



MICROPROCESSOR

PROJECT REPORT

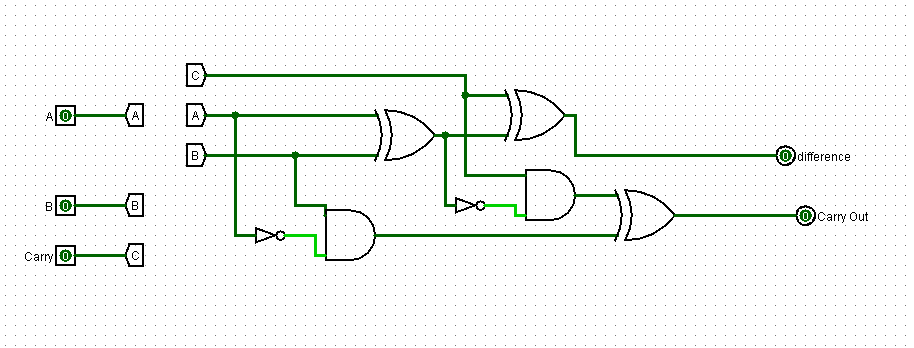
# PART 1: ALU Design

In this part, ALU was implemented with the opcodes in the table below.

|  |  |  |
| --- | --- | --- |
| **OPCODE** | **Instruction** | **Operation** |
| **.0000** | sub | C=A-B |
| **.0001** | add | C=A+B |
| **.0010** | sll (shift left logical) | C = A << B |
| **.0011** | mult | C=A\*B |
| **.0100** | or | C=A|B |
| **.0101** | xor | C=A^B |
| **.0110** | and | C=A&B |
| **.0111** | slt (set less than) | C = 0 IF (A >= B); 1 IF (A < B) |
| **.1000** | sra (shift right arithmetic) | C = A >>> B |
| **.1001** | srl (shift right logical) | C = A >> B |
| **.1010** | Combiner |  |
| **.1011** | equal? | C = (? = A B) |
| **.1100** | not Equal? | C = ~ (? = A B) |

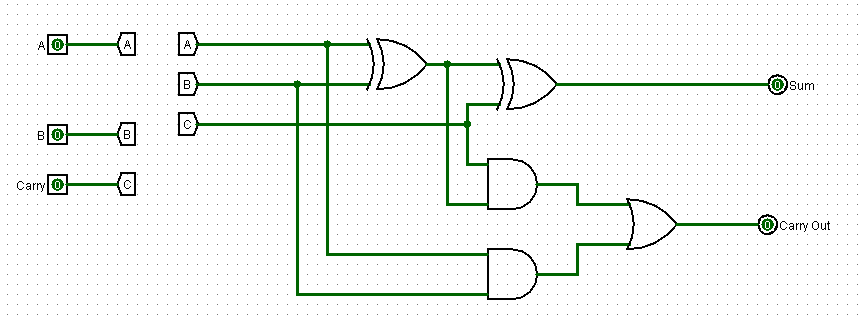
# Subtractor:

For sub instruction, firstly 1-bit subtractor was implemented. Then, 32-bit subtractor was implemented using 1-bit subtractor. One-bit subtractor is shown below.



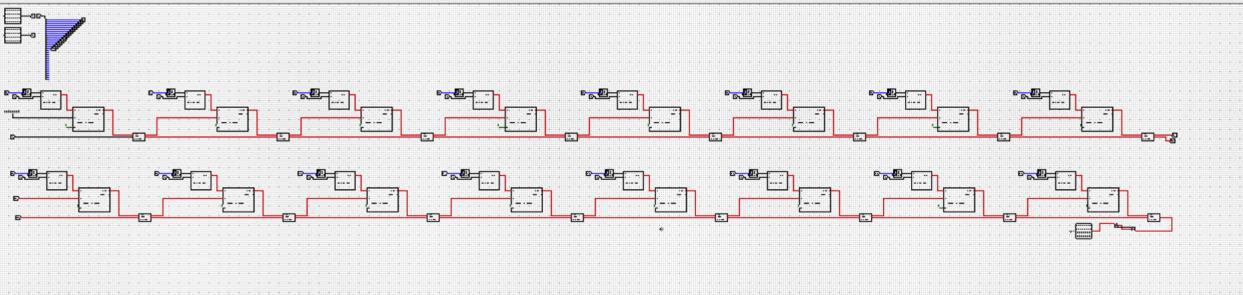
# Adder

For add instruction first we implemented 1 bit adder. Then we implemented 32-bit adder using this 1-bit adder. One bit adder is shown below.



# Multiplier

Multiplier is shown below.



# XOR, OR, AND

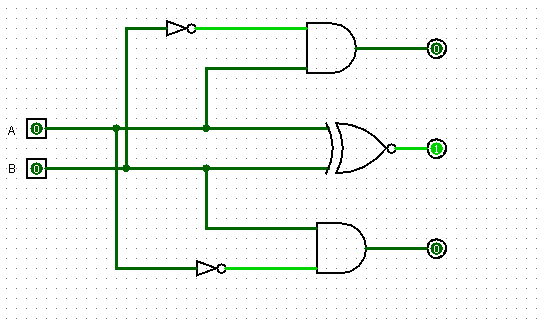
For ‘and’, ’or’ and ‘xor’, these gates were implemented to all bits of inputs.

# Shifter

For sl, sea and srl, barrel shifter was used.

# Set Less Than (SLT)

For SLT operation first 1-bit SLT implemented than 32-bit SLT implemented. After several logical operations SLT only returns 1 or 0.



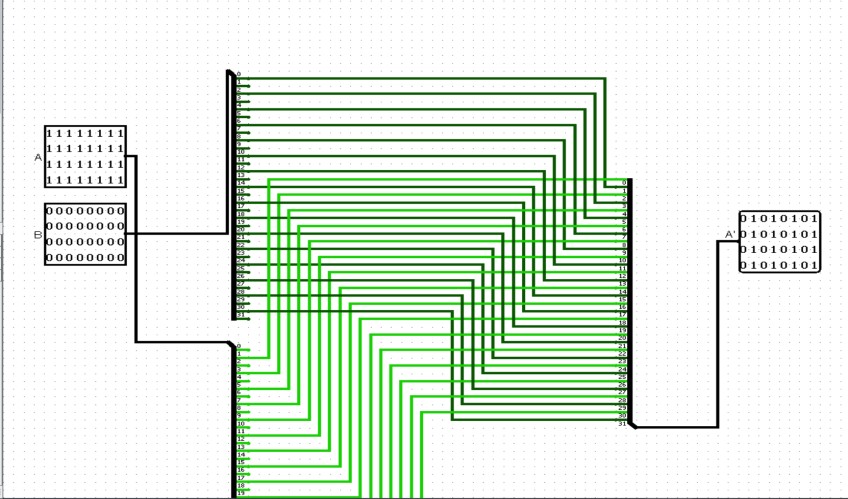
# SPECIAL OPERATIONS IN ALU:

1. **Combiner**

It combines two inputs and gives it as an output. For example:

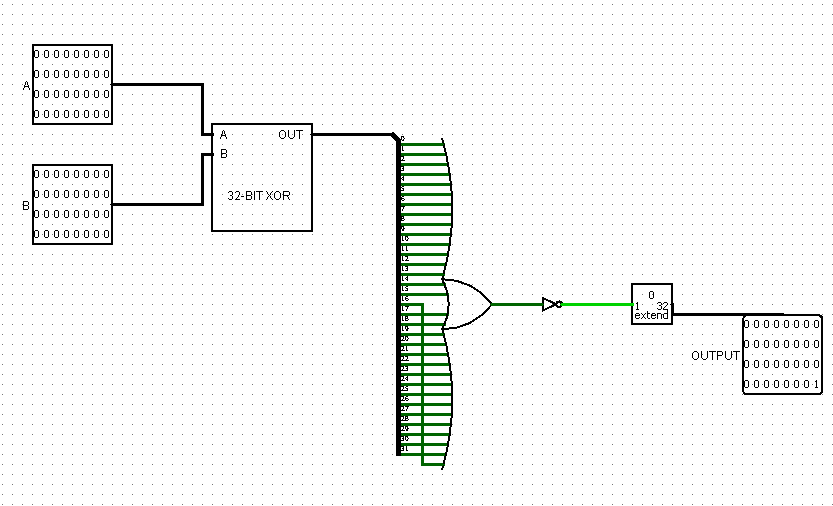
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input 1: | 0000 | 0000 0000 0000 | 0000 | 0000 | 0000 | 0000 |
| Input 2: | 1111 | 1111 1111 1111 | 1111 | 1111 | 1111 | 1111 |
| Output: | 0101 | 0101 0101 0101 | 0101 | 0101 | 0101 | 0101 |

Design of this is shown below.



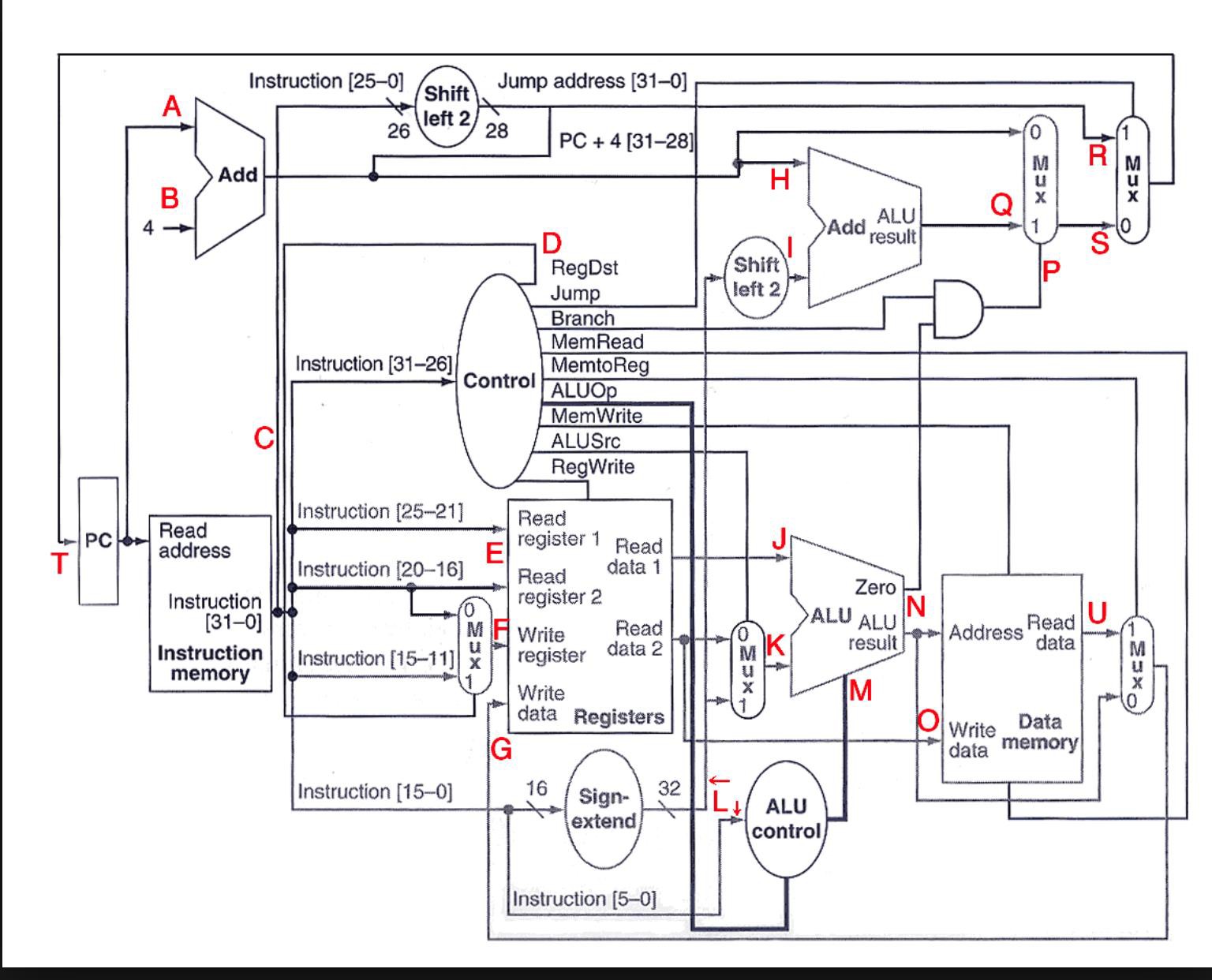
# Equal? and Not Equal?

Equal? returns 1 if inputs are equal. Its implementation is shown below. Not Equal? returns 1 if inputs are not equal.

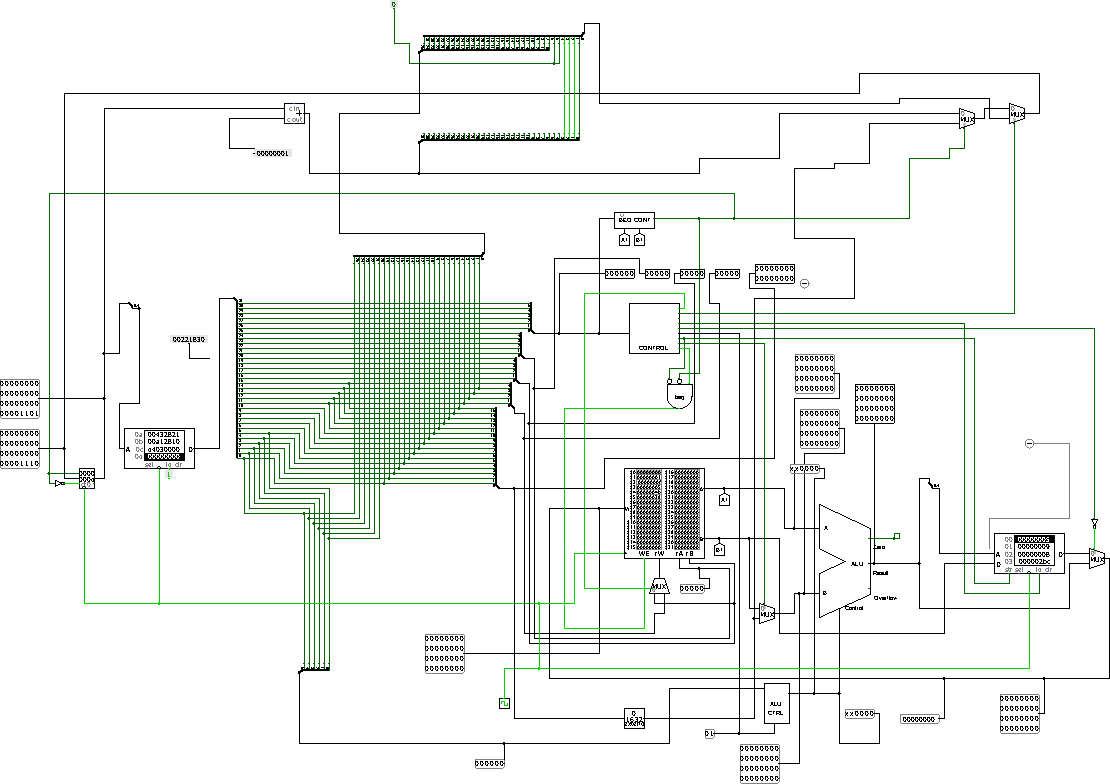


# PART 2: Single Cycle Processor Design - SCP

In this part, single cycle processor was designed. While processor is being designed, the figure below taken from the lecture book was used as a reference.



Our Design:



Our controller and processor were designed according to below operations.

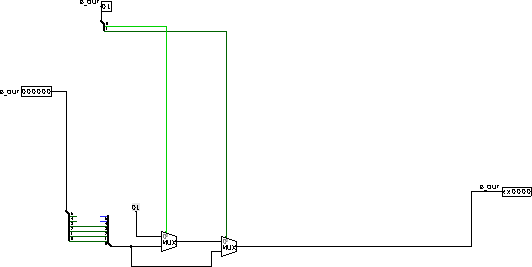
|  |  |  |
| --- | --- | --- |
| **OPCODE** | **Instruction** | **Operation** |
| **000000** | ALU | Arithmetic OP |
| **100011** | lw | Load word from memory into register |
| **001011** | sw | Store word from register to memory |
| **001100** | beq | Branch to a label if two registers are equal |
| **001101** | bne | Branch to a label if two registers are not equal |
| **001110** | j | Unconditional branch |
| **000000** | CUSTOM | CUSTOM METHOD IS IN ALU |

As custom instructions, it was designed Combiner, Equal? and Not Equal? Since they were put in the ALU, they were explained ALU part of this report.

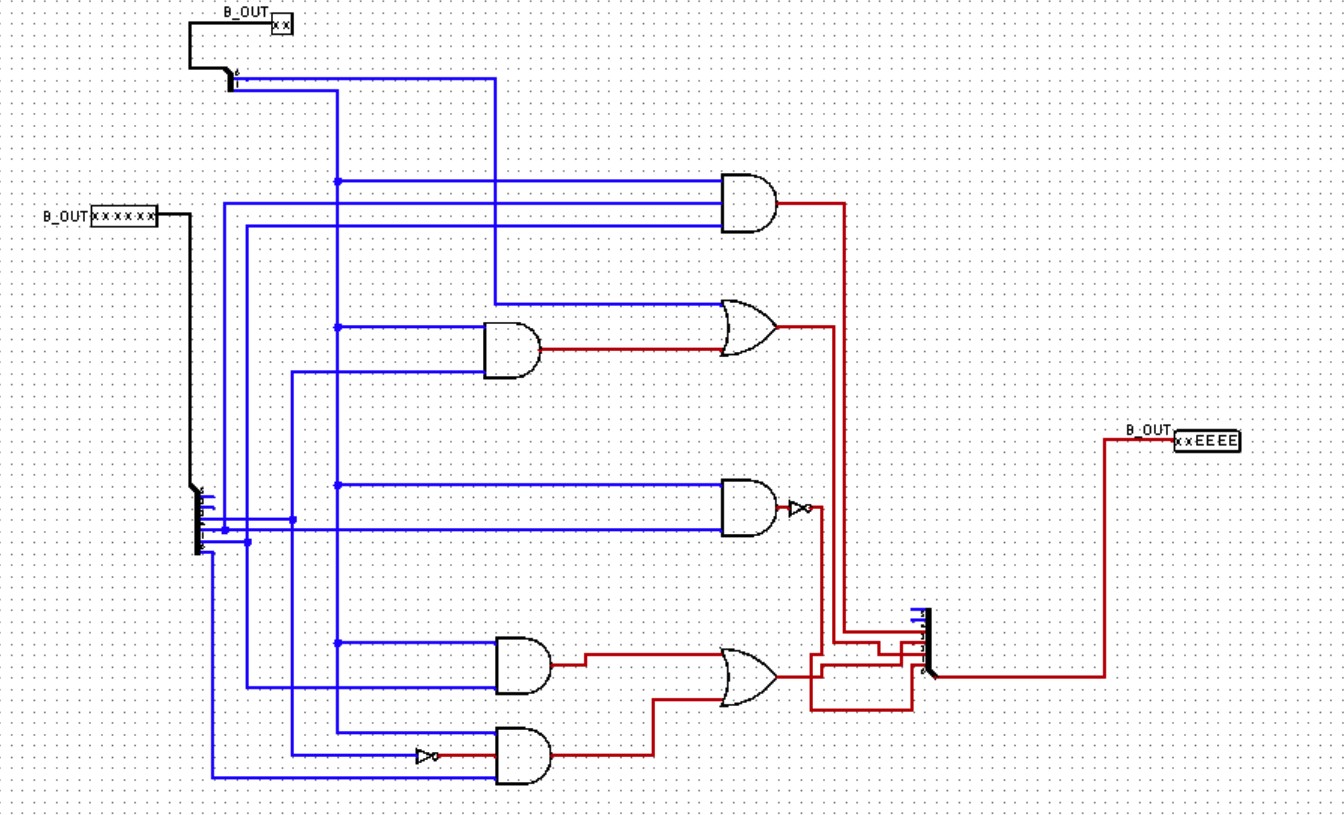
For implementation of the single cycle processor, ALU controller, beq controller and Main Controller were implemented.

# ALU Controller:

ALU Controller is designed as fallows.

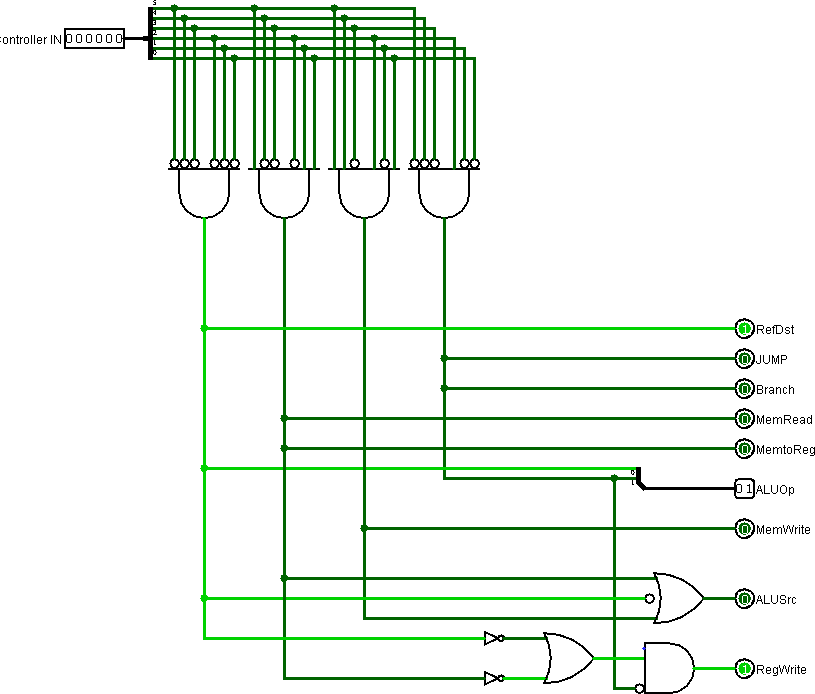


At the beginning of the project, ALU controller was designed as below. It was found only this design for ALU controller which gives 4-bit output. (First two bits are not important). It did not work correctly. Thus, the one above was used. This is very simple, but it works correctly for our usage.



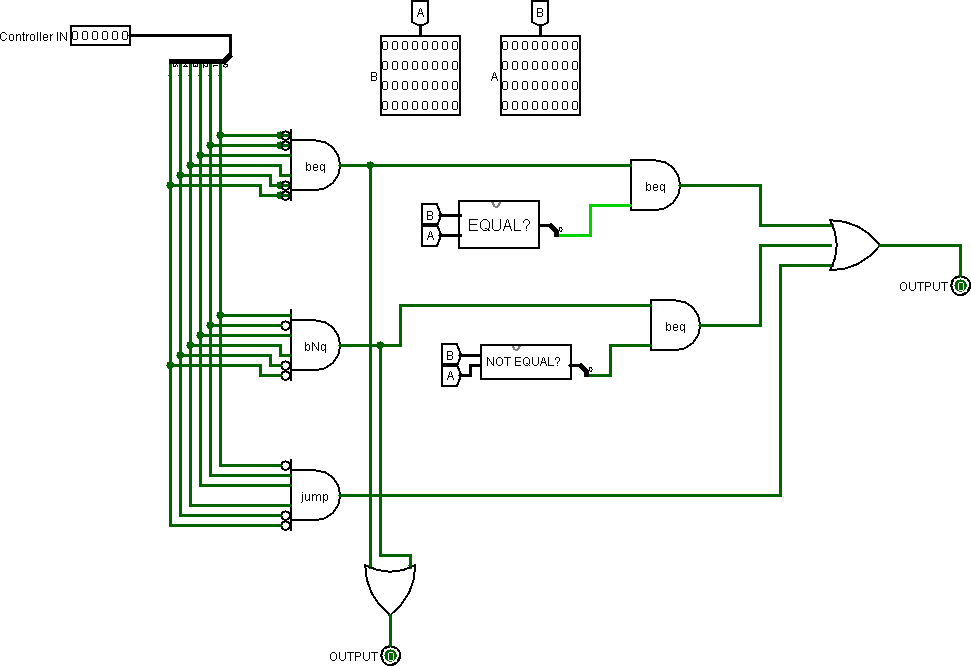
# Main Controller

Main controller was designed as below.



# Beq Controller

This additional controller was design to handle jump, beq and bne. This box returns true only if there will be a jump. Equal and Not Equal boxes were designed in ALU.



# PART 3: TEST

In this part, two sample code was implemented.

All instructions in this project are working. However, in test part, only lw, sw, J, add, addi, mult, sll, slt and beq were implemented. These are almost all of the instructors. Test codes are as fallows.

# Test Case

In this part, there is a rectangle.

Length of the rectangle was loaded from memory to a $3.

$1 is set to 1.

Hight of the rectangle is written to $2 as 15.

Then, area of the rectangle is calculated by multiplying values in $2 and $3. Area of the rectangle is written to $4.

Also, sum of the values in $2 and $3 is calculated and written to $5. Then, value in $5 is shifted left to obtain circumference of the rectangle.

|  |  |
| --- | --- |
| MIPS LANGUAGE | HEXADECIMAL FORM |
| J 6  -- no operation  -- no operation  -- no operation  -- no operation  -- no operation lw $3, 16($2) addi $1 $0 1 addi $2 $0 15 mult $4 $3 $2 add $5 $2 $3  sll $5 $5 $1  sw $3, 0($0) | 38000006  00000000  00000000  00000000  00000000  00000000  8c430004  20010001  2002000F  00622030  00432821  00A12810 D4030000 |

# Test Case

|  |  |  |
| --- | --- | --- |
| Java Version | MIPS LANGUAGE | HEXADECIMAL FORM |
|  | addi $1 $0 1 addi $2 $0 100 mult $4 $6 $6 add $5 $5 $4 addi $6 $6 1 stl $s3 $s2 $s5 beq $3 $1 2 | 20010001  20020064  00C62030  00A42821  20C60001  00451838  30610002 |