EEE 445 COMPUTER ARCHITECTURE I

CNG 331 COMPUTER ORGANISATION

TERM PROJECT PART 2 REPORT

Fatma Erem Aksoy
2315075

Part 1: Building a Register File

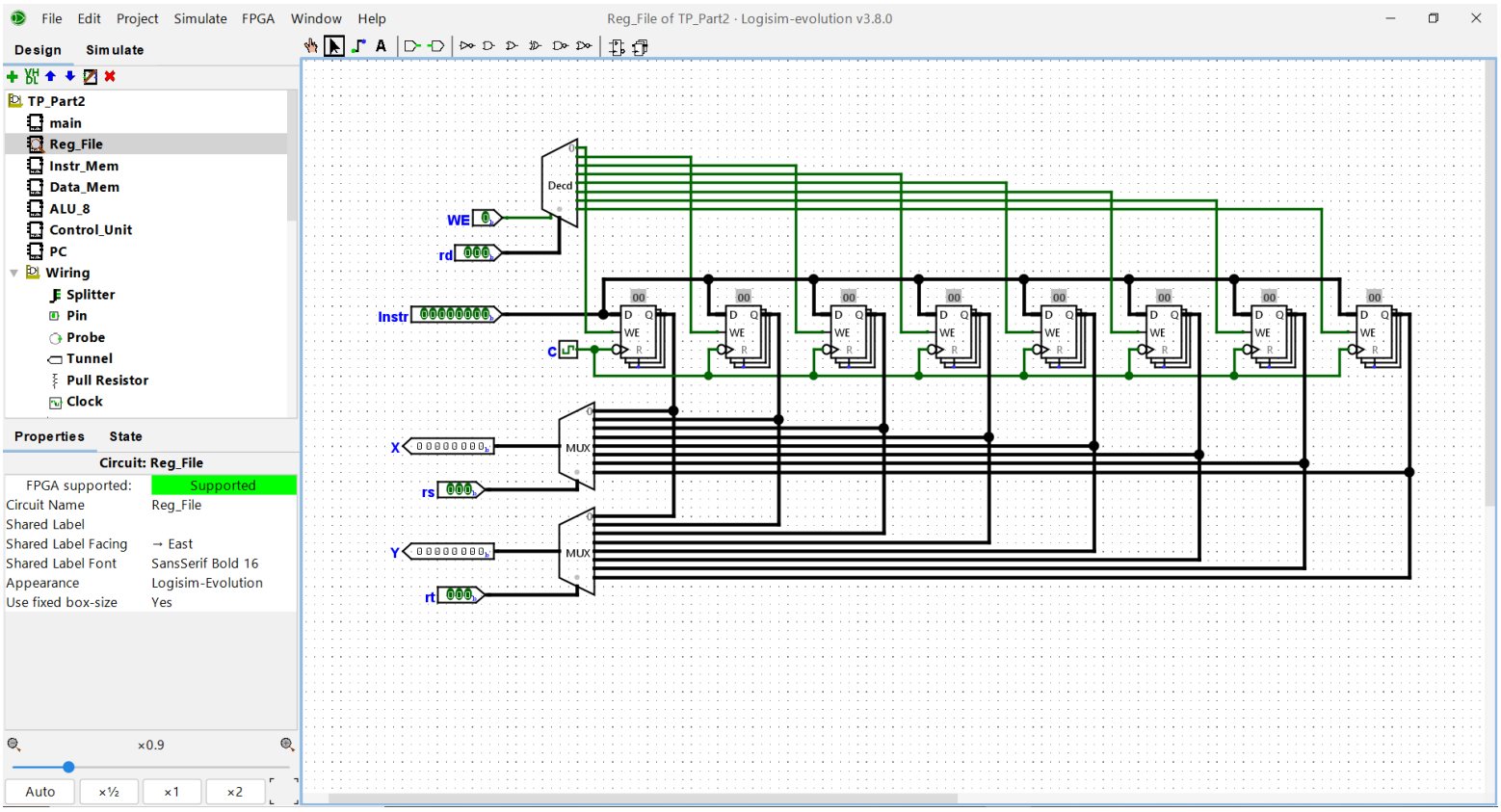

Figure 1: Register File

In the figure above, Figure 1, there are 8 registers connected to each other and each has 8-bit size since the ALU is 8-bit functionality. There are two different operations here, which are write and read. These operations are controlled by the control bits which are WE (write enable) and other register address bits; rd, rs and rt. The write operation happens with the help of the decoder shown in the figure. If WE input is set, then the writing operation will be performed based on the address sent by rd value. So the value on the register selected, with the rd control bits, will be written.

The data, Instr input in the figure, is sent to all the registers, and if the read operation will be performed, then the addresses of the registers, which register to select, will be determined by the control select inputs, which are rs and rt. Two MUXes are used here and both will generate an output, X and Y, from the Register file to be sent for different operations to different units (ALU, and ALU or Data Memory depending on the MUX select).

Both input data, Instr, and the outputs, X and Y, will be 8-bit data. The inputs Instr (after MUXing), rs, rd and rt are coming from the Instruction Memory where WE is coming from the Control Unit. A clock is also connected to the unit.

Part 2: Building Instruction and Data Memory

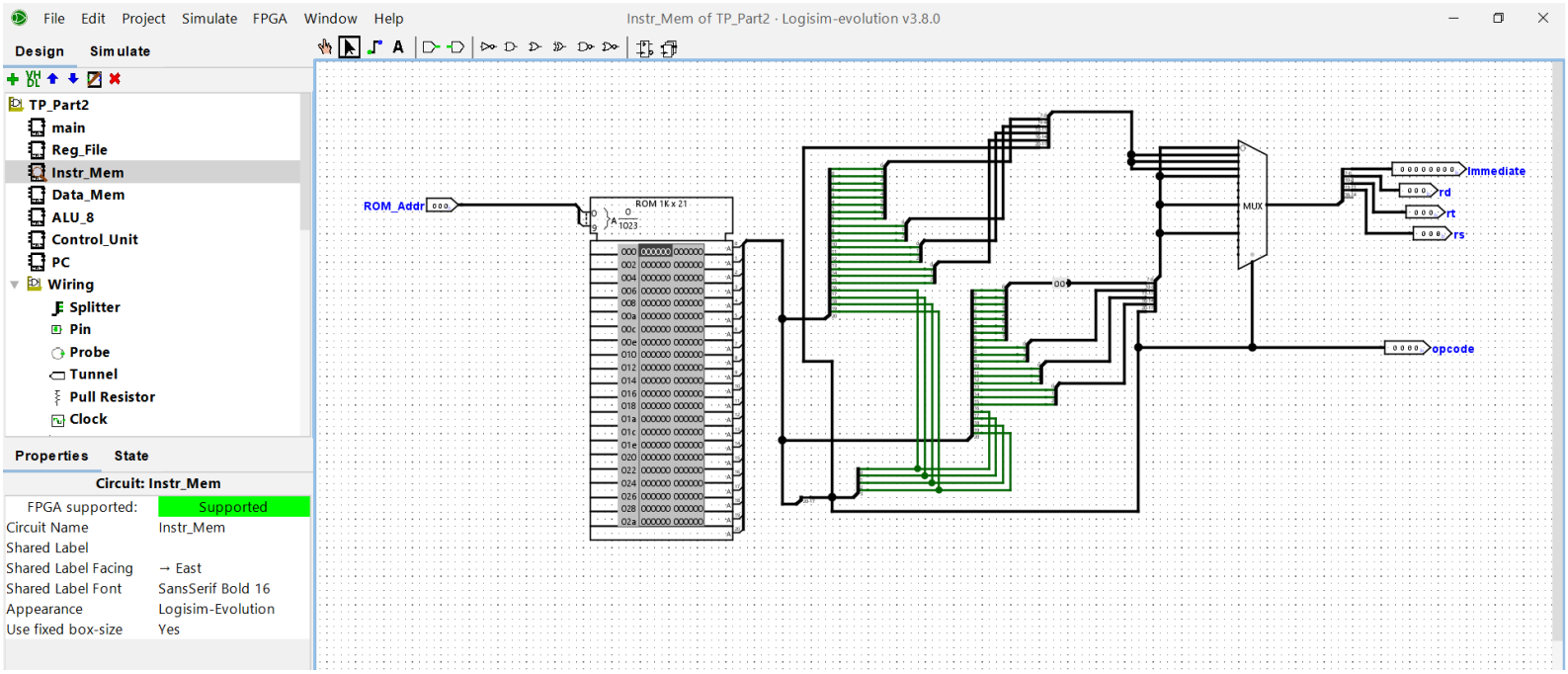

Figure 2: Instruction Memory

In the figure above, Figure 2, the Instruction Memory is shown. A ROM is used as the instruction memory. The address of ROM is 10-bit size and every 21 bits holds one address memory. The input ROM_Addr will come from the PC as the counter increases, the current address will be sent here to perform the necessary operations. The address will be used to get the 21-bit machine code and then it'll be used to generate opcode, rd, rs, rt and immediate (if it's an I type instruction) values. It will be done by using the splitters and a MUX. If it's an R type instruction then the last 8 bits will be all 0's, otherwise the last 8 bits are reserved for the immediate value. After splitting the address and appending 0's (if R type), the MUX will be used to select the type and the structure of the code basically. Then all the necessary values will be generated as outputs with the help of the table provided for those values.

So the output of this system will be Immediate (will be all 0's if R type), rd, rs, rt and opcode (for sending to the Control Unit to generate the control bits).

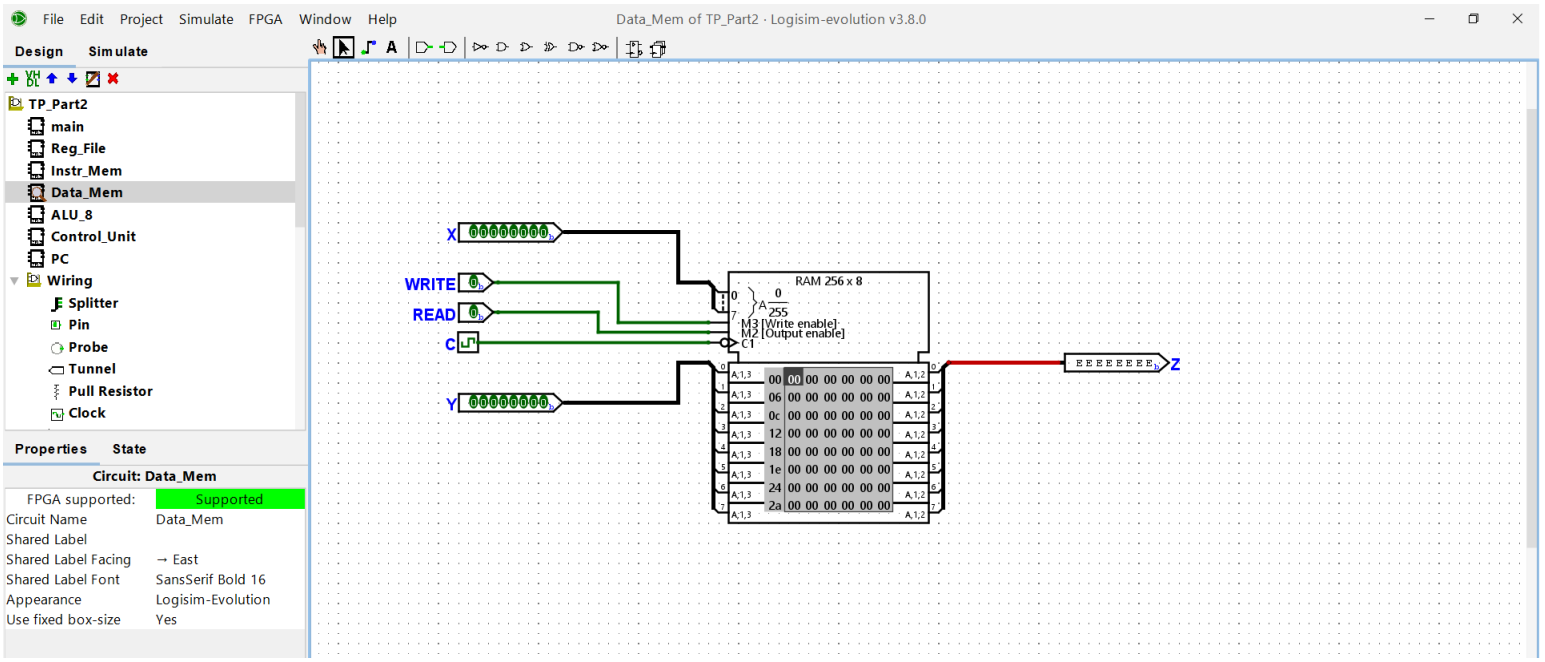

Figure 3: Data Memory

In the figure above, Figure 3, a Data Memory is implemented using RAM. X input represents the RAM address and it's 8-bit. WRITE and READ inputs are control bits coming from the Control Unit and Y is the data, which is coming from the ALU, that will be loaded/stored. A clock is also connected to the RAM. The output of the Data Memory is Z, which will be sent to the MUX between the Instruction Memory and the Register File.

So this unit is responsible from writing/reading the data by doing the operations load/store.

⇨ The difference between RAM and ROM is that the ROM is not responsible from writing the data because it is only able to read the data, but RAM is able to both read and write the data so it is responsible from storing the data. That's why the content of ROM will not be saved, but the data in RAM will be saved to the memory location specified when the write operation is performed. And their address bit is different as well. ROM has a 10-bit address length where RAM has 8-bit.

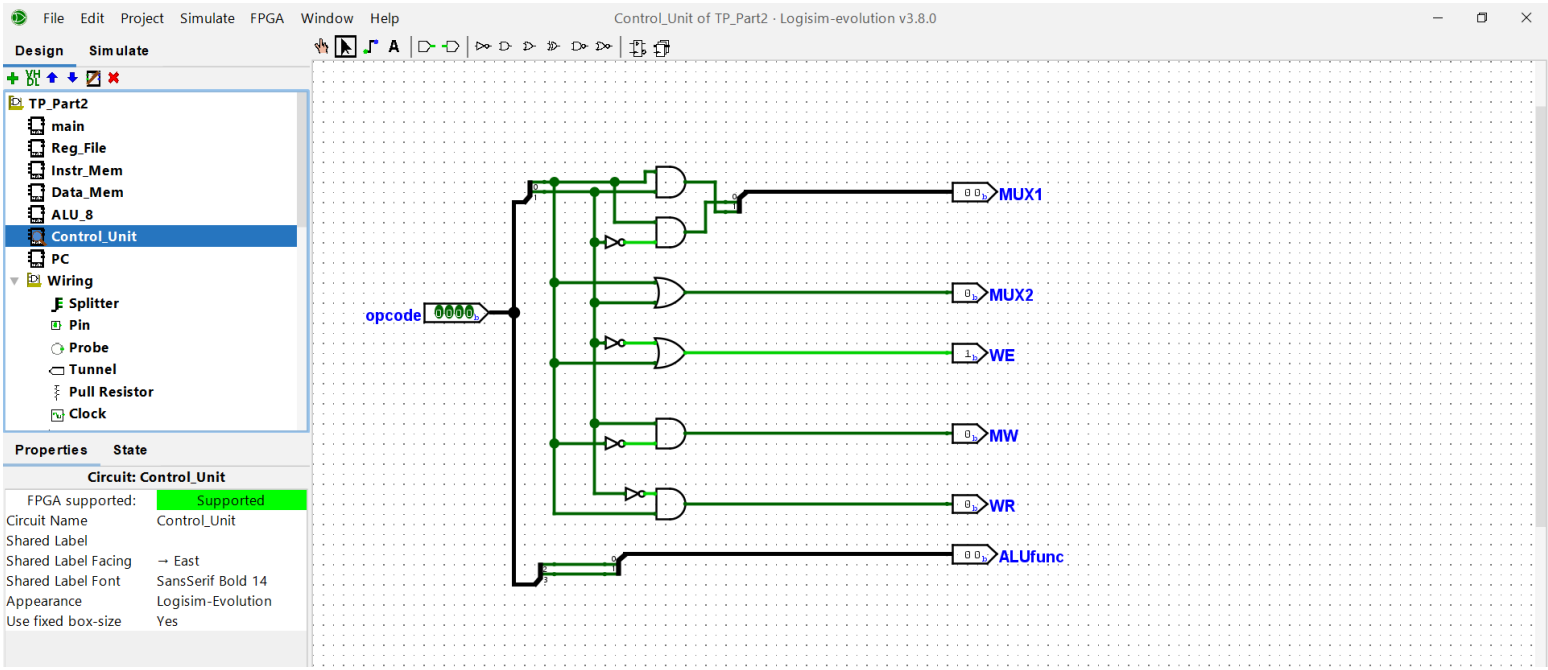

Figure 4: Control Unit

In the figure above, Figure 4, the Control Unit implementation is shown. The input to this unit, which is opcode, is coming from the Instruction Memory to generate all the necessary control bits for other units. MUX1 and MUX2 are the control bits for the MUXes used while connecting all the units as a whole CPU; WE is a control bit for the Register File; MW and WR are the control bits to read and write to the Data Memory; and ALUfunc is a control bit for the ALU.

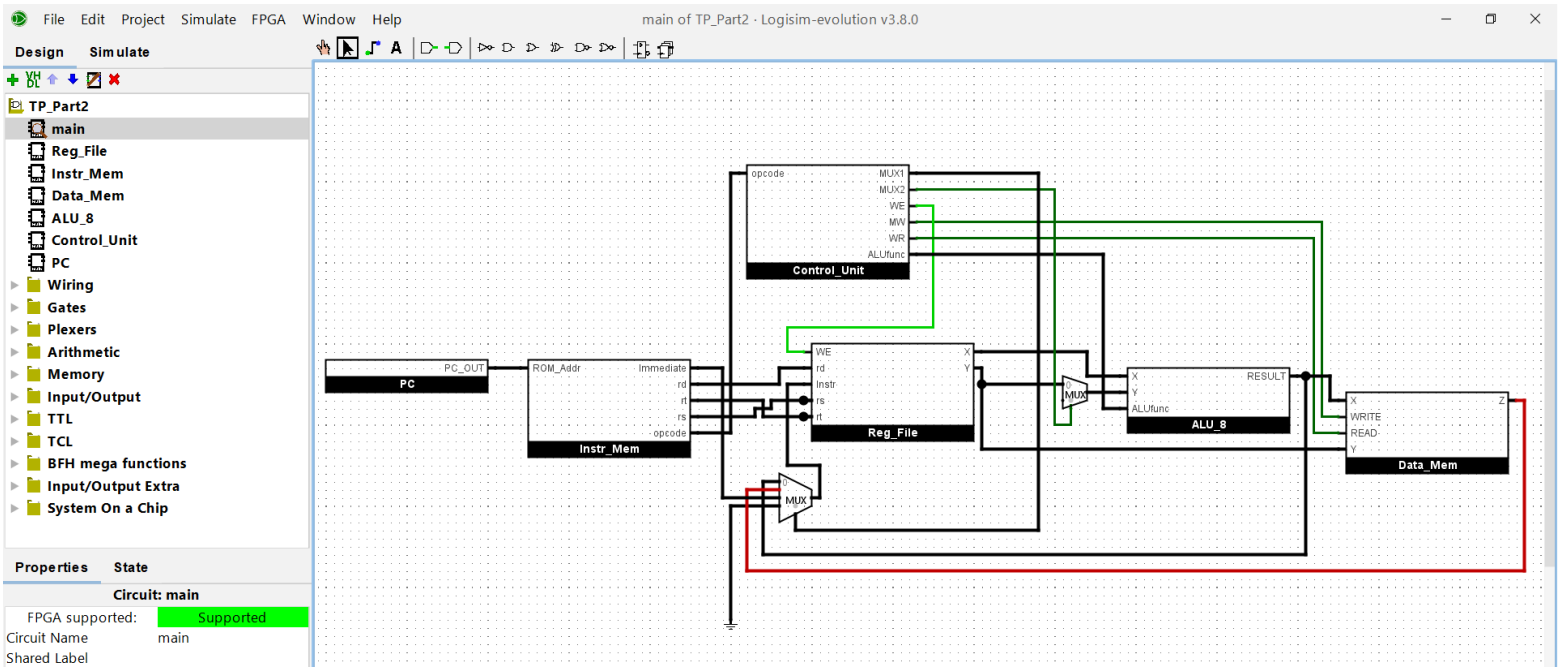

Figure 5: The Whole System - CPU Circuit Design

As shown in the Figure 5 above, all the units created are connected to each other here like in the schematic provided to create the CPU design. Two extra MUXes are used for the system to perform as expected.

To test the file uploaded, the content of it is loaded to ROM (the instruction memory).
The result for each unit and their outputs after the execution are shown with the figures below:

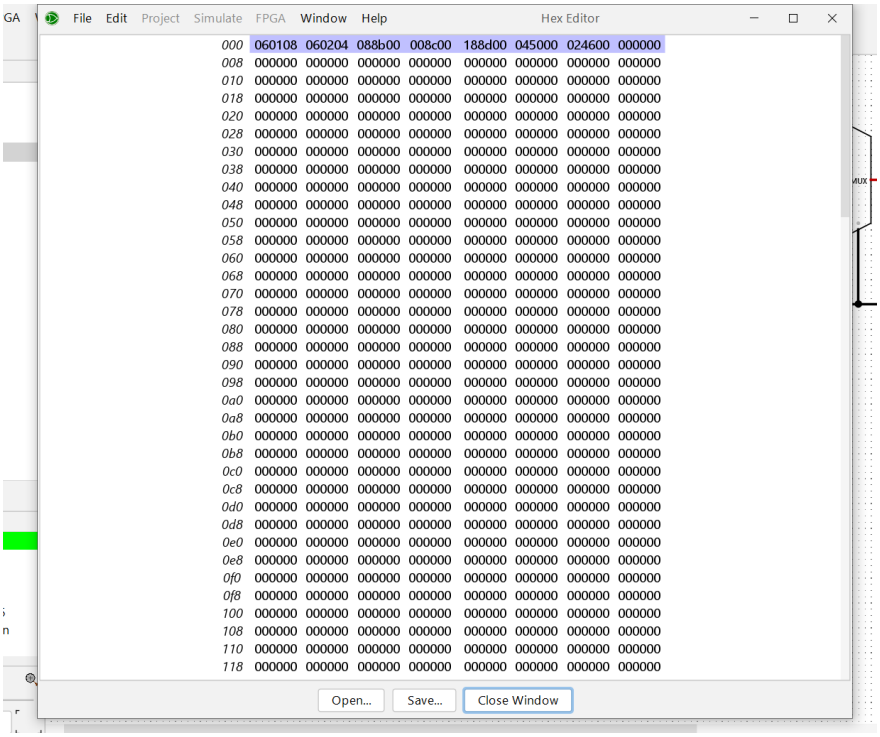

Figure 6: Test File Content to Load to the Instruction Memory (ROM)

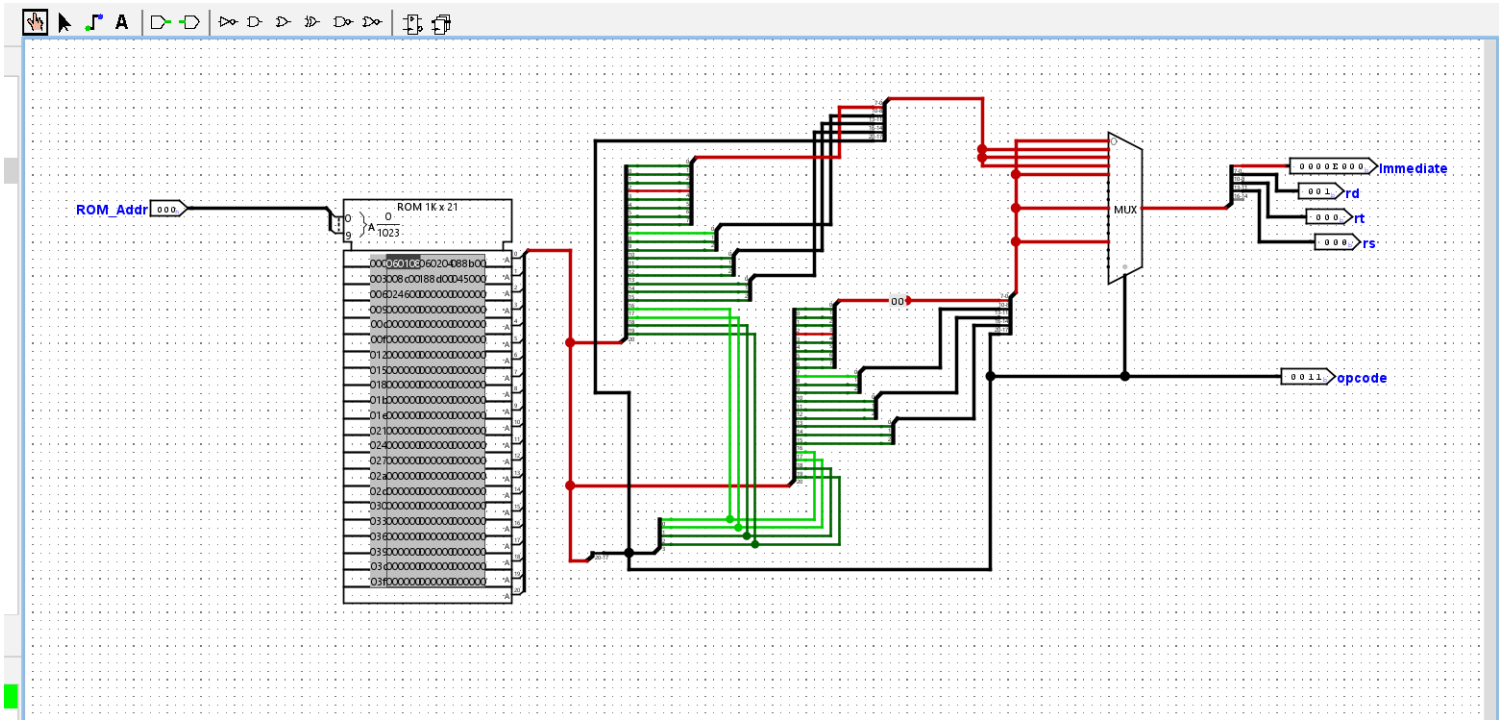

Figure 7: Outputs and the Address When the PC (ROM_Addr) = 0

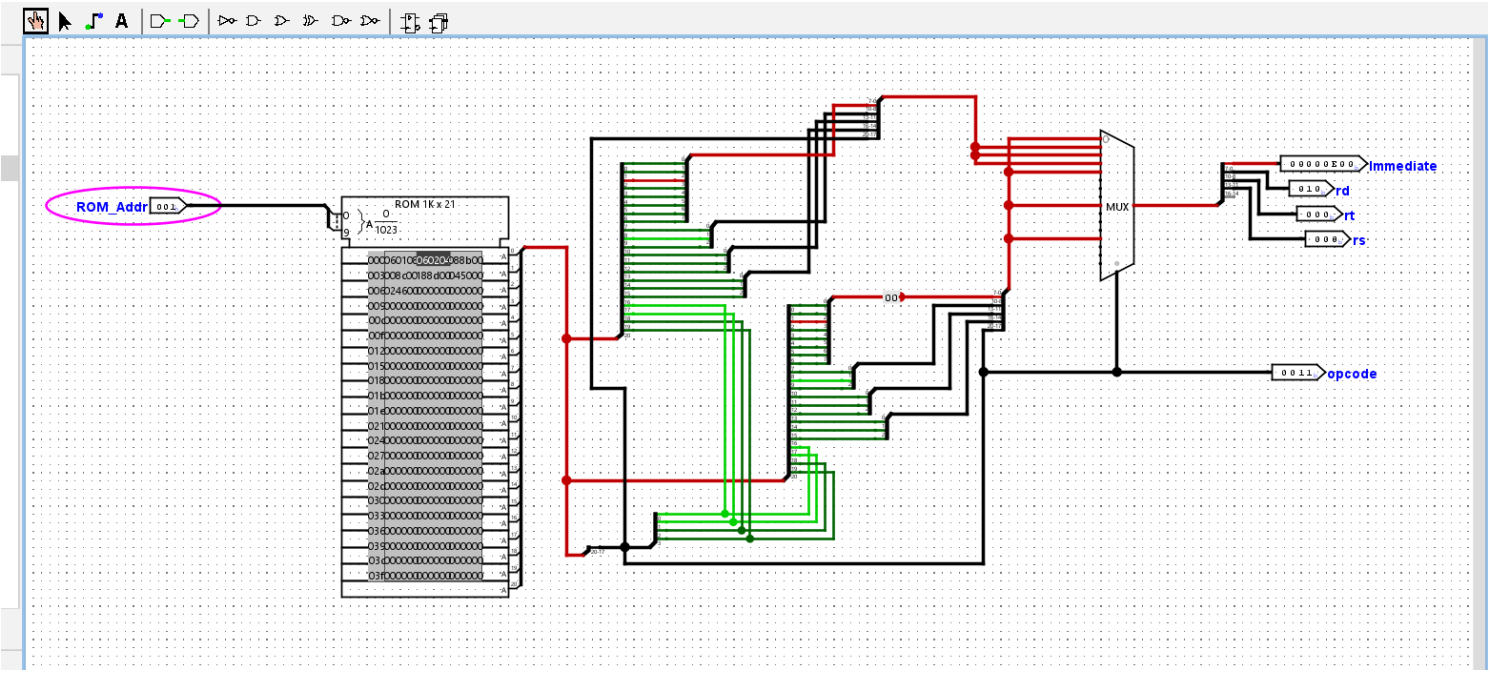

Figure 8: Outputs and the Address When the PC (ROM_Addr) = 1

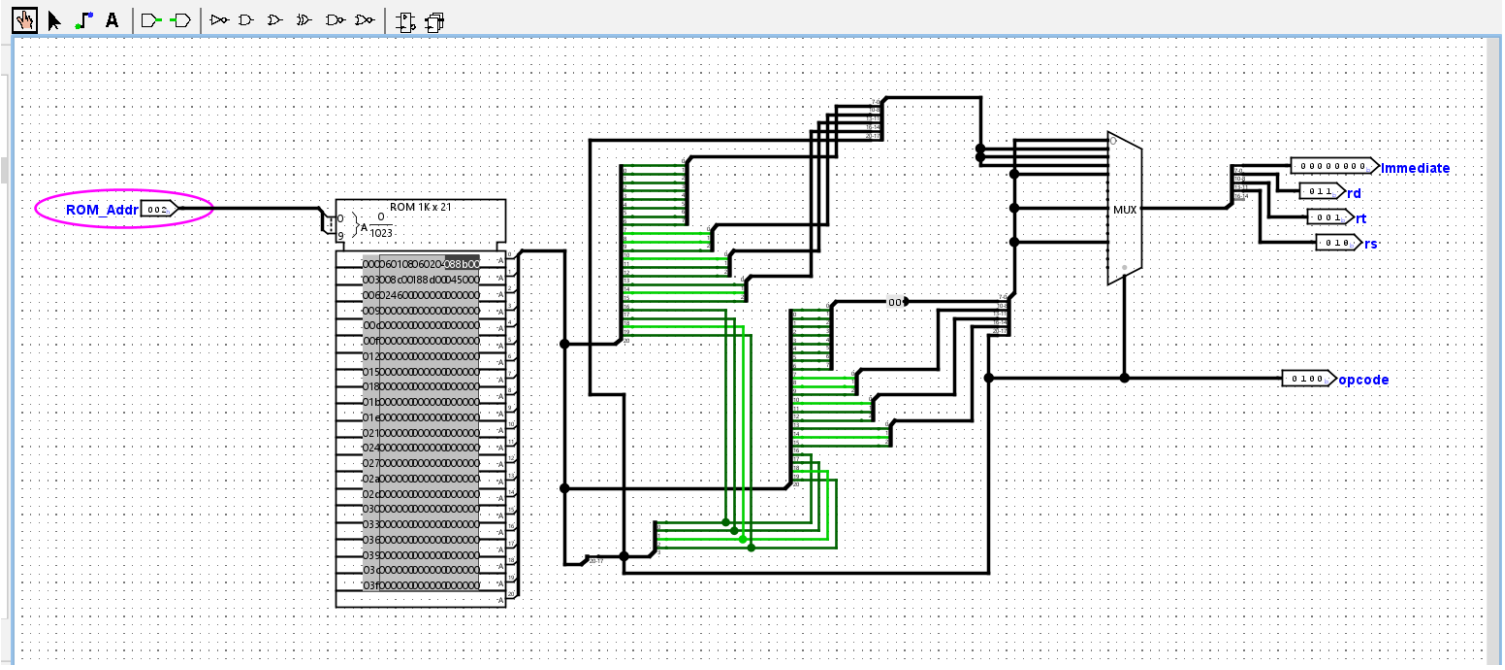

Figure 9: Outputs and the Address When the PC (ROM_Addr) = 2

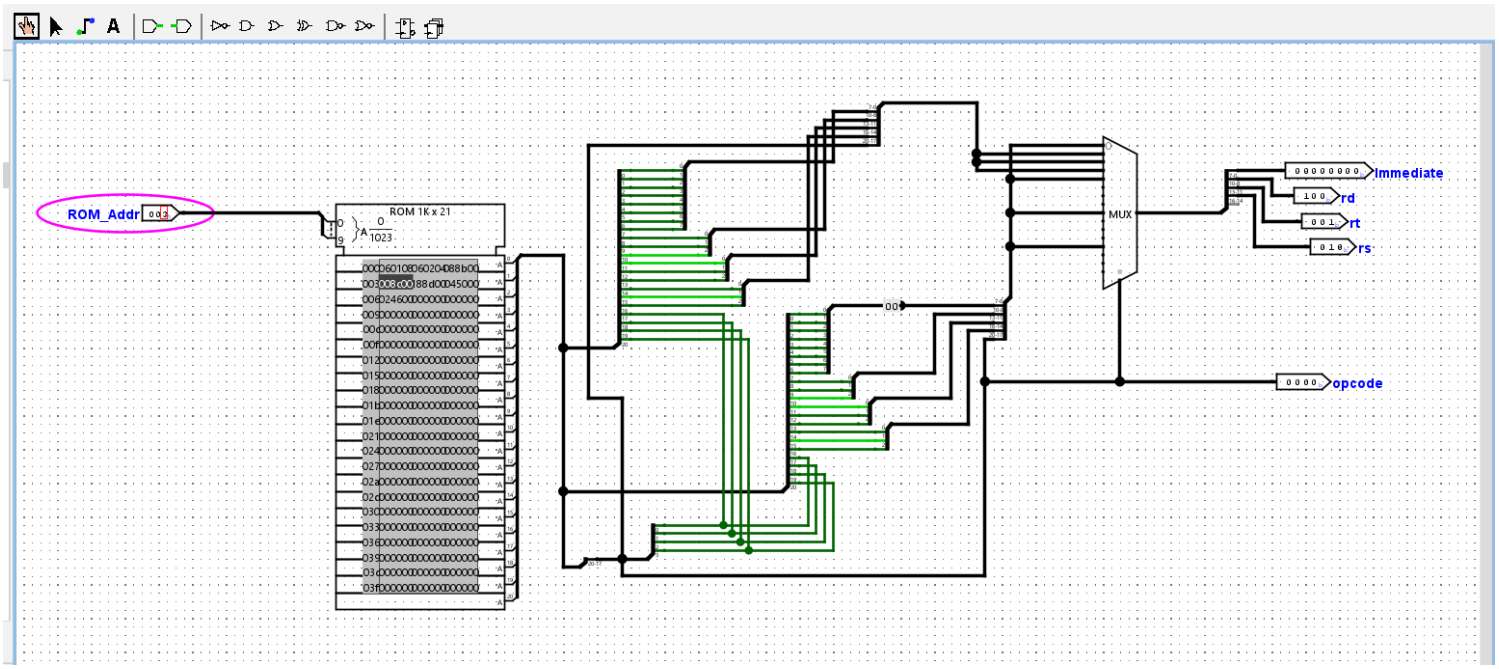

Figure 10: Outputs and the Address When the PC (ROM_Addr) = 3

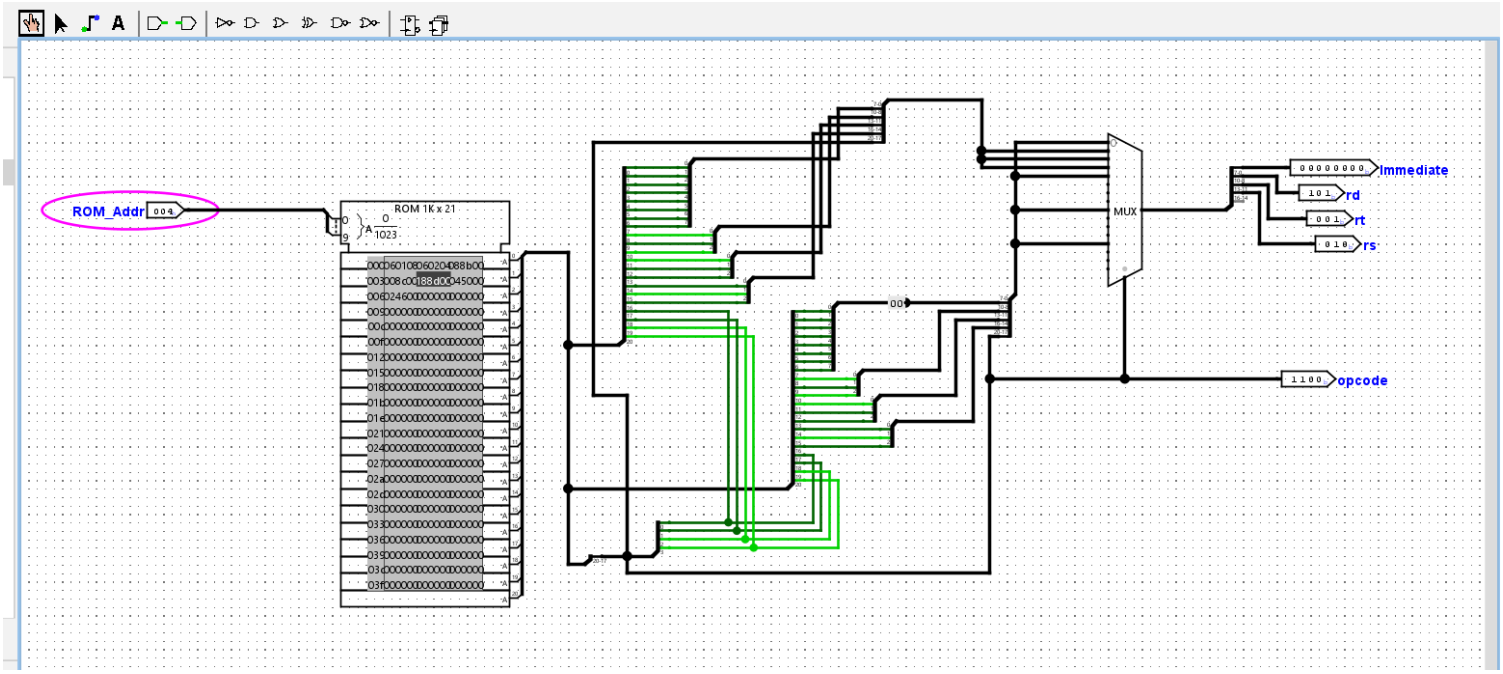

Figure 11: Outputs and the Address When the PC (ROM_Addr) = 4

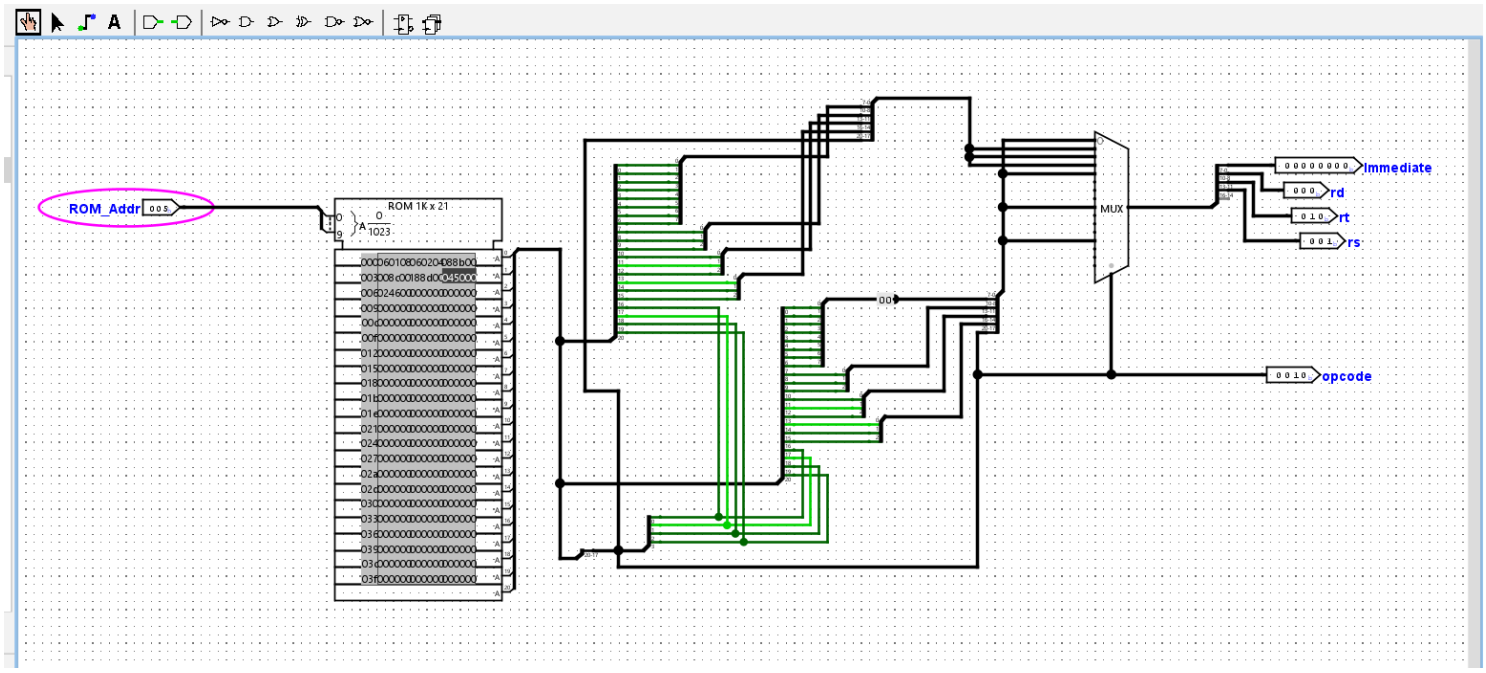

Figure 12: Outputs and the Address When the PC (ROM_Addr) = 5

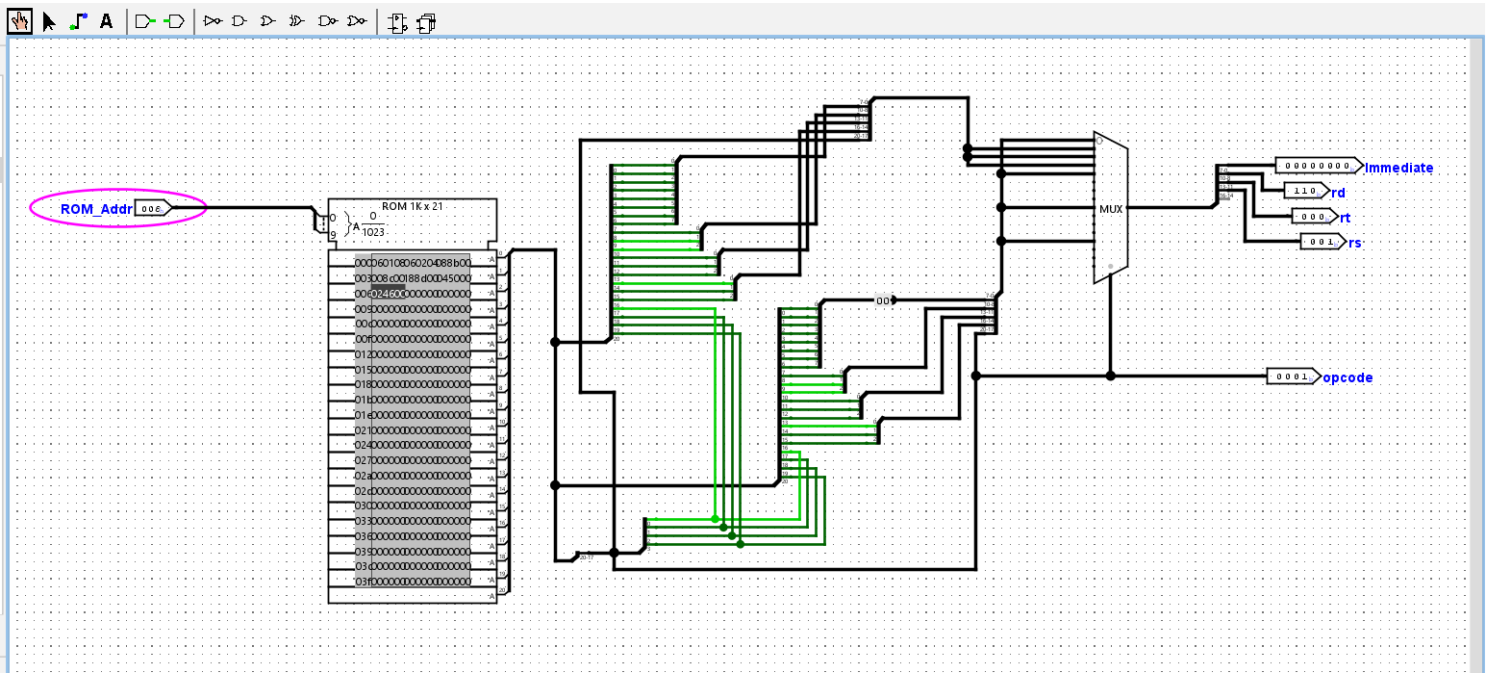

Figure 13: Outputs and the Address When the PC (ROM_Addr) = 6

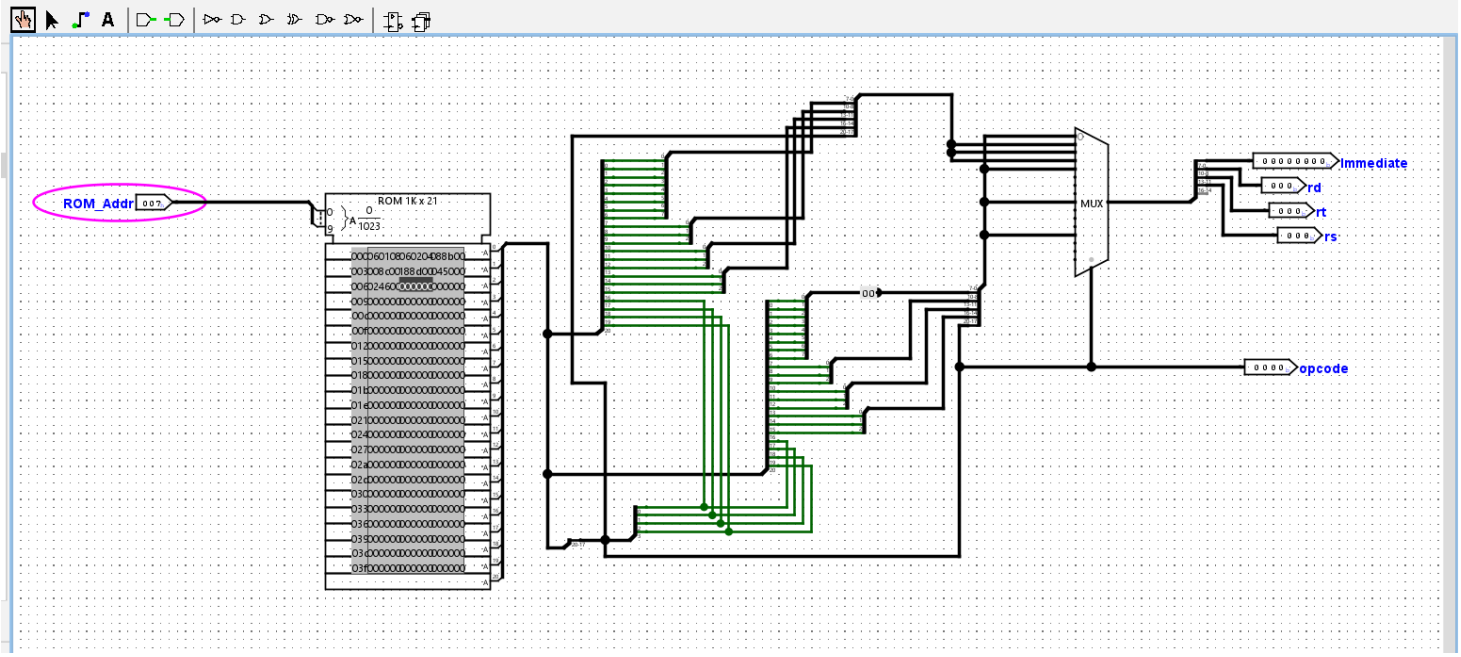

Figure 14: Outputs and the Address When the PC (ROM_Addr) = 7

The content of each unit and their corresponding outputs are shown in the figures above after the execution. The test file content is loaded to ROM to be used for the further operations. It can also be seen from the counter (the address for the ROM) and the changes in the outputs that the units work perfectly fine. The function of the other units are discussed above in the related sections and it can be seen from the output that reading from/writing to the memory is done successfully.

The assembly code for the instructions are computed by using the structure given in the pdf and the binary conversion of these addresses and the code for the whole system is as follows:

060108: 0011 0000 0000 1000 0100 0

060204: 0011 0000 0001 0000 0010 0

088b00: 0100 0100 0101 1000 0000 0

008c00: 0000 0100 0110 0000 0000 0

188d00: 1100 0100 0110 1000 0000 0

045000: 0010 0010 1000 0000 0000 0

024600: 0001 0010 0011 0000 0000 0

So the assembly code is:

```
LI Rd, 0x08          ; Rd=0x01
LI Rd, 0x04          ; Rd=0x02
AND Rd, Rs, Rt       ; Rd=0x03, Rs=0x02, Rt=0x01
ADD Rd, Rs, Rt       ; Rd=0x04, Rs=0x02, Rt=0x01
ROTL Rd, Rs, Rt      ; Rd=0x05, Rs=0x02, Rt=0x01
ST Rt, 0(Rs)         ; Rt=0x02, Rs=0x01
LD Rd, 0(Rs)         ; Rd=0x06, Rs=0x01
```