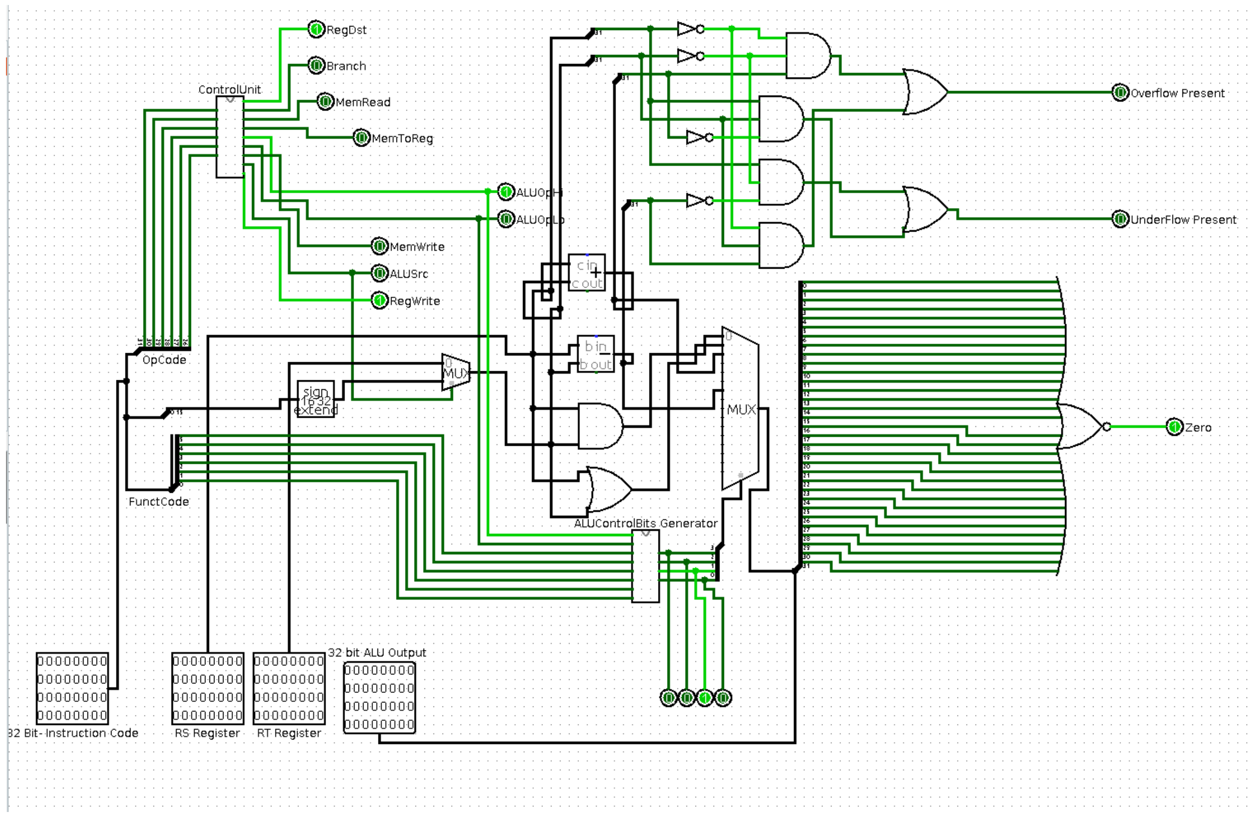
ASSIGNMENT 7: COMPUTER ARCHITECTURE

Name: Yogita Yogesh Mundankar (B22CS068)

The assignment required to create ALU using logisim.

Main CIrcuit:



Input:

- A 32-bit binary MIPS instruction
- Two 32-bit values representing register contents, acting as `rs` and `rt` from the register file

Output:

- **Control Signals (9 total):**
  - `RegDst`, `ALUSrc`, `MemToReg`, `RegWrite`, `MemRead`, `MemWrite`, `Branch`, and `ALUOp` (split into `ALUOpHigh` and `ALUOpLow`)
  - ALU Control: 4-bit ALUControl signal
  - Result Values:
    - 32-bit ALU result
    - Zero bit, indicating a result of zero
    - Underflow and overflow flags for error checking

Control Path :

- The instruction code is processed through splitters to isolate key elements: the opcode, function bits, and the 16-bit immediate or offset value.
- The opcode is then utilized to create control signals, including the 2-bit ALUOp.
- The 16-bit immediate/offset undergoes sign extension, converting it into a 32-bit value.
- The ALUSrc signal directs whether the ALU's second operand comes from the Rt register or the extended immediate value, while the Rs register content is consistently used as the first operand.
- Combined ALUOp and function bits generate a 4-bit ALU control signal, allowing the ALU to execute one of four operations: ADD, SUB, AND, or OR.
- A NOR gate applied to all 32 bits of the ALU result generates the ZERO signal, which indicates an all-zero result when active.

Overflow and Underflow Conditions:

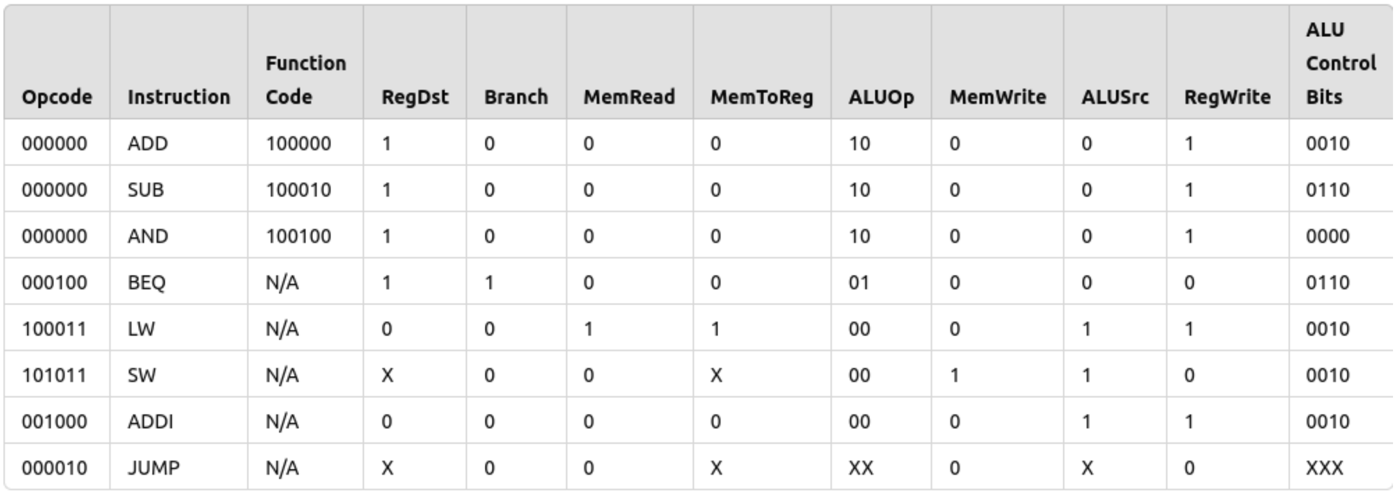
Overflow occurs if both inputs are negative but yield a positive result or if both inputs are positive but yield a negative result**.**
Underflow is detected when either a positive minuend with a negative subtrahend yields a negative result or a negative minuend with a positive subtrahend yields a positive result. The MSB (Most Significant Bit) is used to evaluate these cases.

The output section, located to the right, displays the status of overflow, underflow, zero, and the 32-bit ALU result.

Control Signals for R, I, and J Type Instructions: The table below details control signals as defined by opcode values, specifying configurations for R, I, and J-type instructions.

Verification of Control Signals: All control signals generated by the circuit have been cross-referenced with expected values for each instruction type, confirming proper operation across R, I, and J instructions.

The control signals generated for the given opcode values for R, I and J type instructions are documented in the table below:

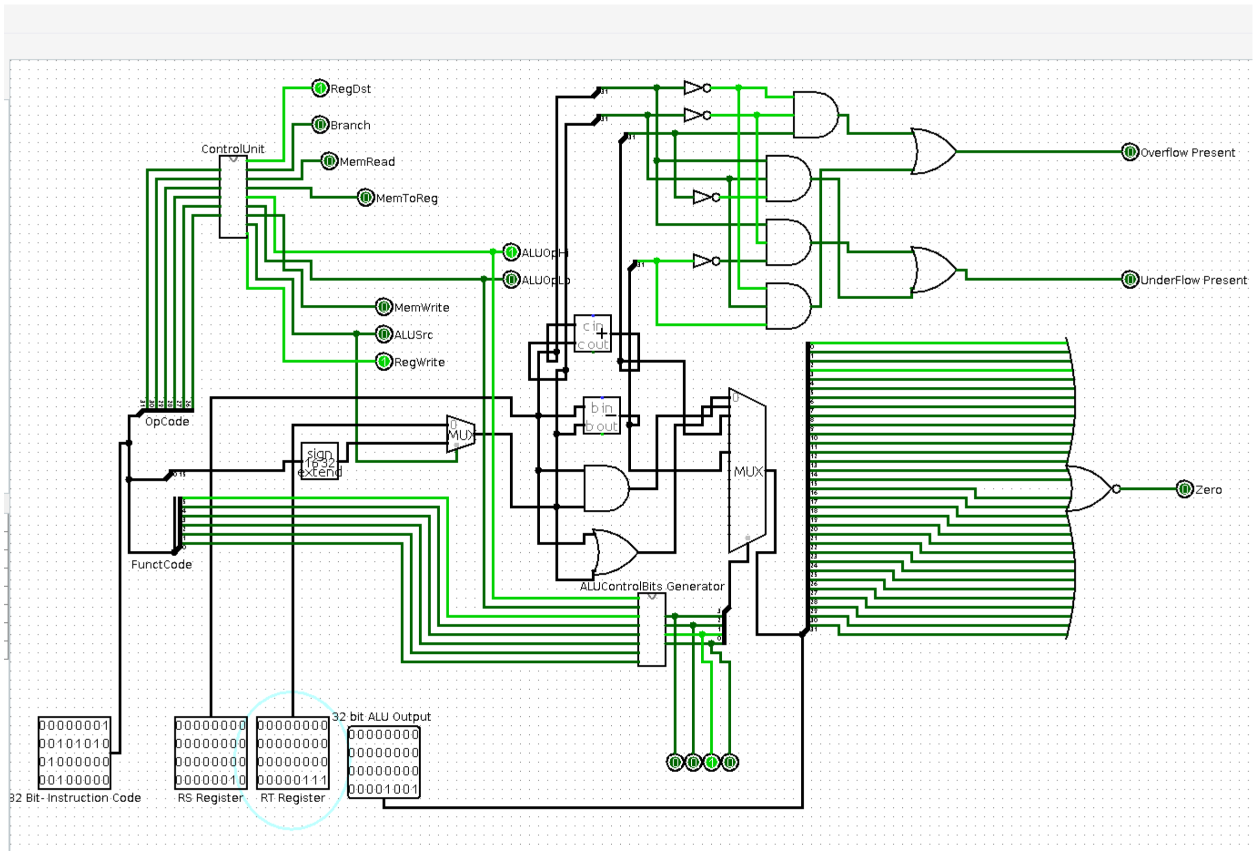


Opcode	Instruction	Function Code	RegDst	Branch	MemRead	MemToReg	ALUOp	MemWrite	ALUSrc	RegWrite	ALU Control Bits
000000	ADD	100000	1	0	0	0	10	0	0	1	0010
000000	SUB	100010	1	0	0	0	10	0	0	1	0110
000000	AND	100100	1	0	0	0	10	0	0	1	0000
000100	BEQ	N/A	1	1	0	0	01	0	0	0	0110
100011	LW	N/A	0	0	1	1	00	0	1	1	0010
101011	SW	N/A	X	0	0	X	00	1	1	0	0010
001000	ADDI	N/A	0	0	0	0	00	0	1	1	0010
000010	JUMP	N/A	X	0	0	X	XX	0	X	0	XXX

TESTING:

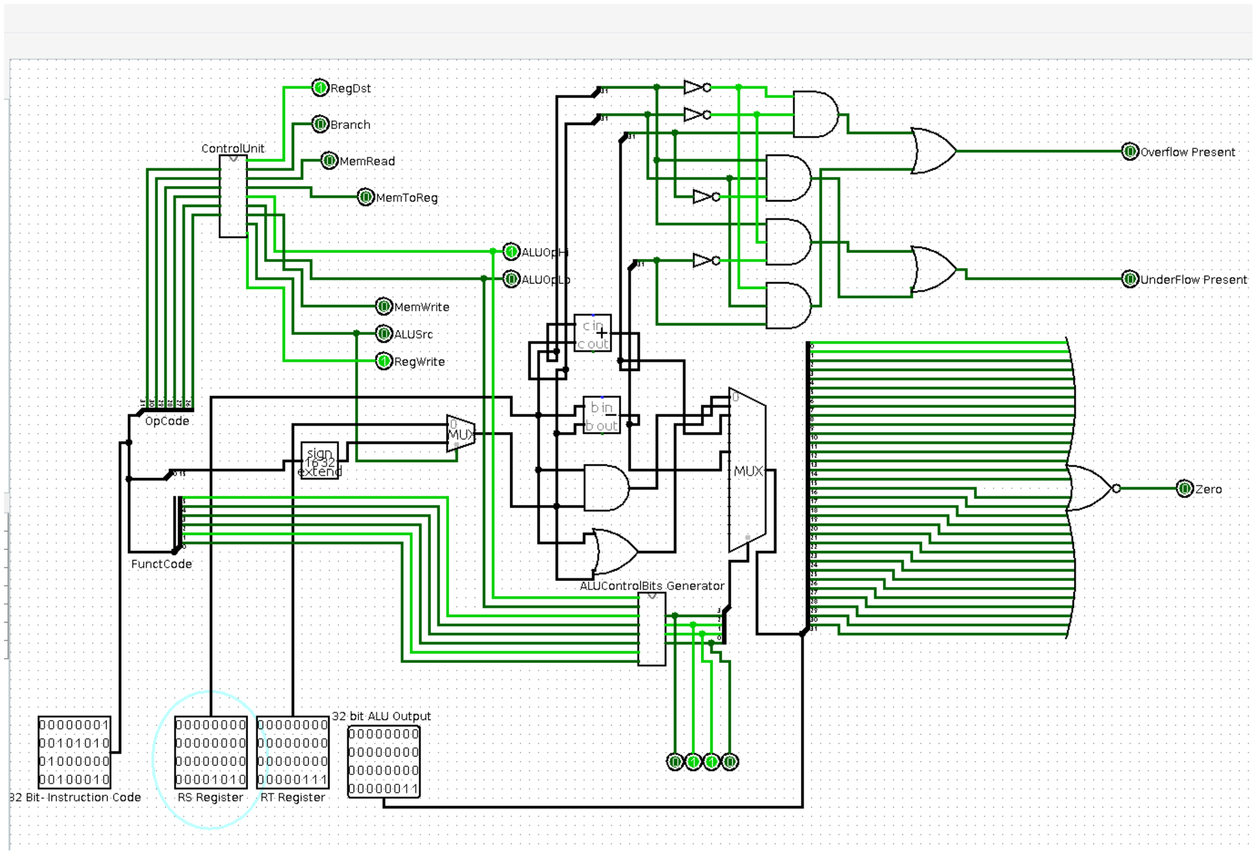
ADD:

- R-TYPE
- add $t0, $t1, $t2
- 32 bit instruction: 000000 01001 01010 01000 00000 100000



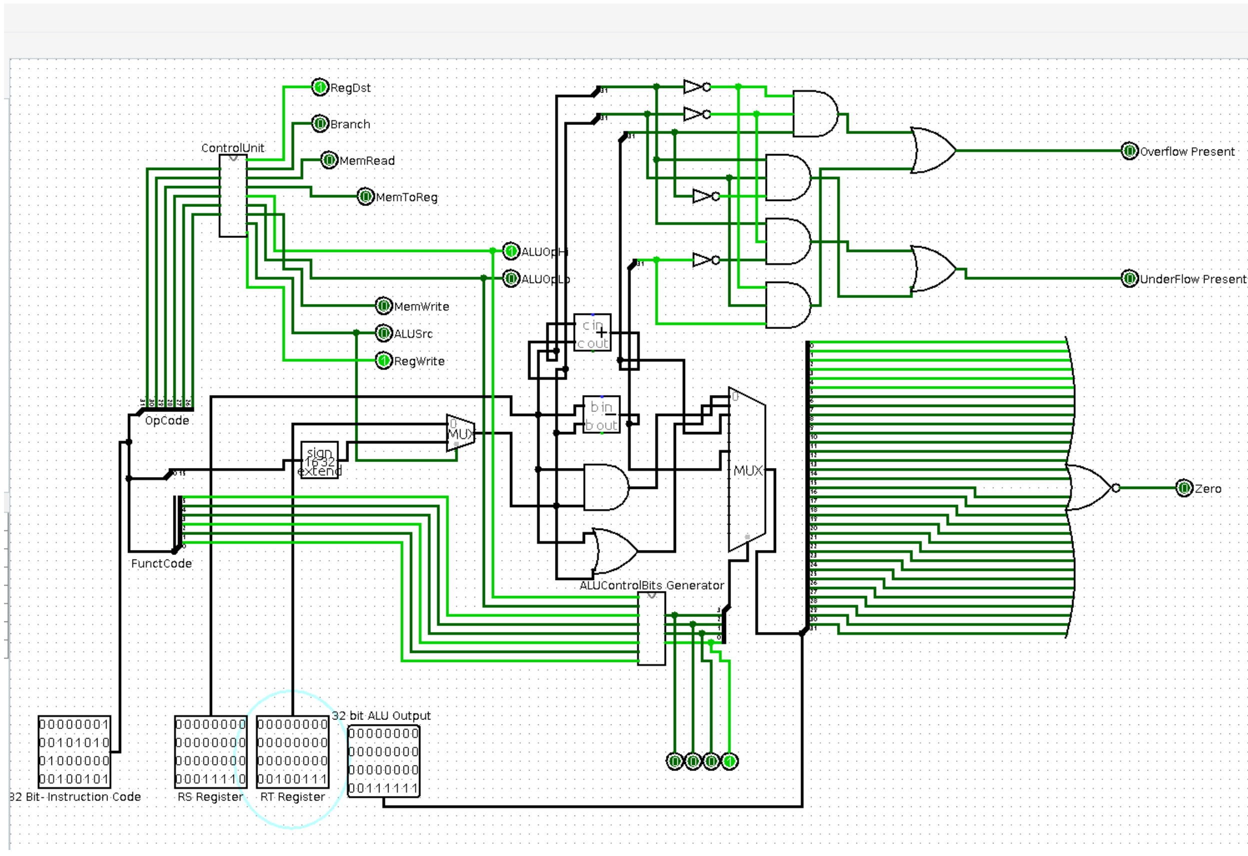
SUB:

- R-TYPE
- sub $t0, $t1, $t2
- 32 bit instruction: 000000 01001 01010 01000 00000 100010



OR:

- R-Type
- or $t0, $t1, $t2
- 32 bit instruction: 000000 01001 01010 01000 00000 100101



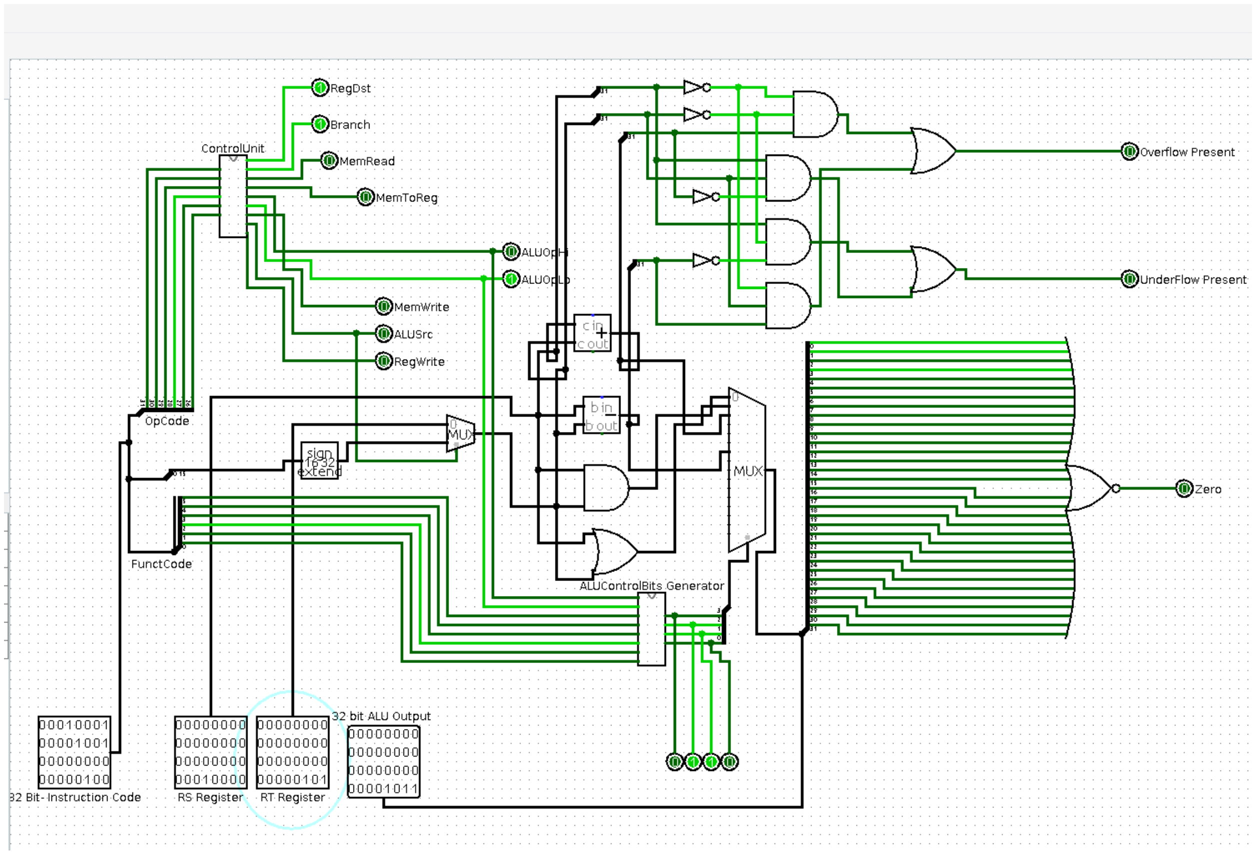
```
32 Bit- Instruction Code   RS Register   RT Register   32 bit ALU Output
00000001                   00000000      00000000      00000000
00101010                   00000000      00000000      00000000
01000000                   00000000      00000000      00000000
00100101                   00011110      00100111      00111111
```

I type:

BEQ:

beq $t0, $t1, 6

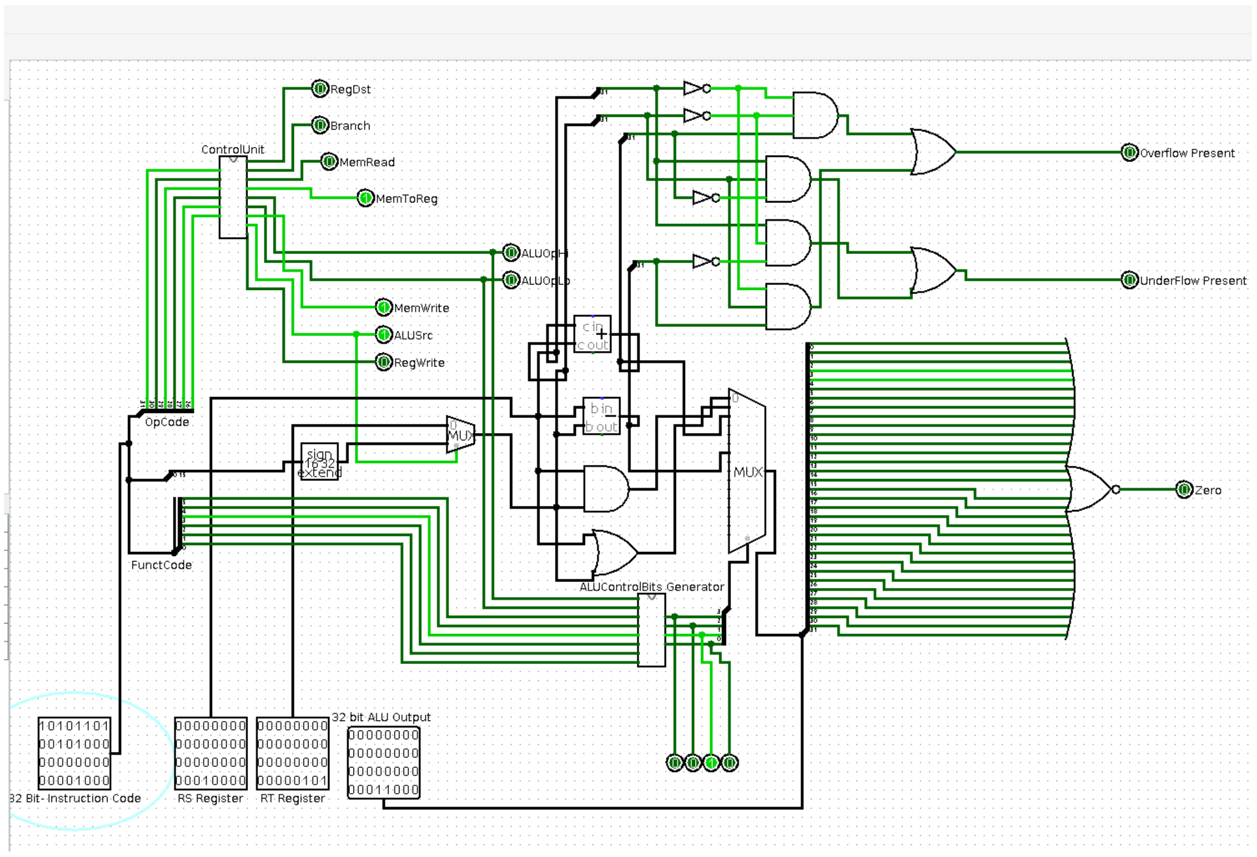
32 bit instruction: 000100 01000 01001 0000 0000 0000 0110



SW:

sw $t0, 8($t1)

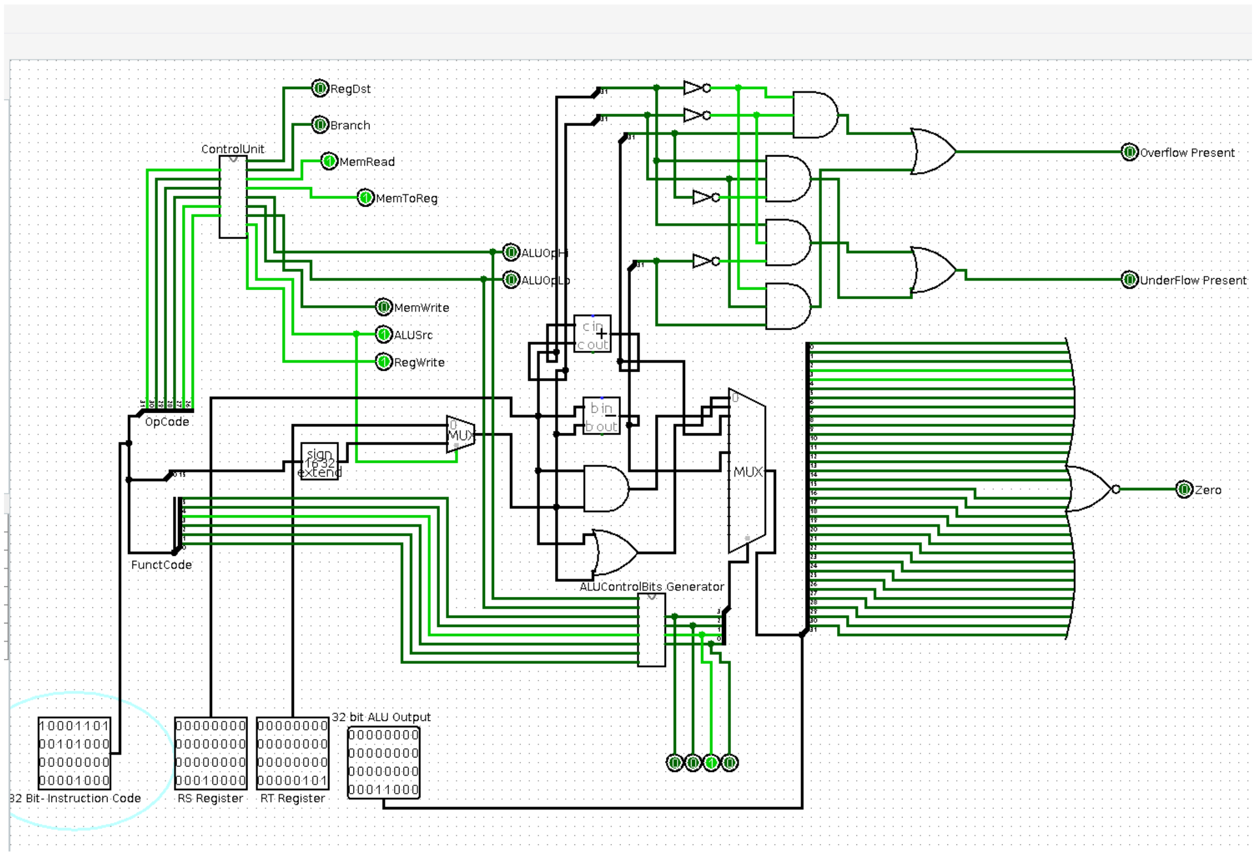
32 bit instruction: 101011 01001 01000 0000 0000 0000 1000



LW:

lw $t0, 8($t1)

32 bit instruction: 100011 01001 01000 0000 0000 0000 1000

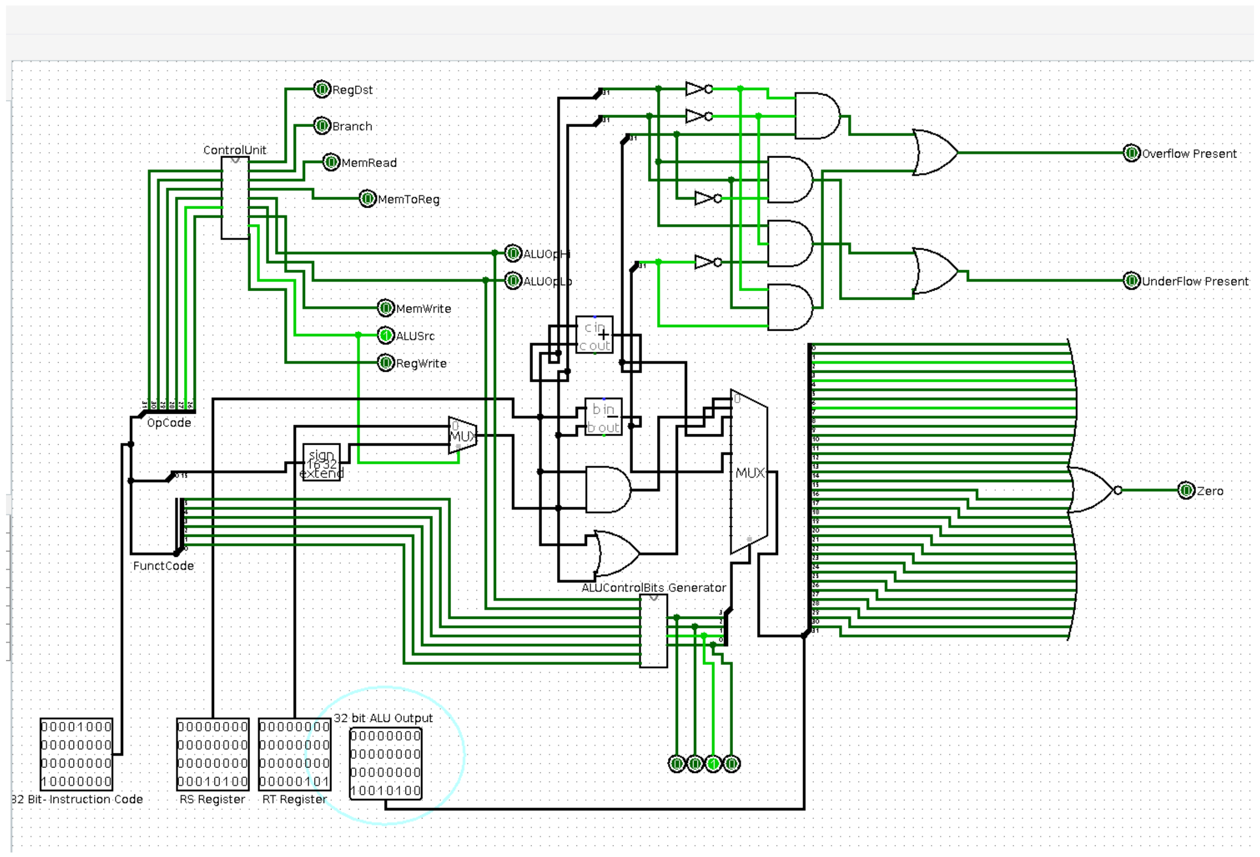


J-Type:

JUMP:

j 128

32 bit instruction: 000010 0000 0000 0000 0000 1000 0000



OVERFLOW AND UNDERFLOW DETECTION:

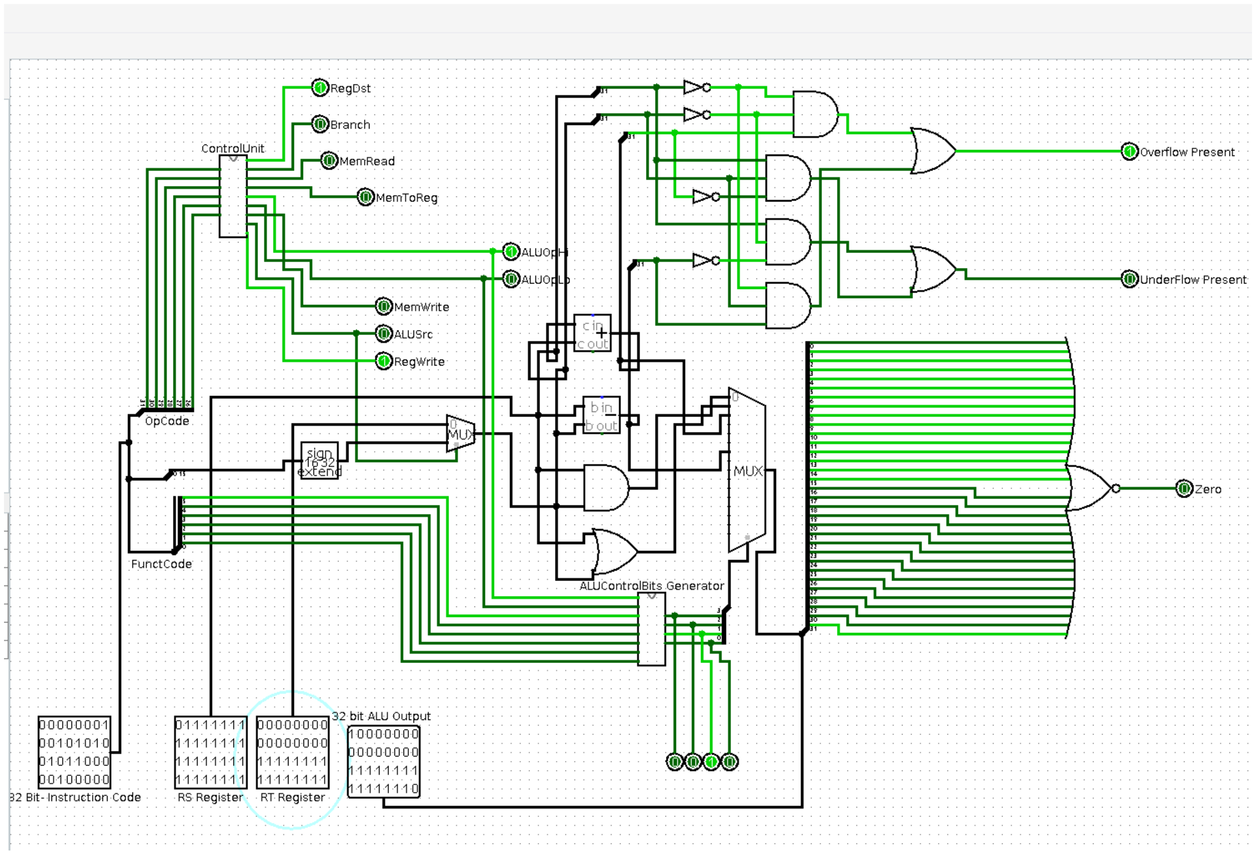
OVERFLOW:

We will add two positive large numbers so that the result overflows.

32 bit instruction: 00000001001010100101100000100000

rs- 01111111111111111111111111111111

rt- 00000000000000001111111111111111



Overflow detected as the ‘overflow present’ is highlighted.

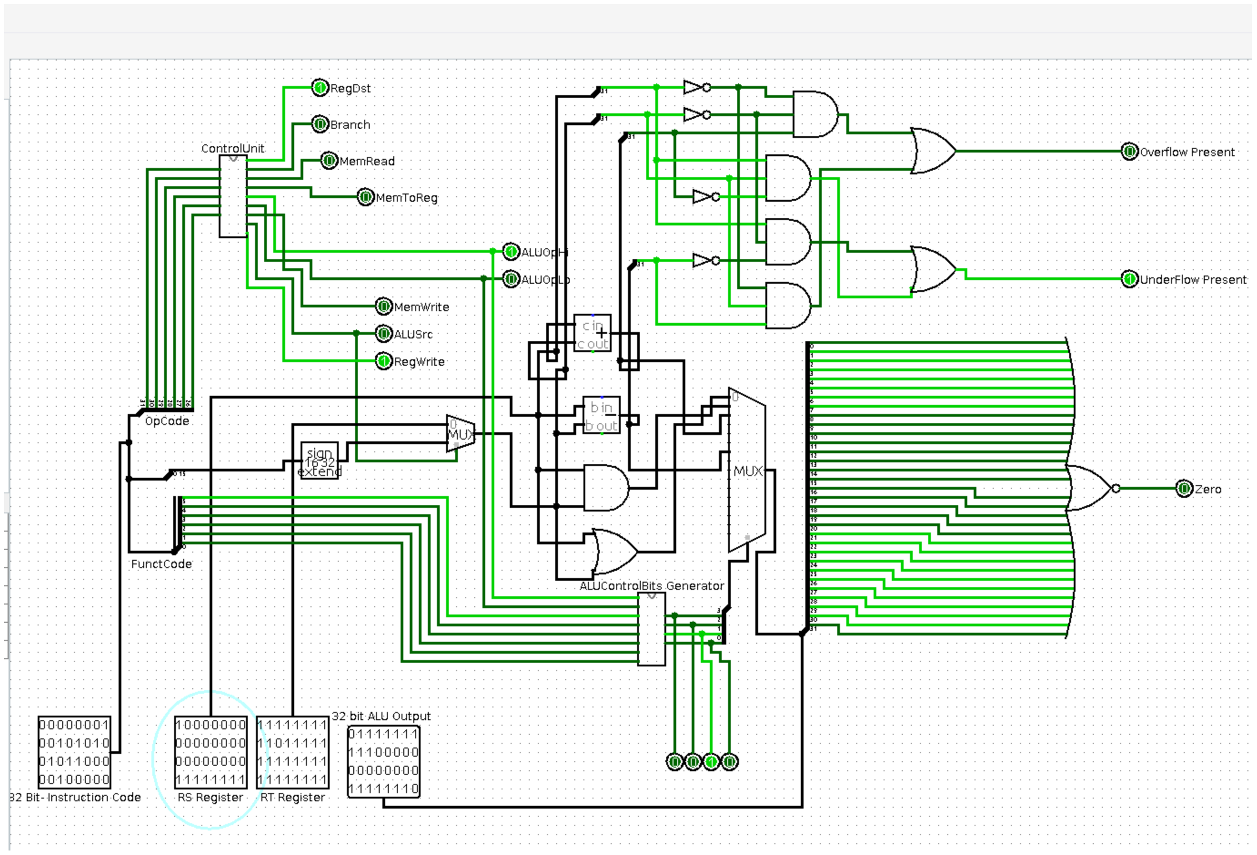
UNDERFLOW:

WE will add two large negative numbers so that the result underflows and gives a positive result.

32 bit instruction: 00000001001010100101100000100000

rs- 11111111111111111111111111111111

rt- 10000000000000000000000001111111



```
00000001   10000000   11111111   01111111
00101010   00000000   11011111   11100000
01011000   00000000   11111111   00000000
00100000   11111111   11111111   11111110
```

UNDERFLOW DETECTED as ‘underflow present’ is highlighted.