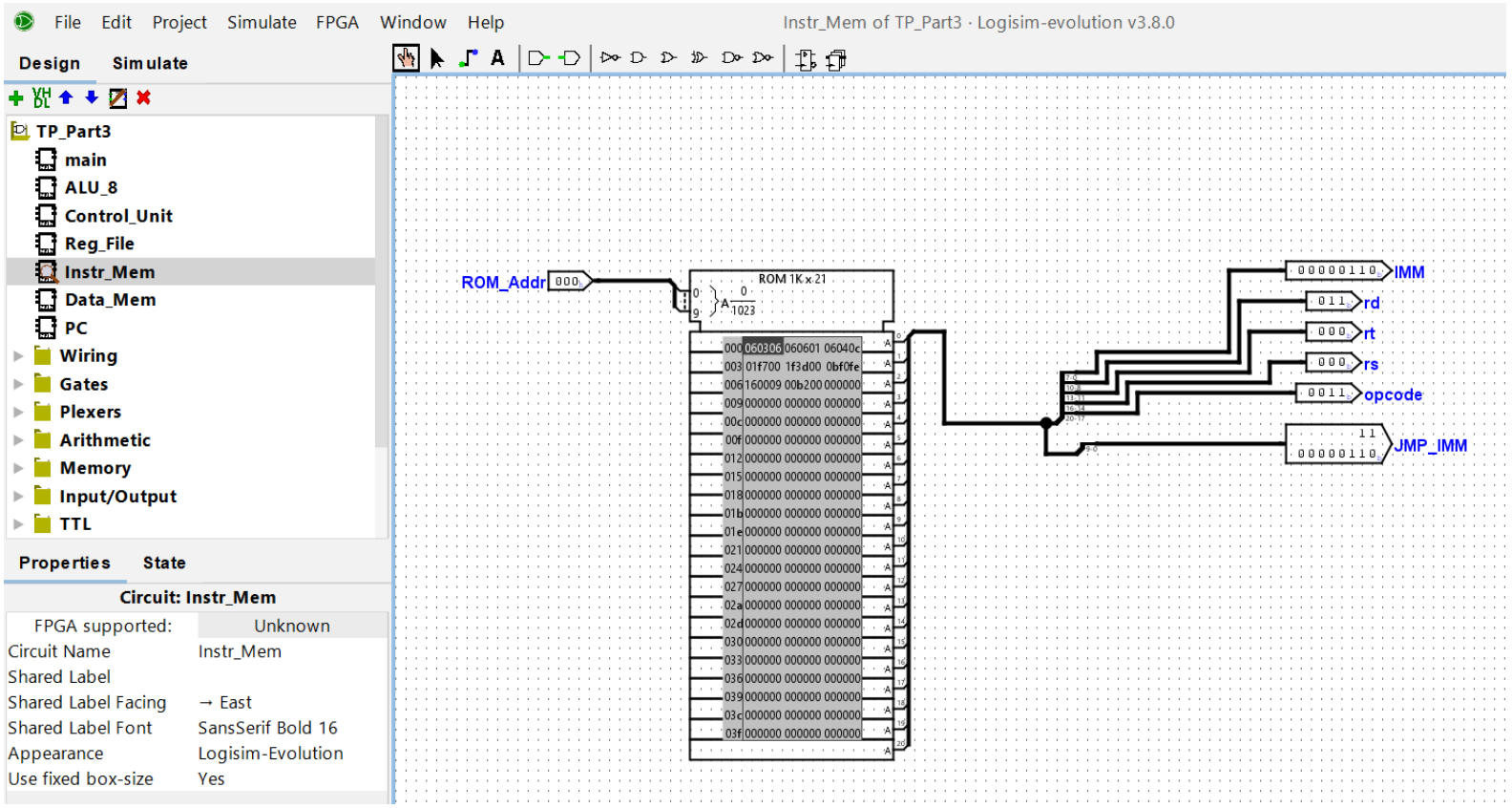
Branch = (opcode[3].Not(opcode[2]).opcode[0]) + (opcode[2].Not(opcode[1]).opcode[0].EQ) +
(Not(opcode[3]).opcode[2].opcode[1].Not(EQ))

The design of the new Control Unit is created based on those calculations, so it can obviously be seen what gates each signal is connected to. Those signals will be sent to the different units of the design for different operations and detailed information of those signals will be given in the related sections later. The screenshot of the new Control Unit can be seen in the figure below.


File
Edit
Project
Simulate
FPGA
Window
Help
Control_Unit of TP_Part3 · Logisim-evolution
Design
Simulate
TP_Part3
main
ALU_8
Control_Unit
Reg_File
Instr_Mem
Data_Mem
PC
Wiring
Gates
Plexers
Arithmetic
Memory
Input/Output
TTL
Properties
State
Circuit: Control_Unit
FPGA supported: Supported
Circuit Name Control_Unit
Shared Label
Shared Label Facing → East
Shared Label Font SansSerif Bold 16
Appearance Logisim-Evolution
Use fixed box-size Yes
×0.6
Opcode
EQ
ALUfunc
Mux1
Mux2
WE
WRITE
READ
PCMux
Branch

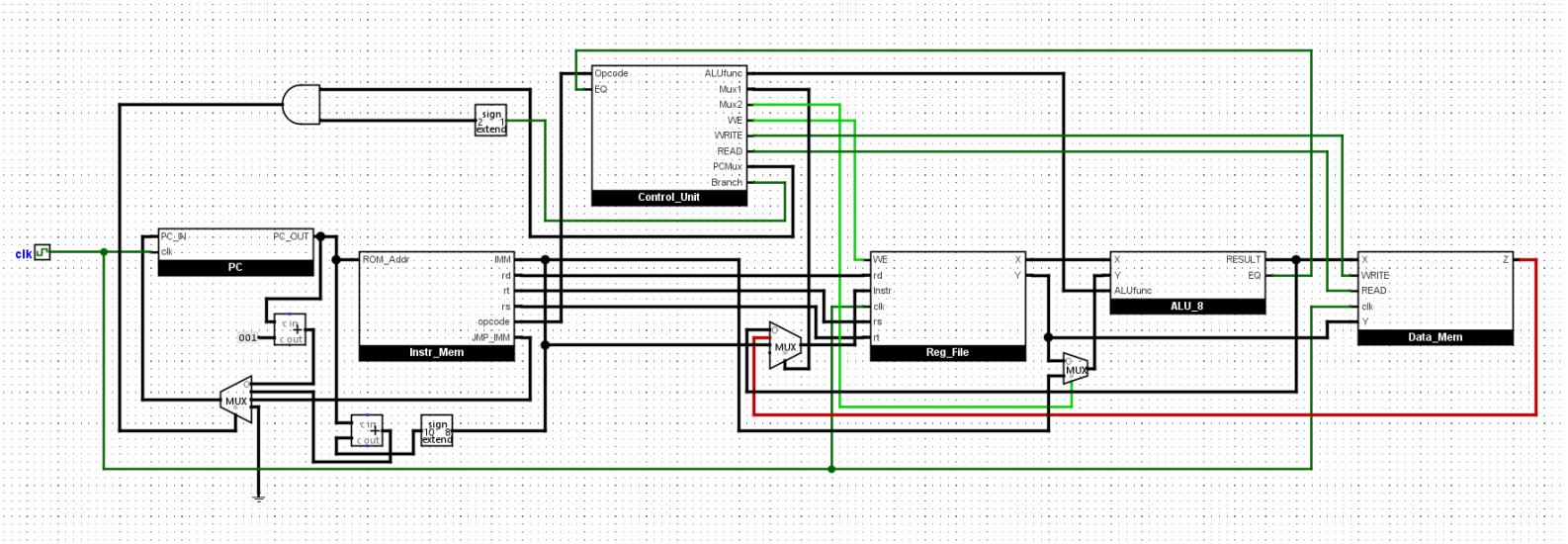
Instruction Memory:

As an additional output to the previous design, this time there are two different immediate values in the output side, showing the type as branch or jump. The difference between these two values apart from what they represent, branch value is 8 bits like a usual immediate and jump immediate value is 10 bits. The operation specifically will be selected later with the help of a MUX, which will select among three options where the first one (PCMux=00), giving the output of PC+1 as usual PC increment, the second one (PCMux=01), giving the output of PC+branch immediate value coming from this unit (the value named IMM in the figure), and the last one (PCMux=10), giving the output of jump immediate value coming from this unit (the one named JMP_IMM in the figure) again.



The Complete CPU Design:

The structure and purpose of each unit and component, which has changed since the previous design, is explained in details in the related sections above. The complete version of the CPU designed at the end can be seen from the figure below.



The Test Code (in Assembly):

LI R3, 6

LI R6, 1

LI R4, 12

L1: ADD R7, R7, R6

SLT R5, R4, R7

BEQ R6, R5, L1

J END

ADD R2, R2, R6

END:

The Test Code after converting it to binary and then hexadecimal:

The binary and hex representations of the instructions given can be found below:

0011 0000 0001 1000 0011 0 => 060306

0011 0000 0011 0000 0000 1 => 060601

0011 0000 0010 0000 0110 0 => 06040C

0000 1111 1011 1000 0000 0 => 01F700

1111 1001 1110 1000 0000 0 => 1F3D00

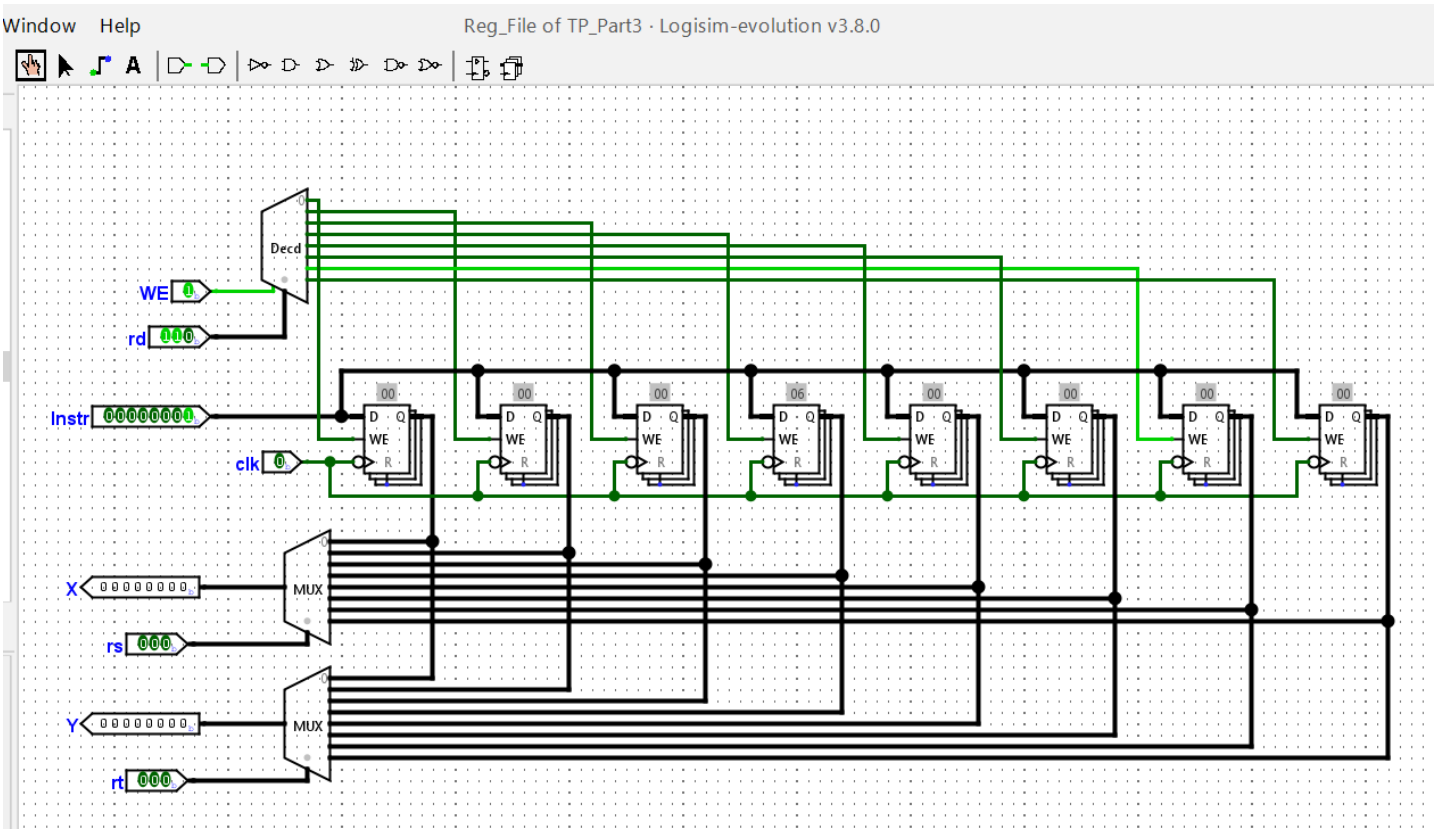
0101 1011 1000 0111 1111 0 => 0B70FE

1011 0000 0000 0000 0100 1 => 160009

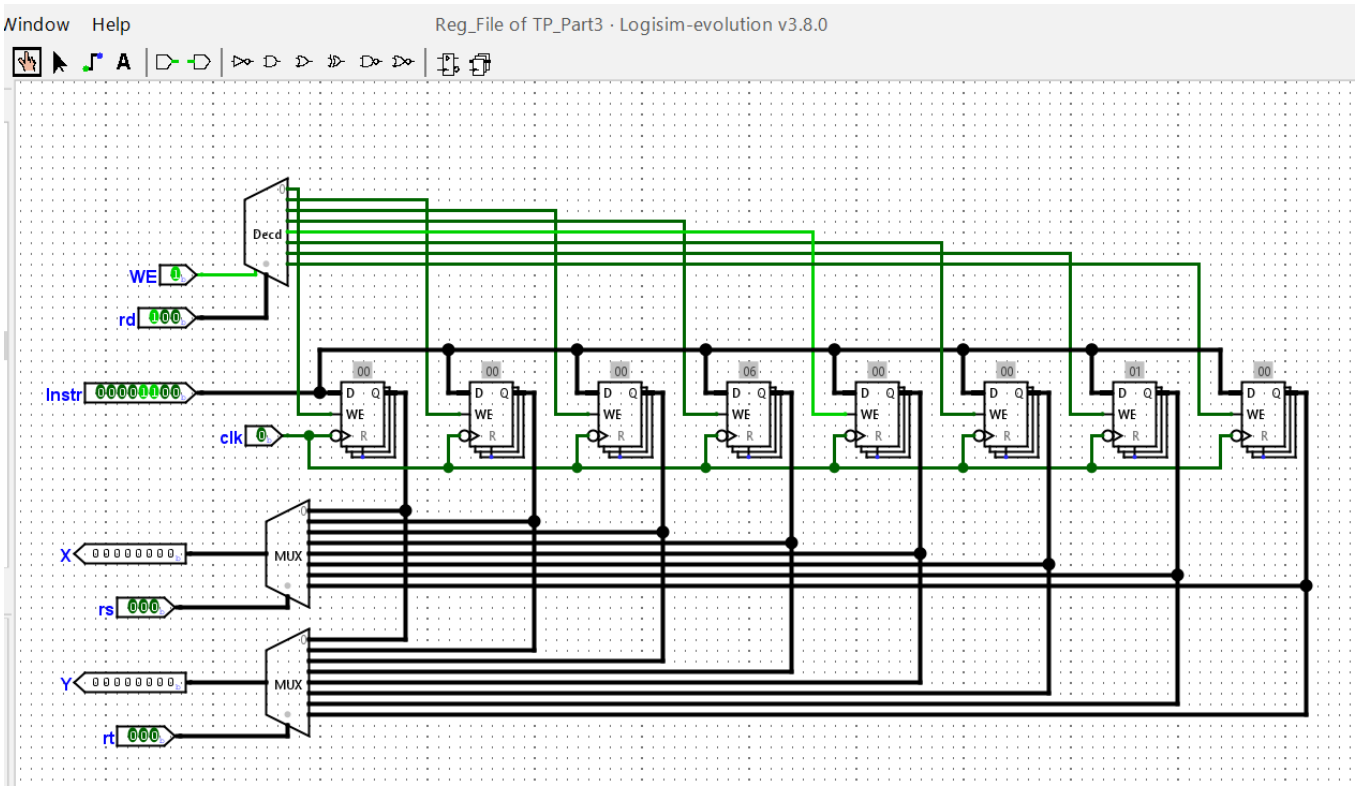
0000 0101 1001 0000 0000 0 => 00B200

Register File:

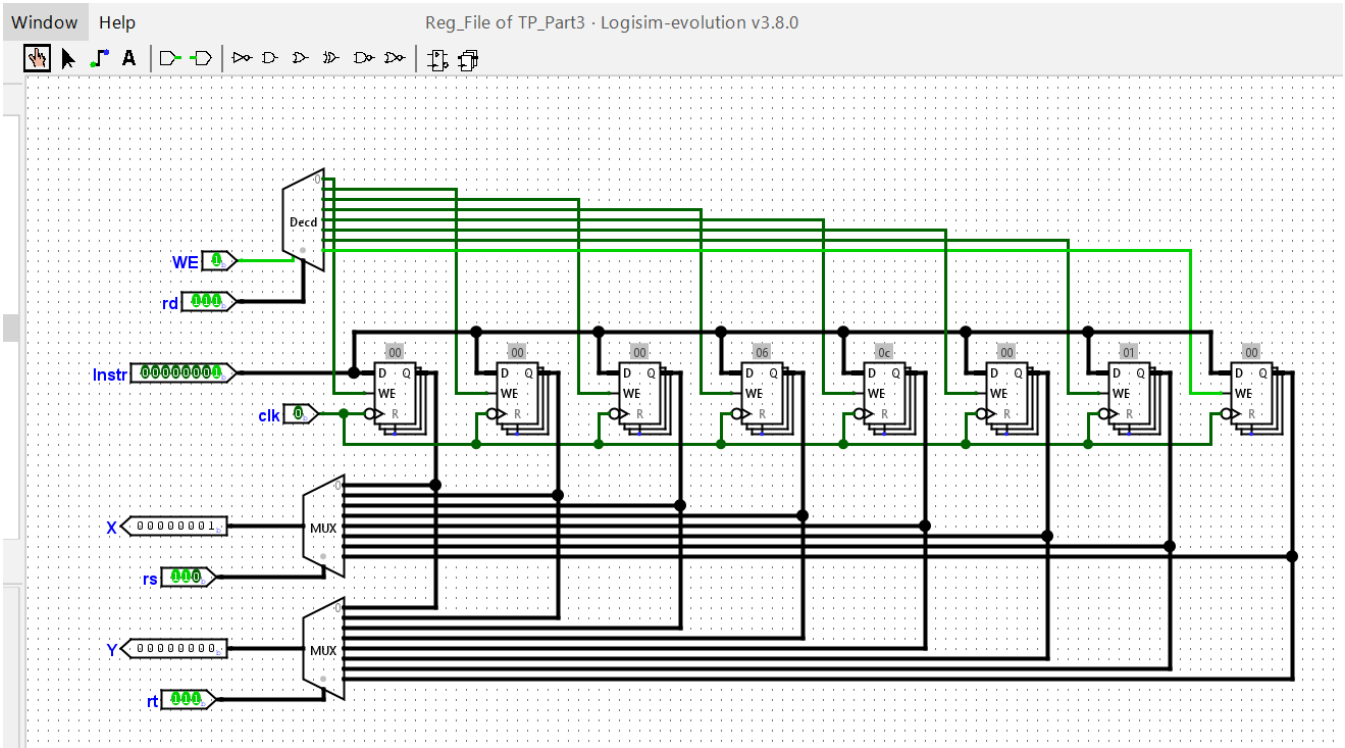
When PC=1 and the value 060306 from ROM is read (value 0x06 is saved in R3):



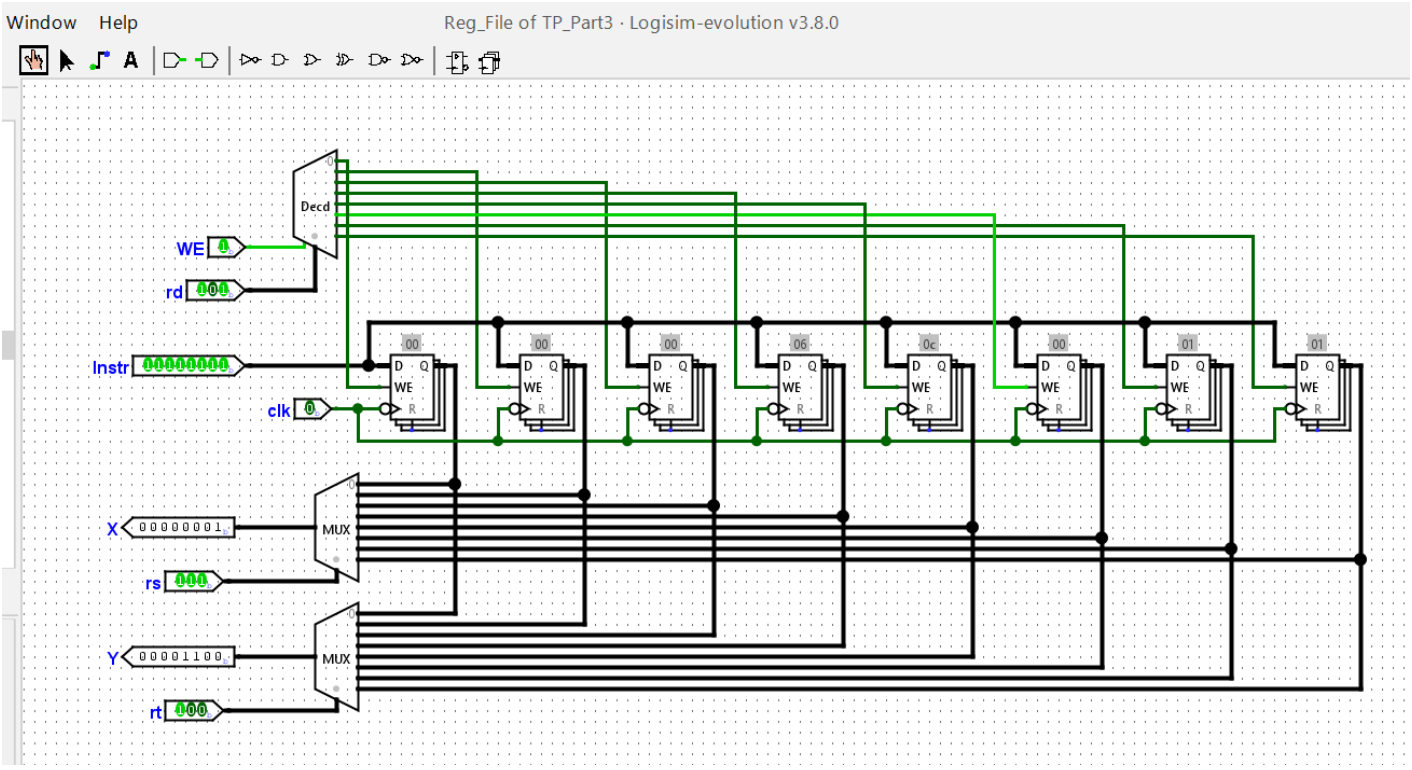
When PC=2 and the value 060601 from ROM is read (value 0x01 is saved in R6):



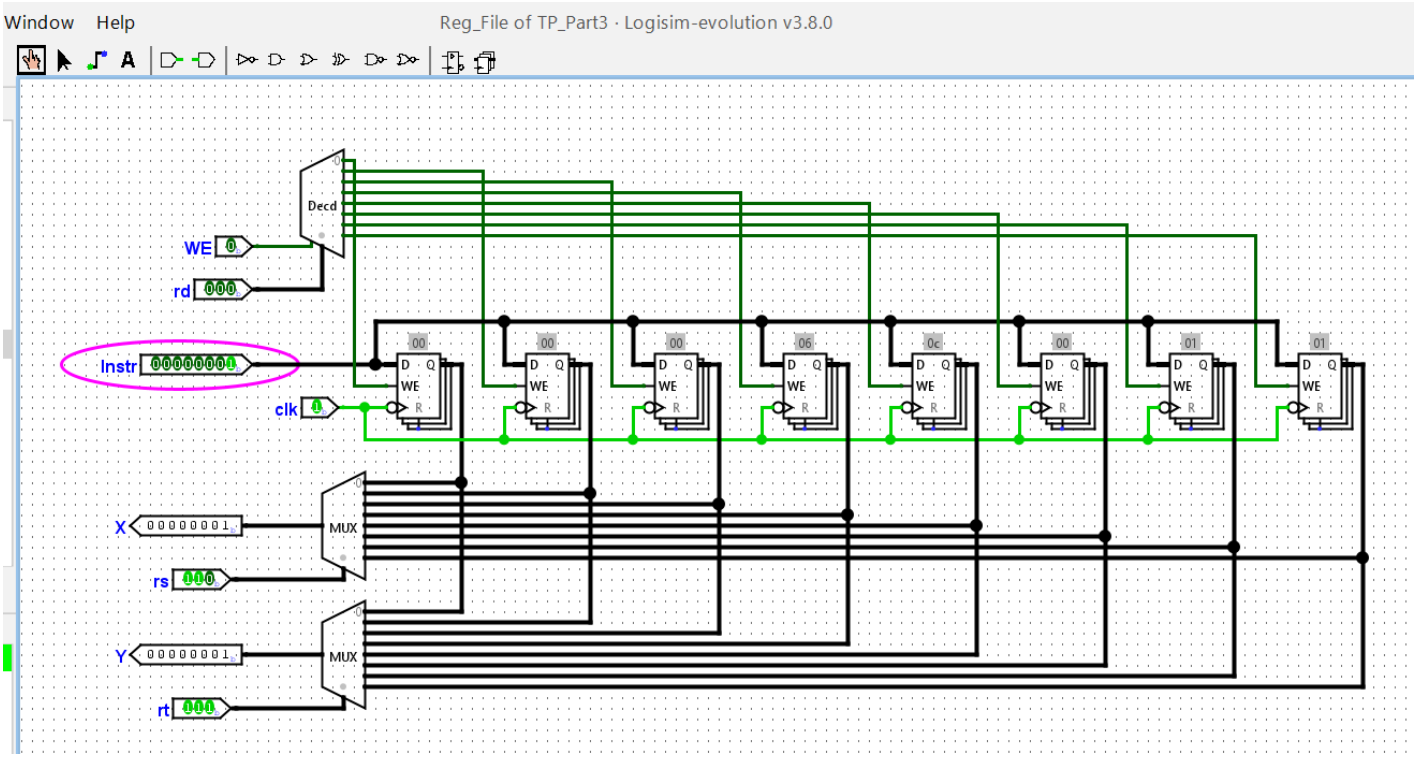
When PC=3 and the value 06040C from ROM is read (value 0x0C is saved in R4):



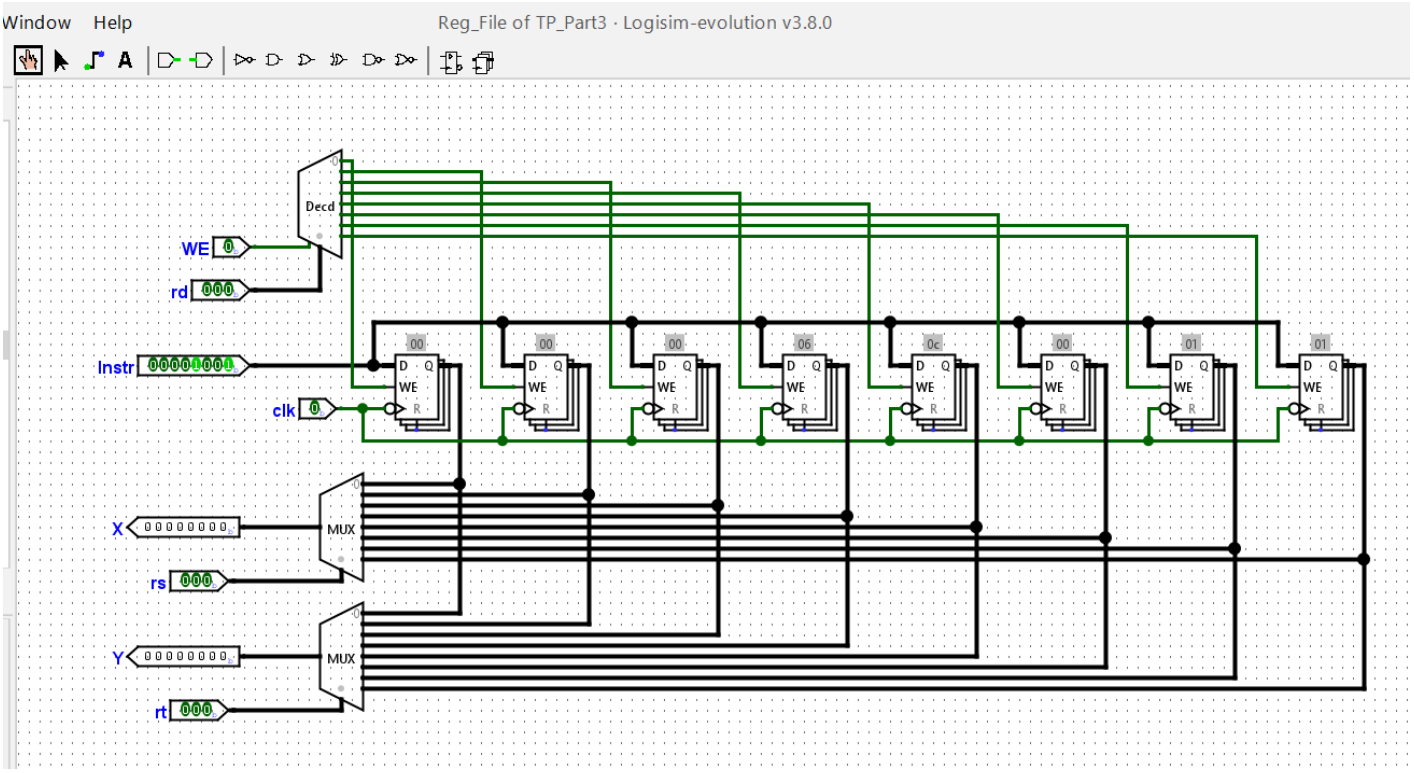
When PC=4 and the value 01F700 from ROM is read (result of R7+R6 saved in R7)



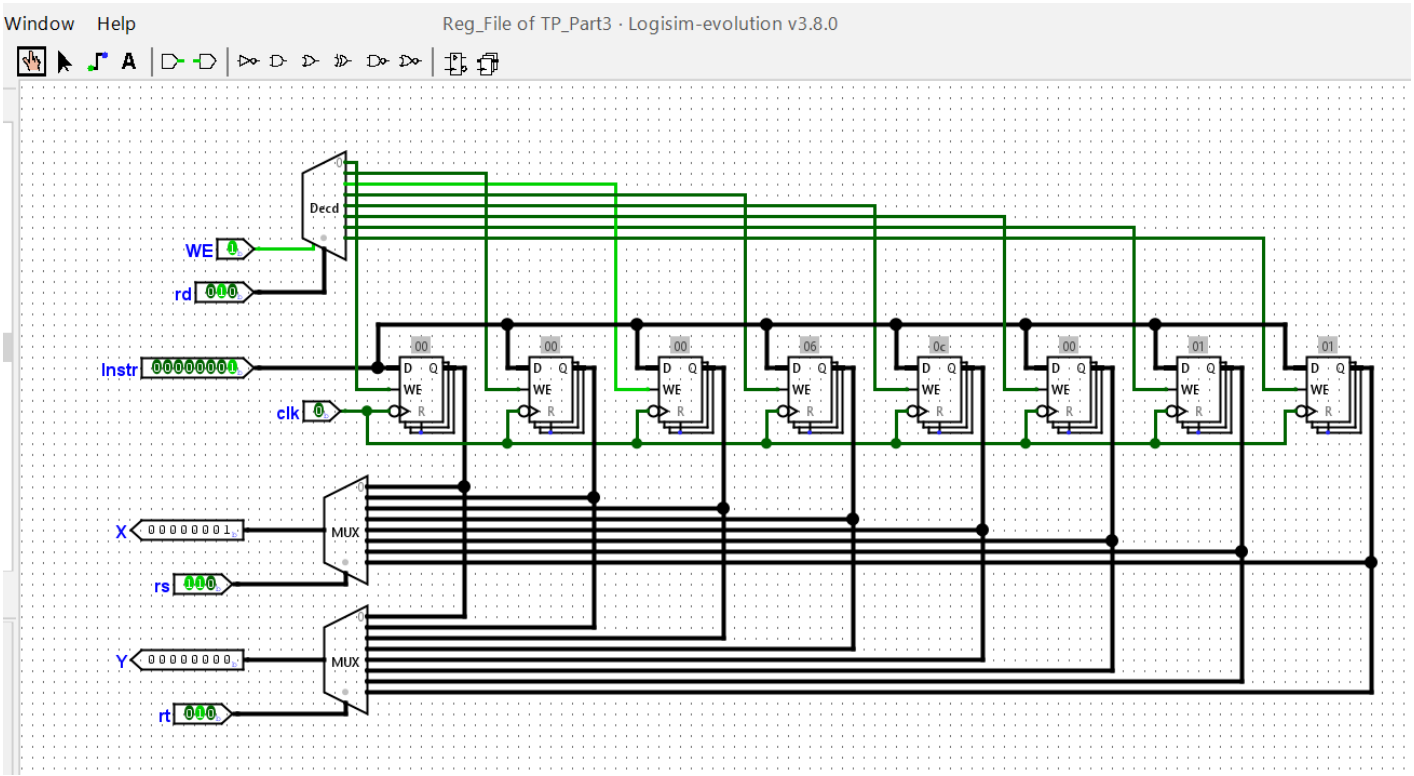
When PC=5 and the value 1F3D00 from ROM is read (result of comparison R4<R7 saved in R5)



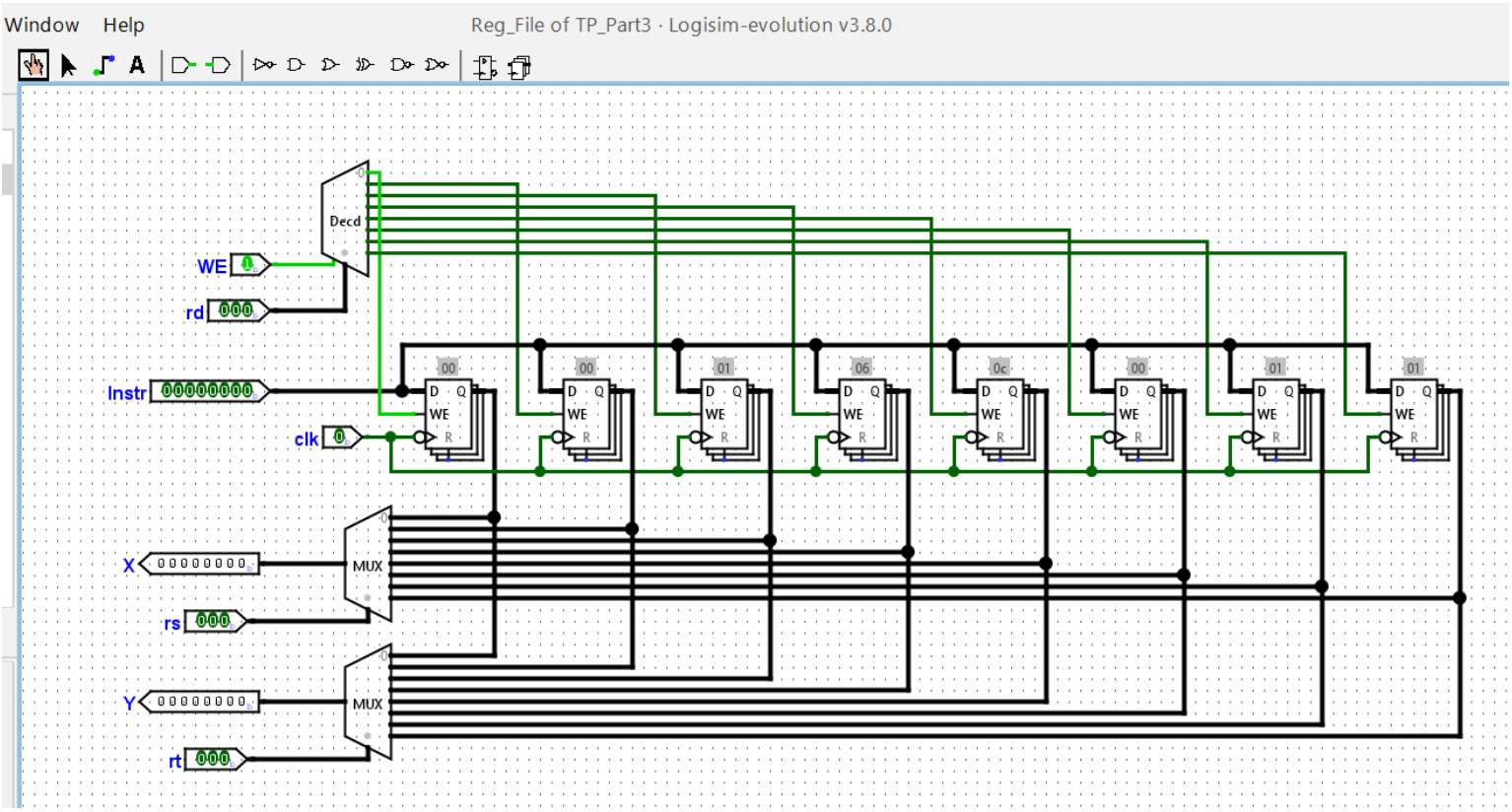
When PC=6 and the value 0B70FE from ROM is read (branch to L1 if comparison R6=R5)



When PC=7 and the value 160002 from ROM is read (jump to END-address 9)



When PC=8 and the value 00B200 from ROM is read (result of R2+R6 saved in R2)



All the cases for Register File for each PC value is shown in the figures above and it can be seen that the values are saved to the corresponding registers successfully.