**CS 322 Lab-9**

Name: P. V. Sriram

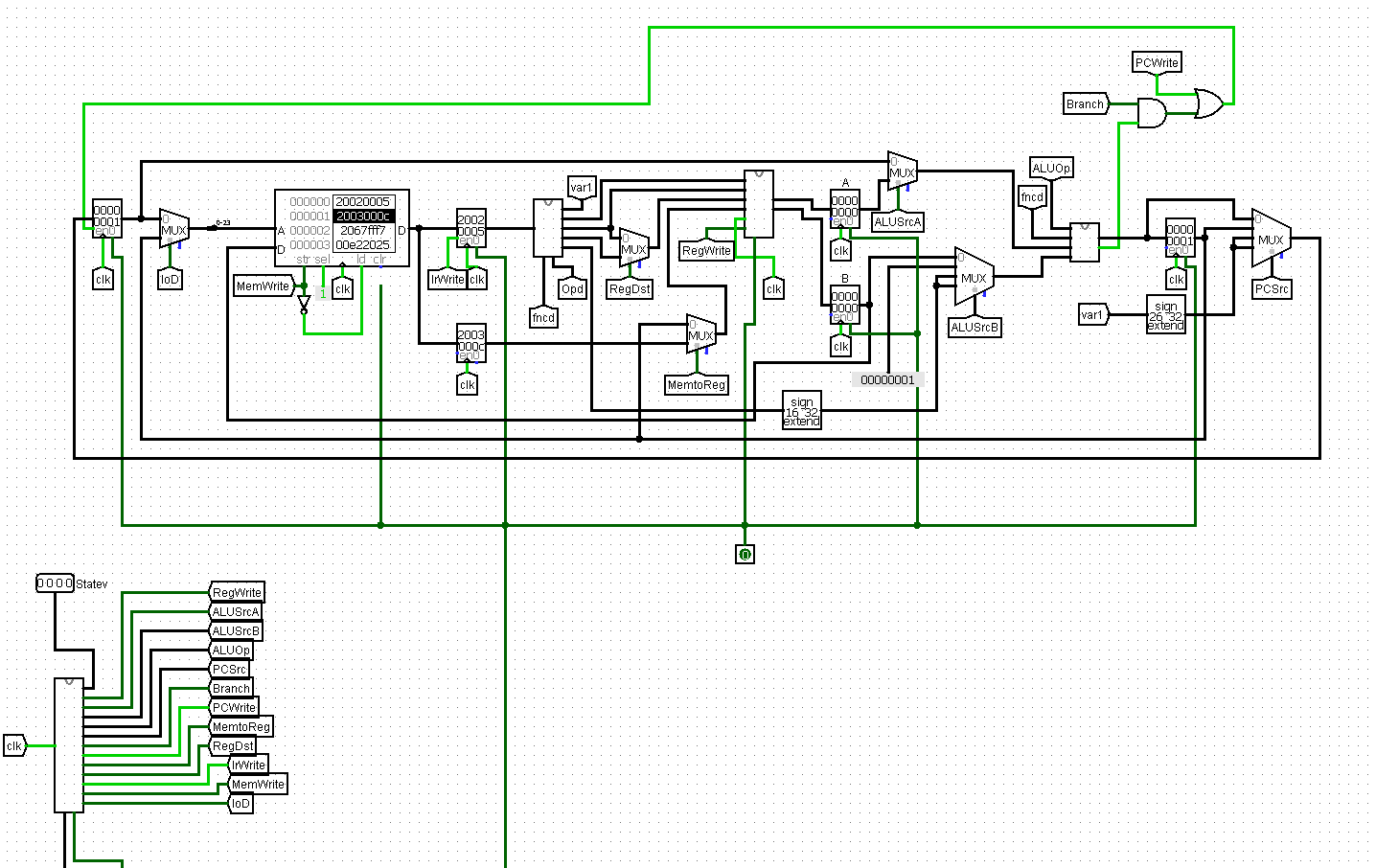
Roll No.: 1801CS37

# Task-1

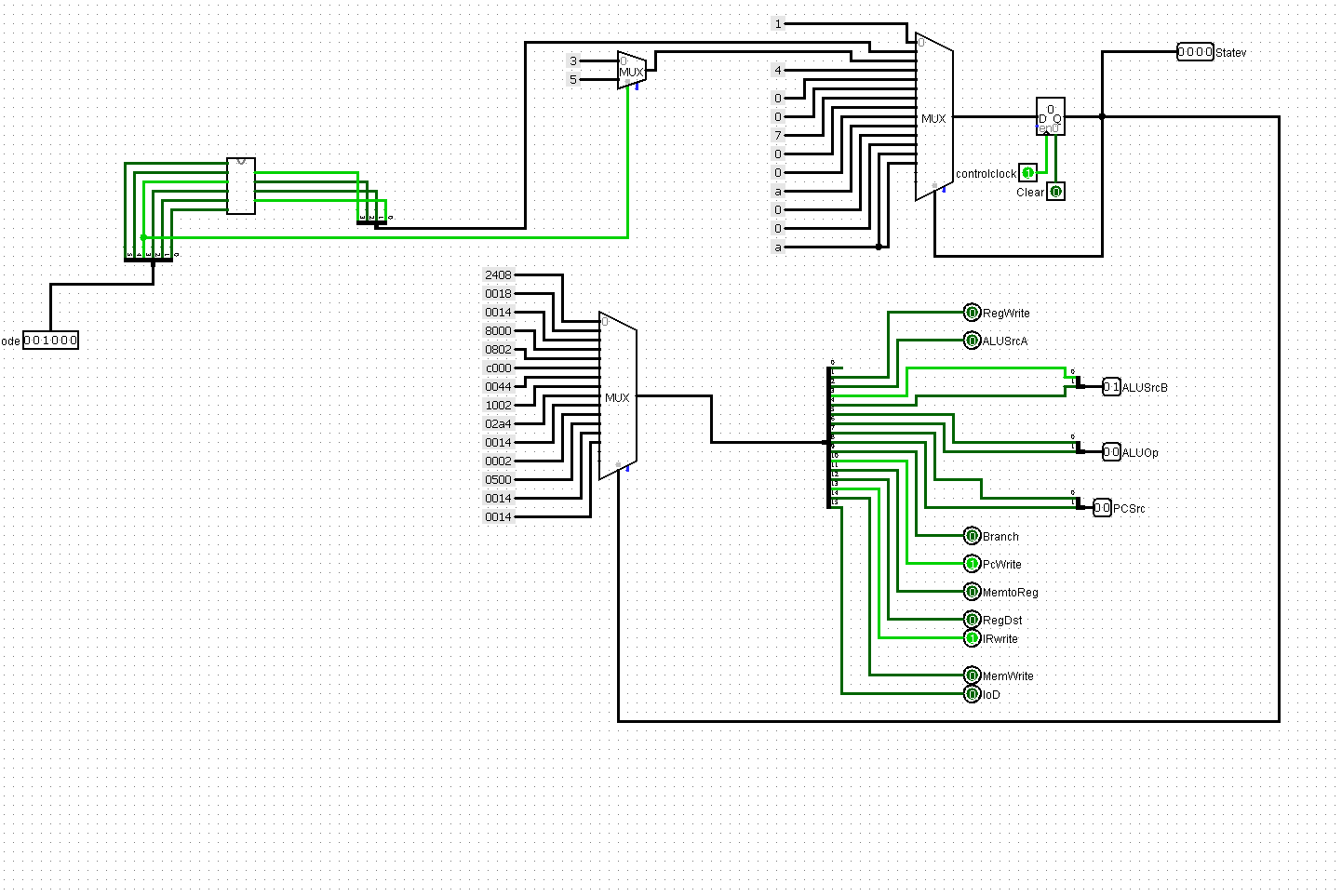
**Study the given multi‐cycle implementation of the processor, and identity error in the**

**design (if any)**

***MIPS Processor***



***Control Unit***

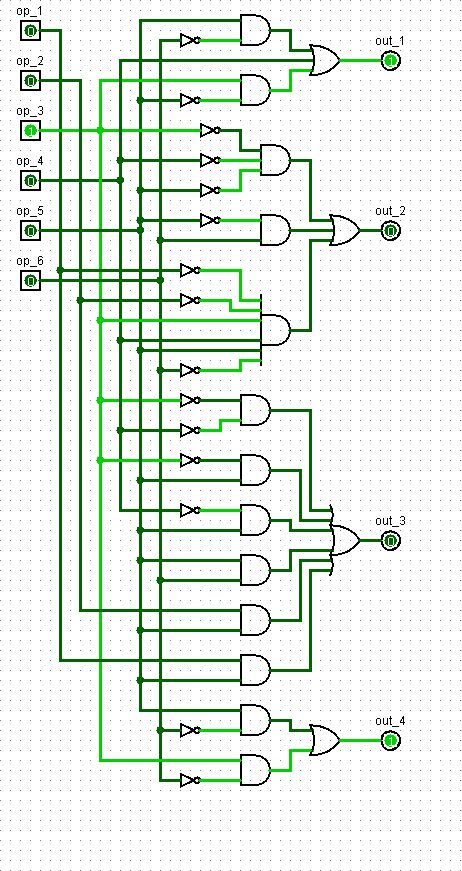


**Table**

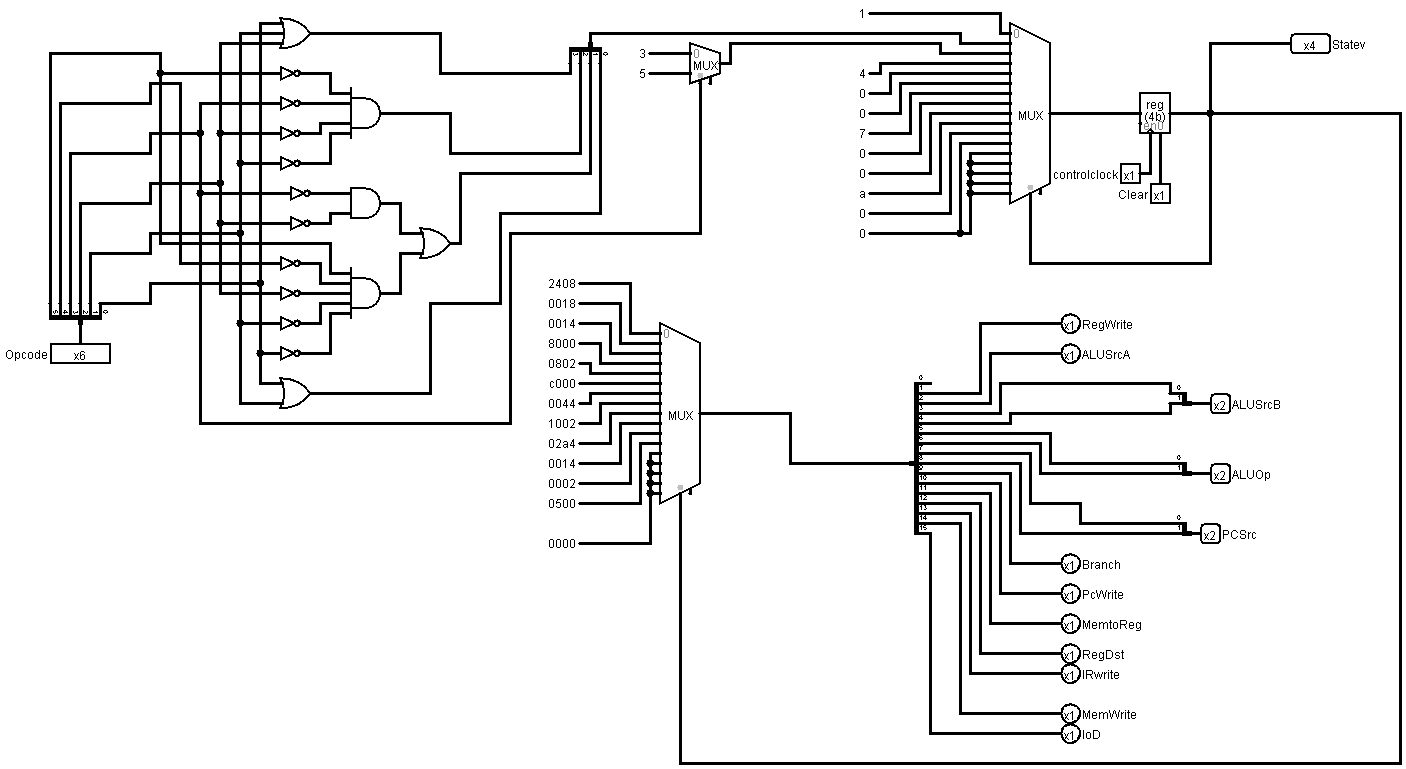
|  |  |  |
| --- | --- | --- |
| Instruction | Opcode | State Mapping |
| J | 000010 | 1011 |
| ADDI | 001000 | 1001 |
| ORI | 001101 | 1100 |
| BEQ | 000100 | 1000 |
| R-Type | 000000 | 0110 |
| LW | 100011 | 0010 |
| SW | 101011 | 0010 |
| ORI | 001101 | 1100 |
| XORI | 001110 | 1101 |

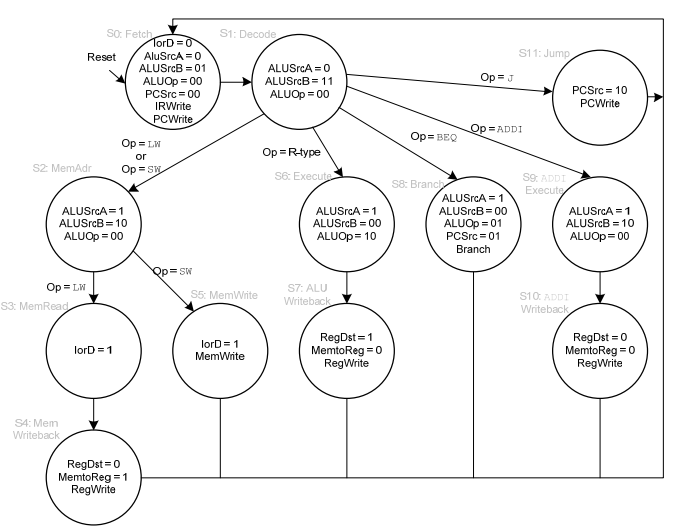
The above table is the right mapping of opcodes to fsm states and using this table we generate the following circuit

***Logic to map the opcodes to states***



In the initial circuit, the logic which maps the opcodes to the corresponding sates is wrong.



I have replaced the circuit according to the following mapping of opcodes to states.

# Task-2

**Study the given multi‐cycle implementation of the processor and test using your own**

**test program (for eg. Sum of 5 numbers using basic instruction; create a new mem.dat)**

**Code:**

Addi $1, $0, 0

Addi $2, $0, 5

Addi $3, $0, 1

Loop: beq $2, $0, exit

Add $1, $1, $2

Sub $2, $2, $3

J loop

Exit: sw, $1, 80($0)

**Algorithm:**

First initialize the registers with suitable values. Here $1 acts as the accumulator (value 0 initially), $2 acts as the counter (value 5 here), $3 is incremented (value 1).

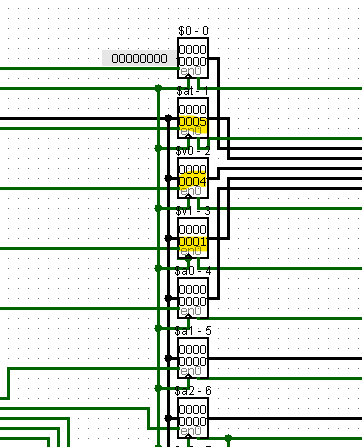
Next, we go into loop, check if the counter is done or not. Or else we add the counter value into the accumulator and go to the next iteration.

**Machine Language:**

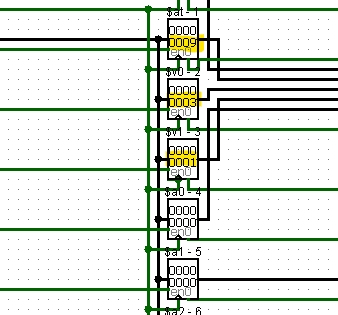
|  |  |
| --- | --- |
| Hexa-decimal | Binary |
| 0x20010000 | 0010 0000 0000 0001 0000 0000 0000 0000 |
| 0x20020005 | 0010 0000 0000 0010 0000 0000 0000 0101 |
| 0x20030001 | 0010 0000 0000 0011 0000 0000 0000 0001 |
| 0x10400003 | 0001 0000 0100 0000 0000 0000 0000 0110 |
| 0x00220820 | 0000 0000 0010 0010 0000 1000 0010 0000 |
| 0x00431022 | 0000 0000 0100 0011 0001 0000 0010 0010 |
| 0x08000003 | 0000 1000 0000 0000 0000 0000 0000 0011 |
| 0xac010050 | 1010 1100 0000 0001 0000 0000 0101 0000 |

**Working:**

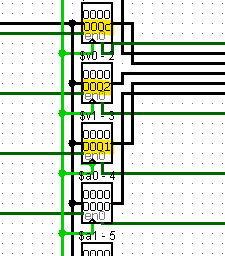
1st iteration, we add 5 to $1



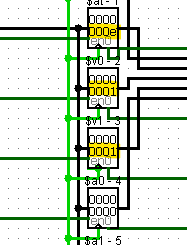
2nd iteration, we add 4 to $1



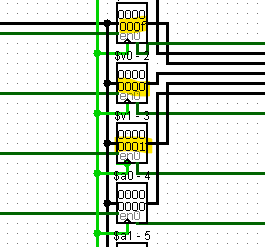
3rd iteration, we add 3 to $1



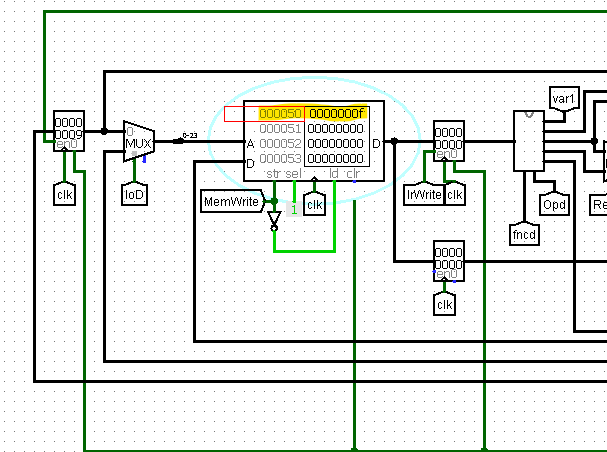
4th iteration, we add 2 to $1



5th iteration, we add 1 to $1



Lastly, we store the value 0f to 80($0). We can see that 0x50 contains 0f

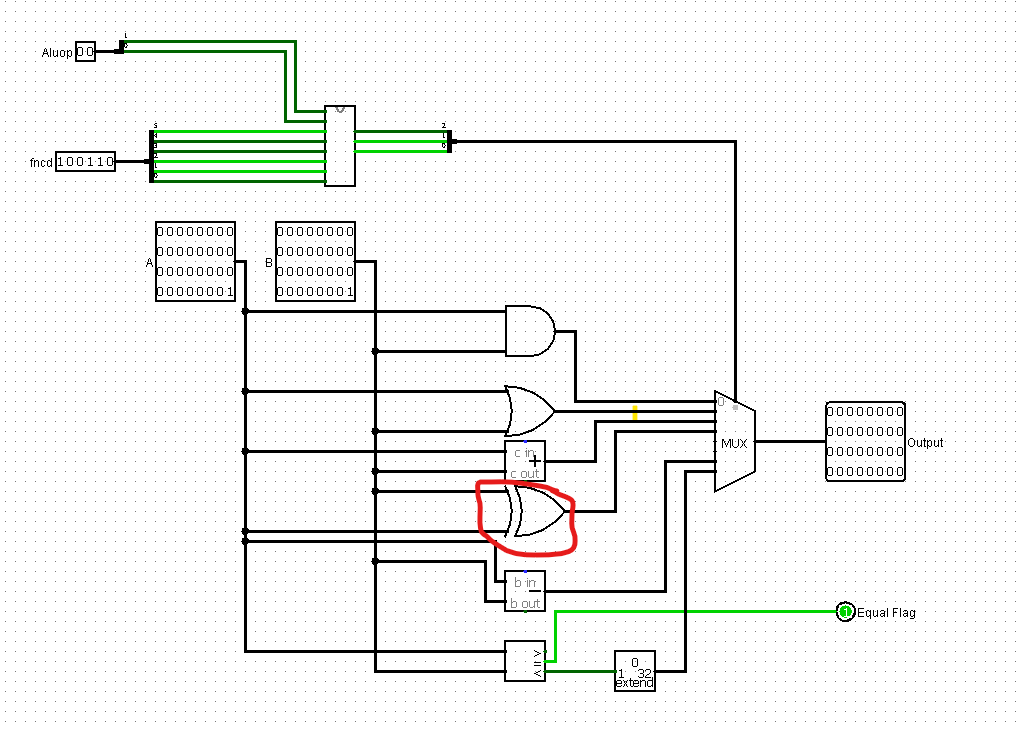


# Task-3

**Add one new instruction to the given architecture and test using new test program**

For this task I decided to add XOR operation. We add new XOR Gate in the ALU unit and use it in operations.

**ALU:**

****

**Code:**

ADDI $1, $0, 5

XOR $1, $0, $2

**Machine Code:**

|  |  |
| --- | --- |
| Instruction | Machine Code |
| ADDI $2, $0, 5 | 0x20020005 |
| XOR $1, $0, $2 | 0x00020826 |

**Result**

We can see that at the end 5 is stored in the register

