CSE116 Computer Architecture

MIPS Datapath and Control Unit Simulator

Submitted by:

Ahmed Bahaa Ibrahim	16P6057
Andrew Sameh Labib	16P6007
Basma Magdy Mohamed	16P8187
Kirolous Raouf Attyea	16P8045
Mario Sherif Rasmy	16P3021

Submitted to:

Dr. Cherif Ramzi Salama

Eng. Ahmed Fathy

Table of Contents

Project Description

This is a program that represents a MIPS simulation supported by "*Educational Graphical User Interface (GUI)"* (low level instructions).

Firstly, the program takes the memory addresses of the values from the user as well as the value stored in each address. Then, it takes PC value and the instructions followed by the registers. Finally, each instruction written by the user to be executed is represented on a GUI Datapath - showing the value carried by each wire- by clicking the "Next" button.

Datapath Components :

For all instructions -not including “jr”, the components of GUI datapath are:

-PC:

Initial memory address is saved in.

-Instruction Memory:

Takes PC value, fetch instruction at that address and outputs a wire to “shift left 2” carrying 26 bit jump offset -in case of J type instructions- , a wire to “control unit” carrying 6 bits opcode of the function, a wire to “register file” carrying 5 bits rs value , a wire to “register file” and branched to “regdst mux” carrying 5 bits rd value, a wire to “regdst mux” carrying 5 bits rt value , a wire to “sign extend” carrying 16 bits immediate value -in case of I type instructions , a wire to “ALU control” carrying 6 bits function code -in case of R type instructions.

- PC+4 Adder:

Adds 4 to the PC value and outputs a wire to “Target Address Adder” branching to ”Beq MUX” carrying PC+4.

-Shift- 2:

Outputs the multiplication of jump offset by 4 that meets junction point which concatenates the most significant 4 bits carried by the output of the “PC+4-Adder” with the output.

-Control Unit:

Takes the input from “Instruction Unit” and outputs 8 wires -carrying 1-bit-1 or 0 , a wire to an “AND gate” , a wire to “Jump MUX”, a wire to” Data Memory” (MemRead) , a wire to “MemtoReg MUX”-in case of load instruction-, a wire to “Data Memory” (MemWrite) , a wire to “RegtoALU MUX”, a wire to “Register File” , a wire to “RegDst MUX”. In addition to a wire carrying 2-bits “ALUOp” to “ALU Control”.

-Register File:

Takes rs, rt , the output of “RegDst” as a write register and “RegWrite” -output of “Control Unit”, and outputs a wire to “ALU” carrying value in rs, a wire to “RegtoAlu” branching to “Data Memory” carrying value in rt.

-Sign Extender:

Takes 16 bits offset extends it to 32 bits in case of I type instructions. It outputs a wire to “Shift-2” carrying the extended offset , a wire to “RegtoALU MUX” carrying the extended offset.

-Shift-2.1:

Takes “Sign Extender “ output and multiplies it by 4 and it is carried to “Target Address Adder” .

-ALU Control

Takes 6 bits function code from “Instruction Memory” -in case of R type- and “ALUOp” from “Control Unit” , and outputs a wire to “ALU” carrying data of “ALU Control” in 4 bits (operation depending).

-ALU:

Takes value in rs (output of “Register File”) , output of “RegtoAlu MUX” and output of “ALU Control” , and outputs a wire to “AND gate “ carrying 1-bit 1 or 0 , a wire to “Data Memory” branching to “MemtoReg MUX” carrying “ALURes”.

-Target Address Adder:

Takes PC+4 (output of “4-Adder”) and output of “Shift-2.1” , and outputs a wire to “Beq MUX” carrying the addition of extended immediate multiplied by 4 32 bits and PC+4.

-Data Memory:

Takes “ALURes” from “ALU”, value in rt (output of “Register File”), “MemWrite” and “MemRead” from “Control Unit”, and outputs a wire to “MemtoReg MUX” carrying read data.

-RegDst MUX:

Takes rt , rd from instruction memory , a wire from the “Control Unit“ carrying 1-bit 1 or 0 and outputs rt or rd (depending on the instruction type).

-RegtoAlu MUX:

Takes the value in rt , extended immediate in 32 bits and a wire from the “Control Unit” carrying 1-bit 1 or 0 and outputs the value in rt or the extended immediate in 32-bits(depending on the instruction).

-MemtoReg MUX:

Takes the data read from the “Data Memory” as first input and “ALUres” and outputs data read from “Data Memory” -in case of load instruction- .

-Beq MUX:

Takes “pc address + 4” and the output of “ Target Address Adder”, the output of “AND GATE” and outputs “Target Address Adder” in case of “beq instruction”.

-Jump MUX:

Takes “Output of the BeqMUX” and “Output of shift-2” and outputs “Output of shift-2” incase of “jump instruction”.

-NOR Gate:

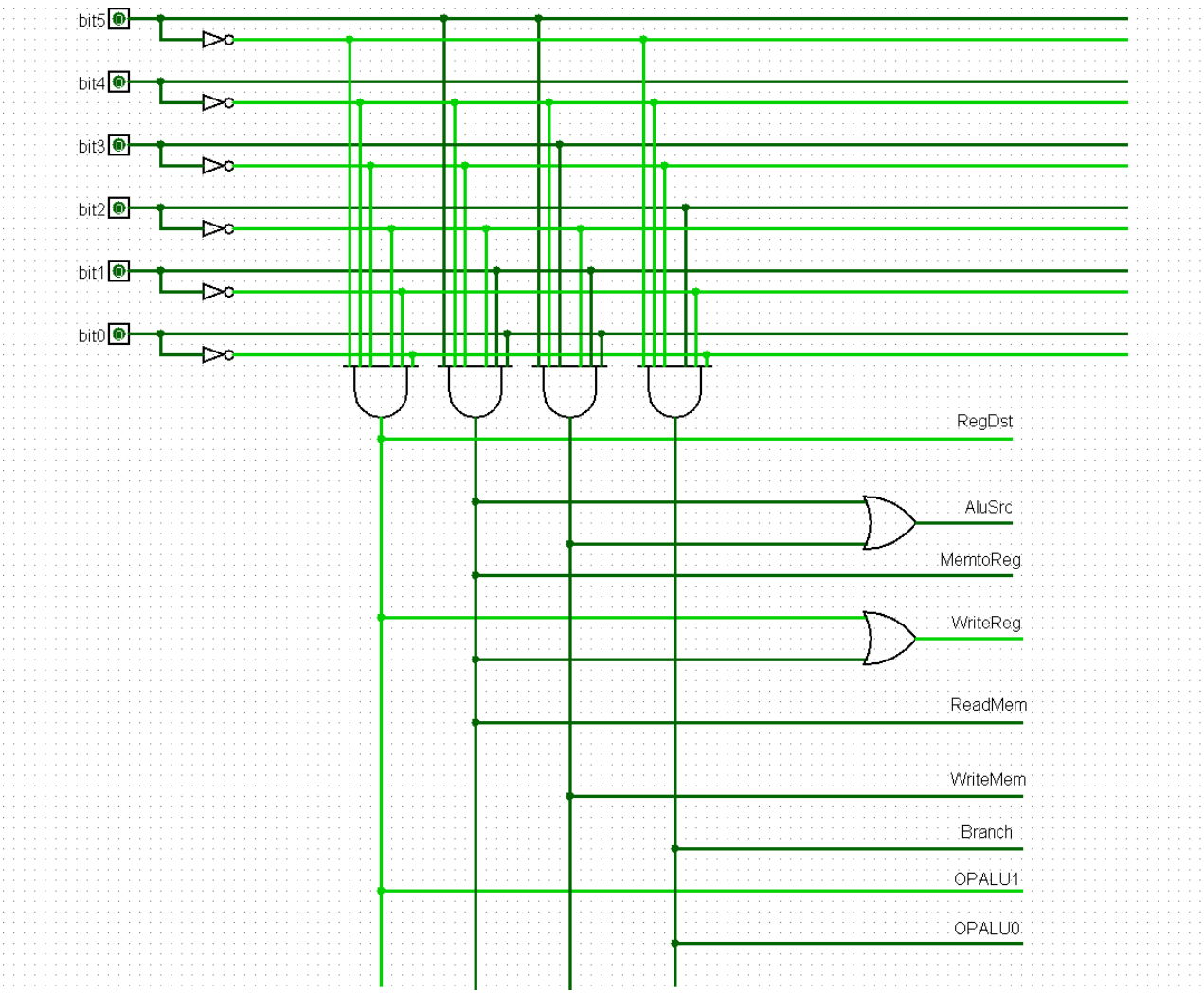
Takes the a wire from the control unit and carrying 1-bit (1 or 0) and “Zero flag output” from ALU, Outputs 1 in case of “Beq Instruction”.

Truth Table and Logic Diagram for the Control Unit

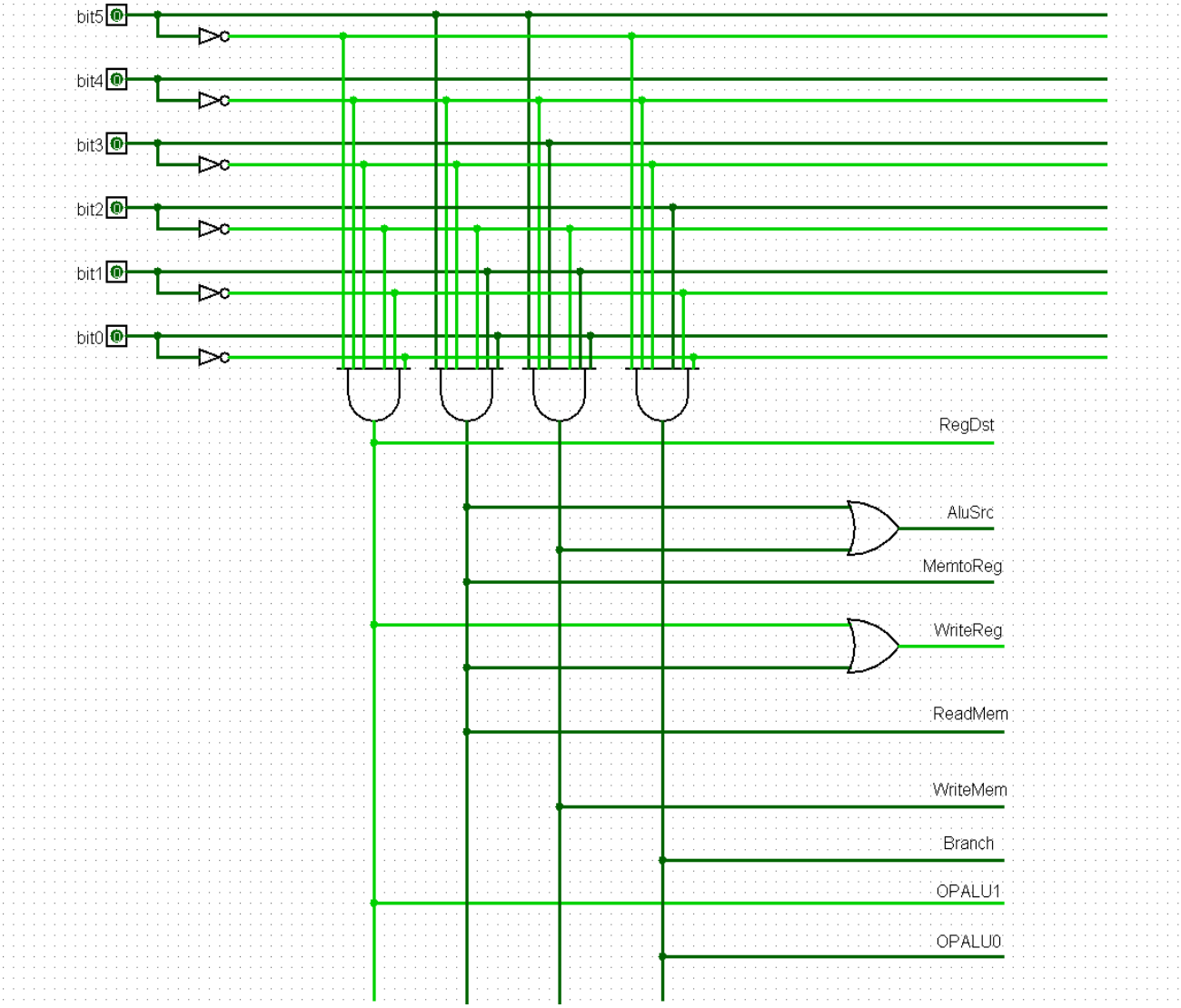
Instruction	RegDst	Jump	Branch	MemRead	MemtoReg	AluOp	MemWrite	AluSRC	RegWRite
add	1	0	0	0	0	10	0	0	1
Lw	1	0	0	1	1	00	0	1	1
Sw	x	0	0	0	x	00	1	1	0
Lb	0	0	0	1	0	00	0	1	1
Sb	x	0	0	0	x	00	1	1	0
addi	0	0	0	0	0	01	0	1	1
beq	x	0	1	0	x	01	0	0	0
Slt	1	0	0	0	0	10	0	0	1
Slti	0	0	0	0	0	11	0	1	1
Lbu	0	0	0	1	0	00	0	1	1
Sll	1	0	0	0	0	10	0	0	1
Nor	1	0	0	0	0	10	0	0	1
J	1	1	x	x	0	00	x	x	x
Jal	1	1	x	x	0	00	x	x	x
Jr	1	0	0	0	0	10	0	0	1

Logic Diagram

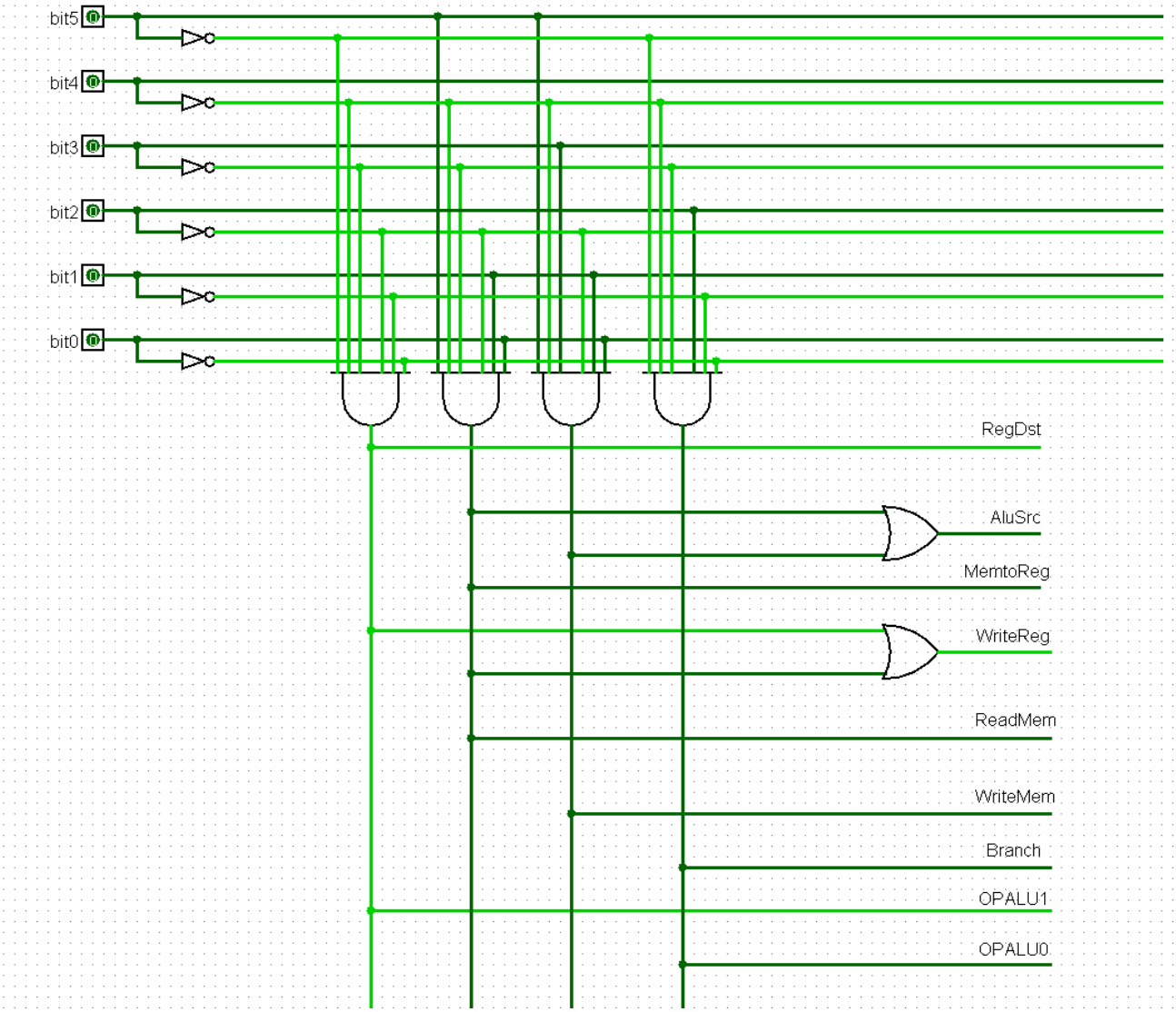
1)ADD Instruction (R-Type):



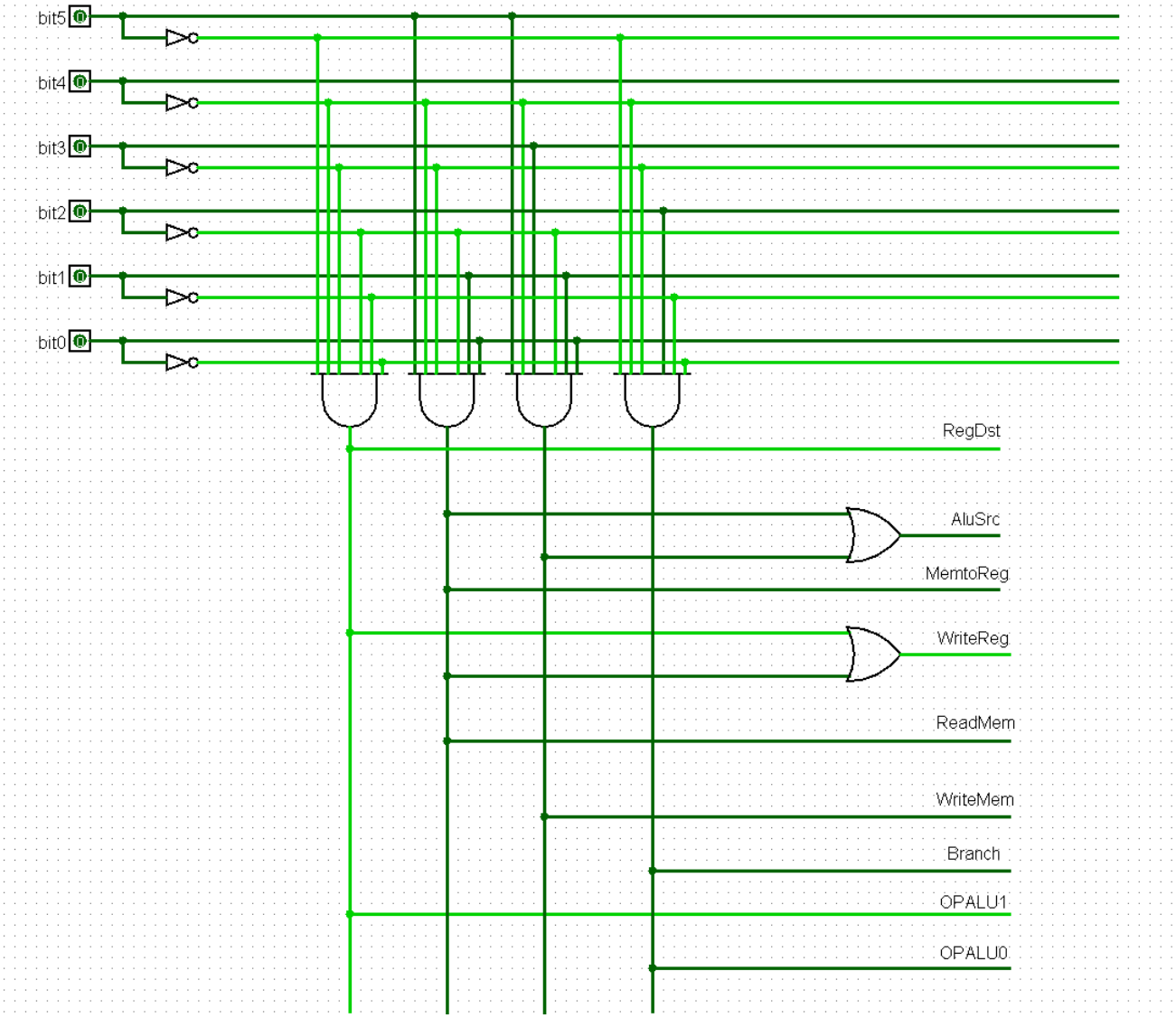
2) ADD Immediate Instruction (I-Type)



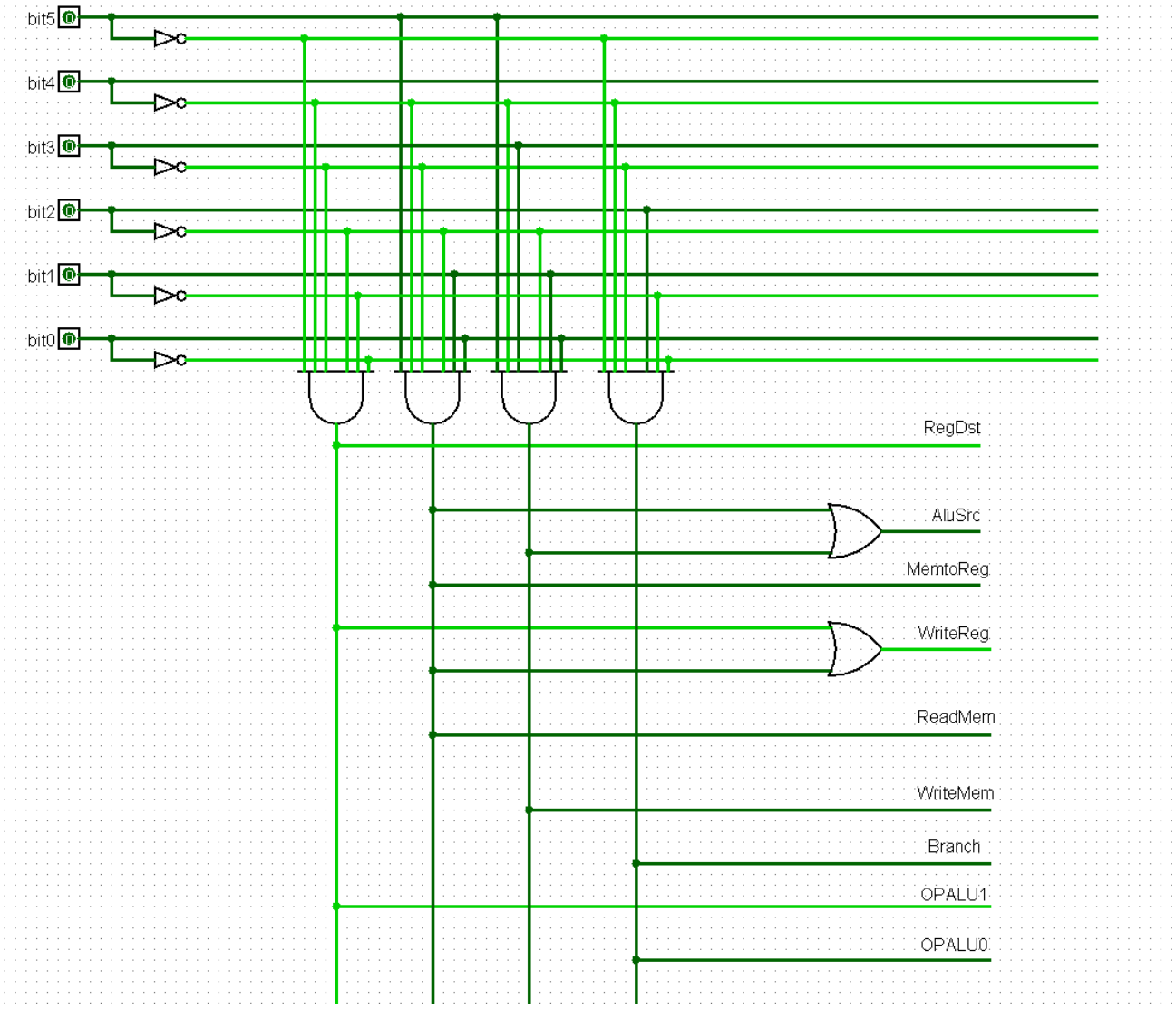
3-Load Word Instruction



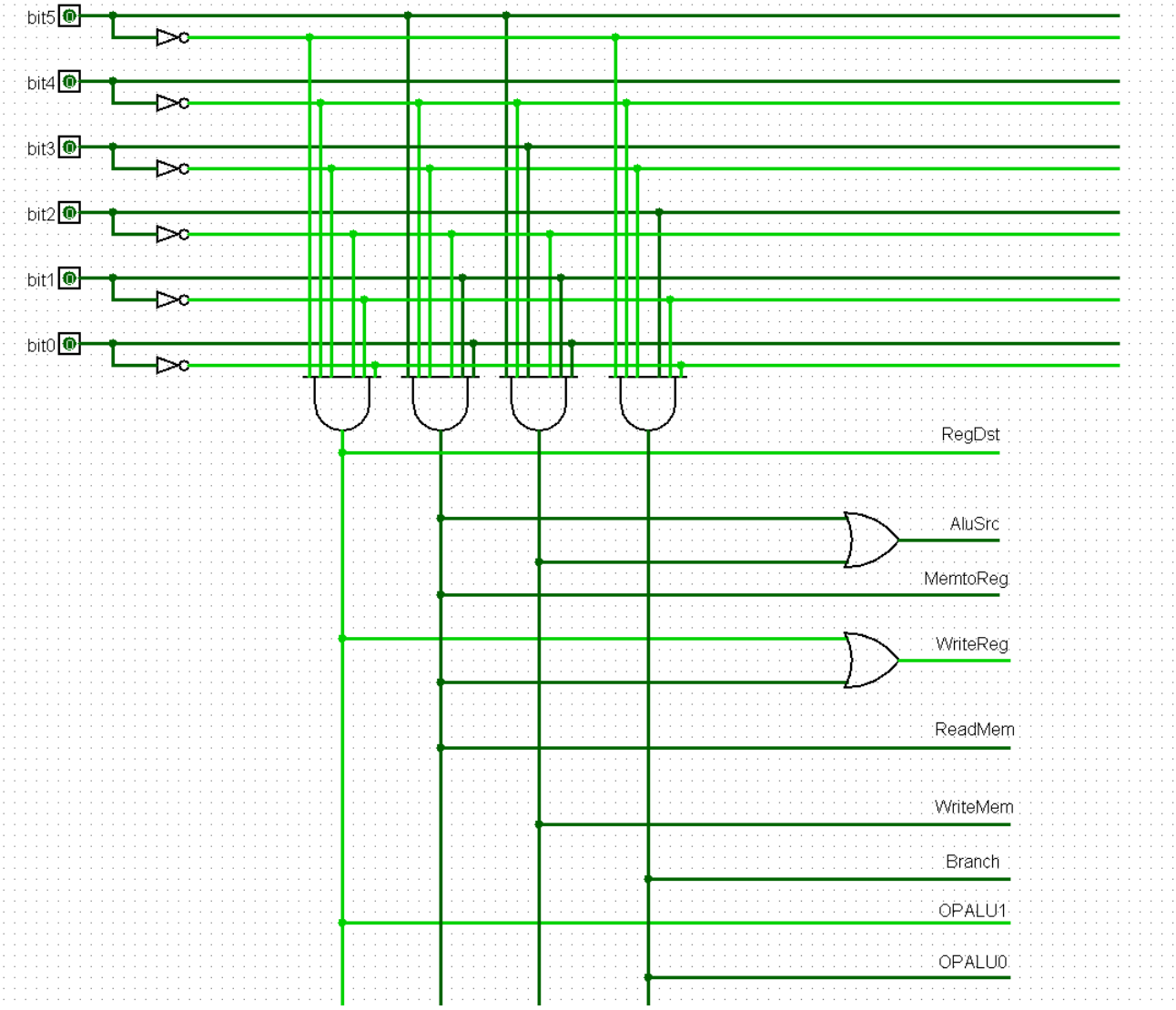
4)Store Word Instruction



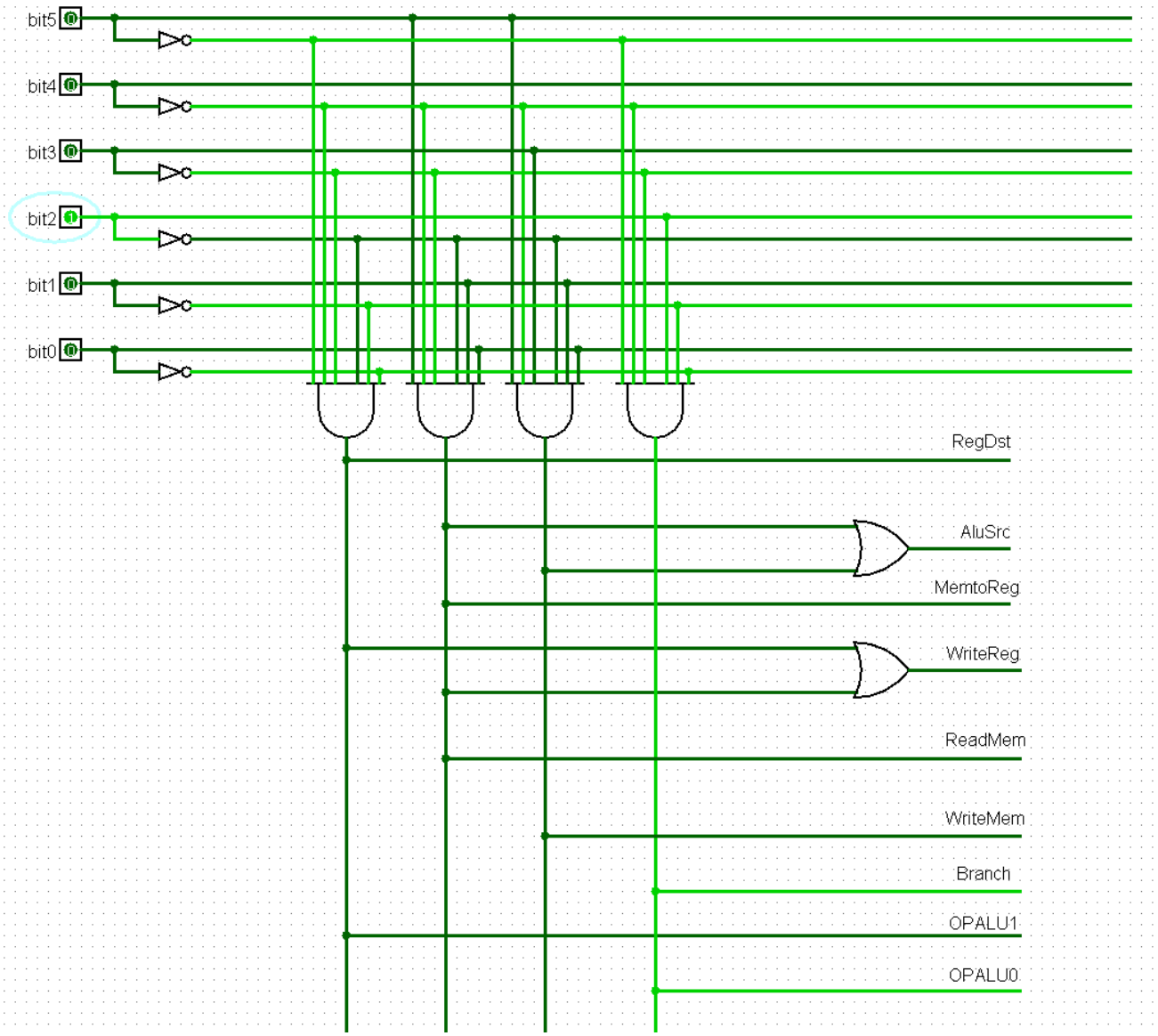
5)Load Byte Instruction (I-Type)



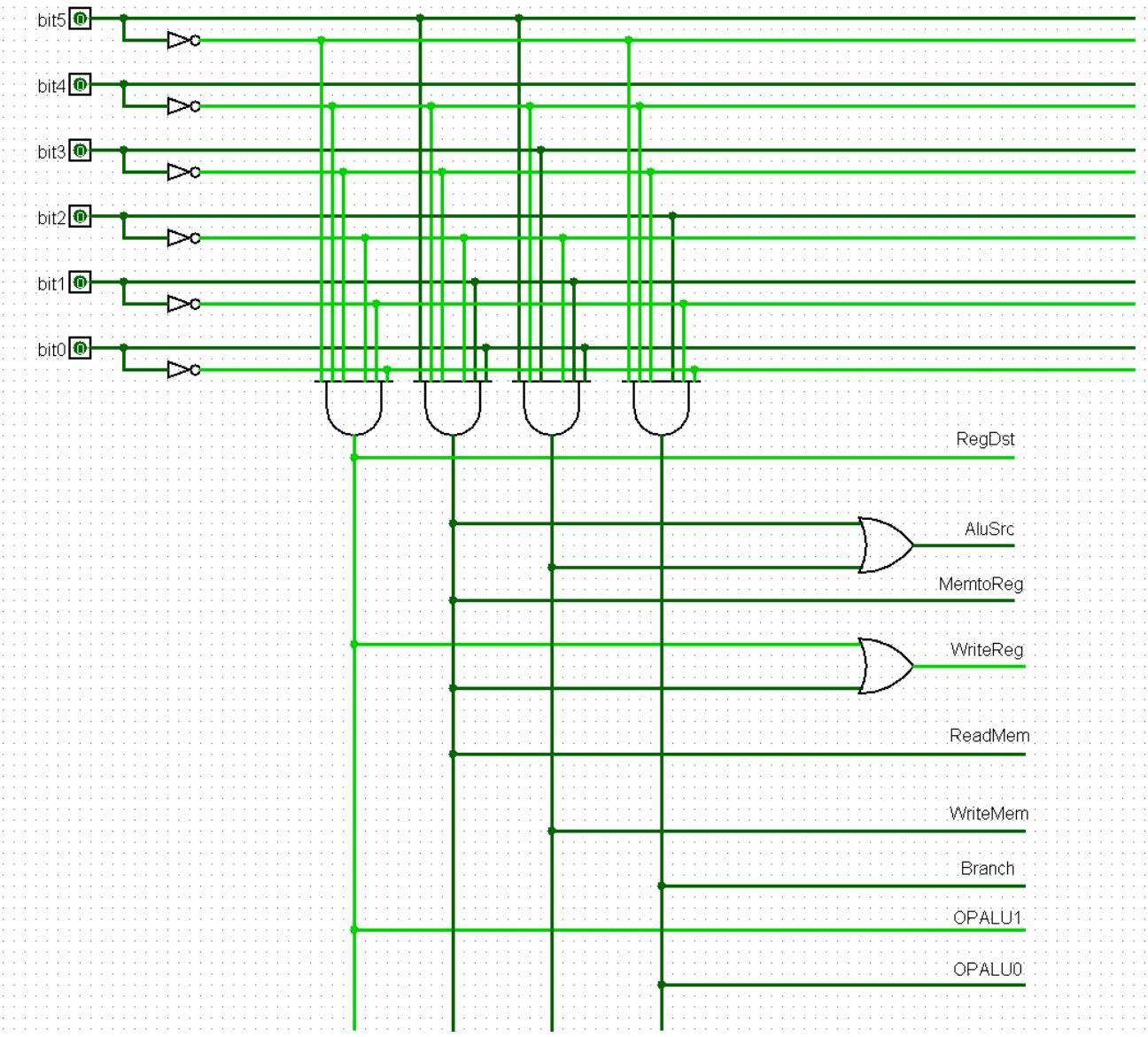
6) Store Byte Instruction (I-type)



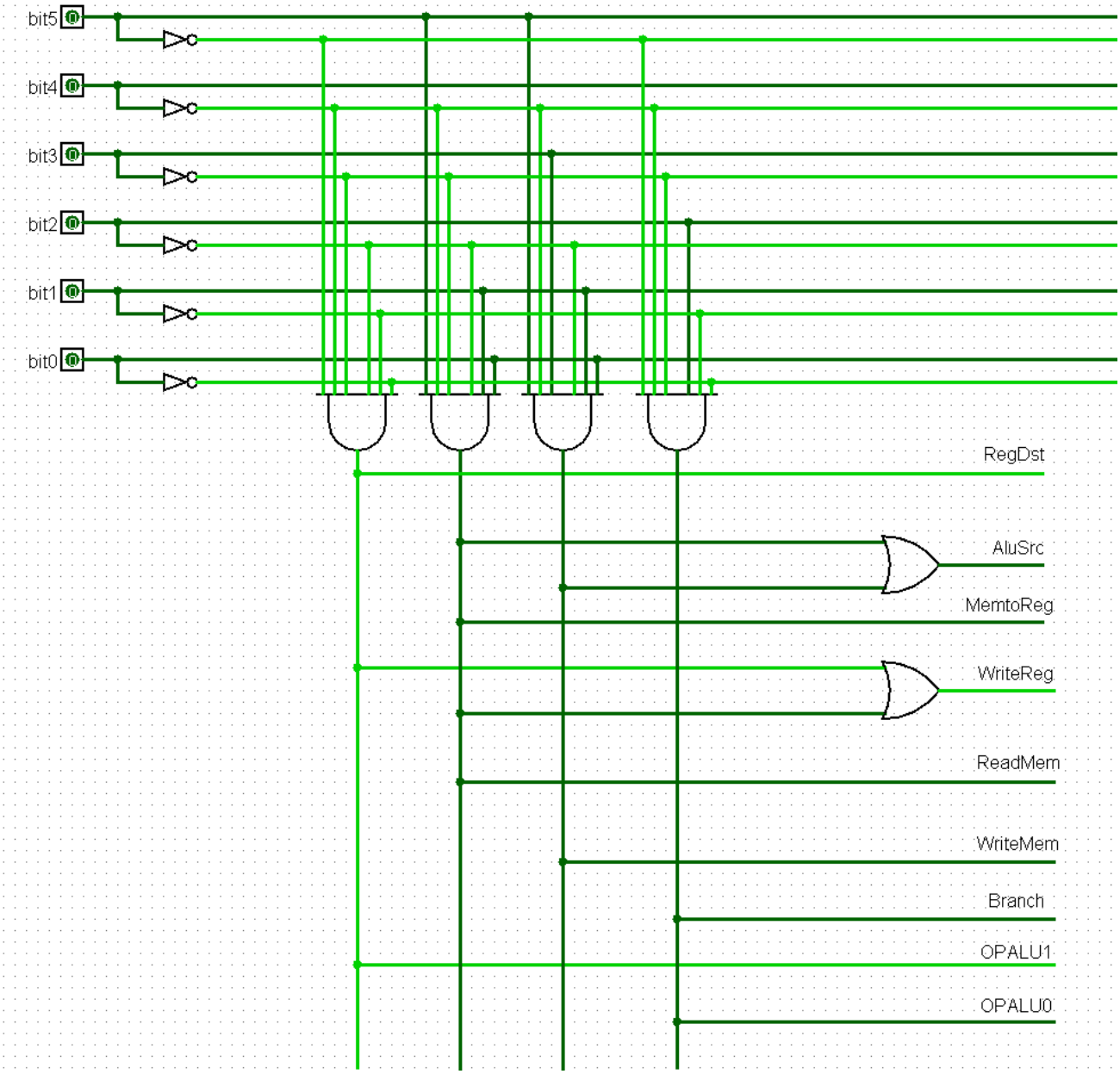
7) Branch Instruction (I-Type)



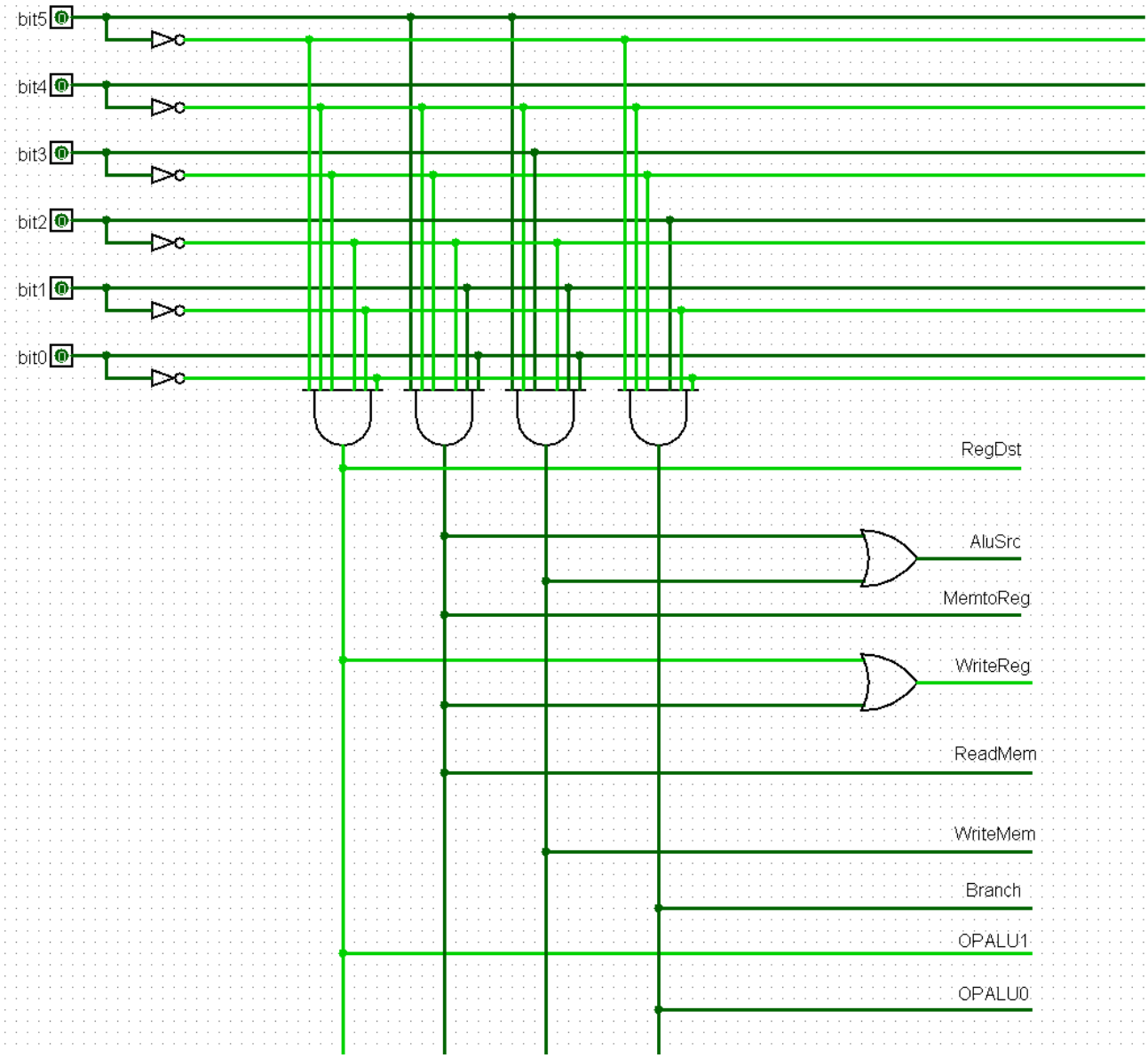
8) Set If Less Than Instruction (R-Type)



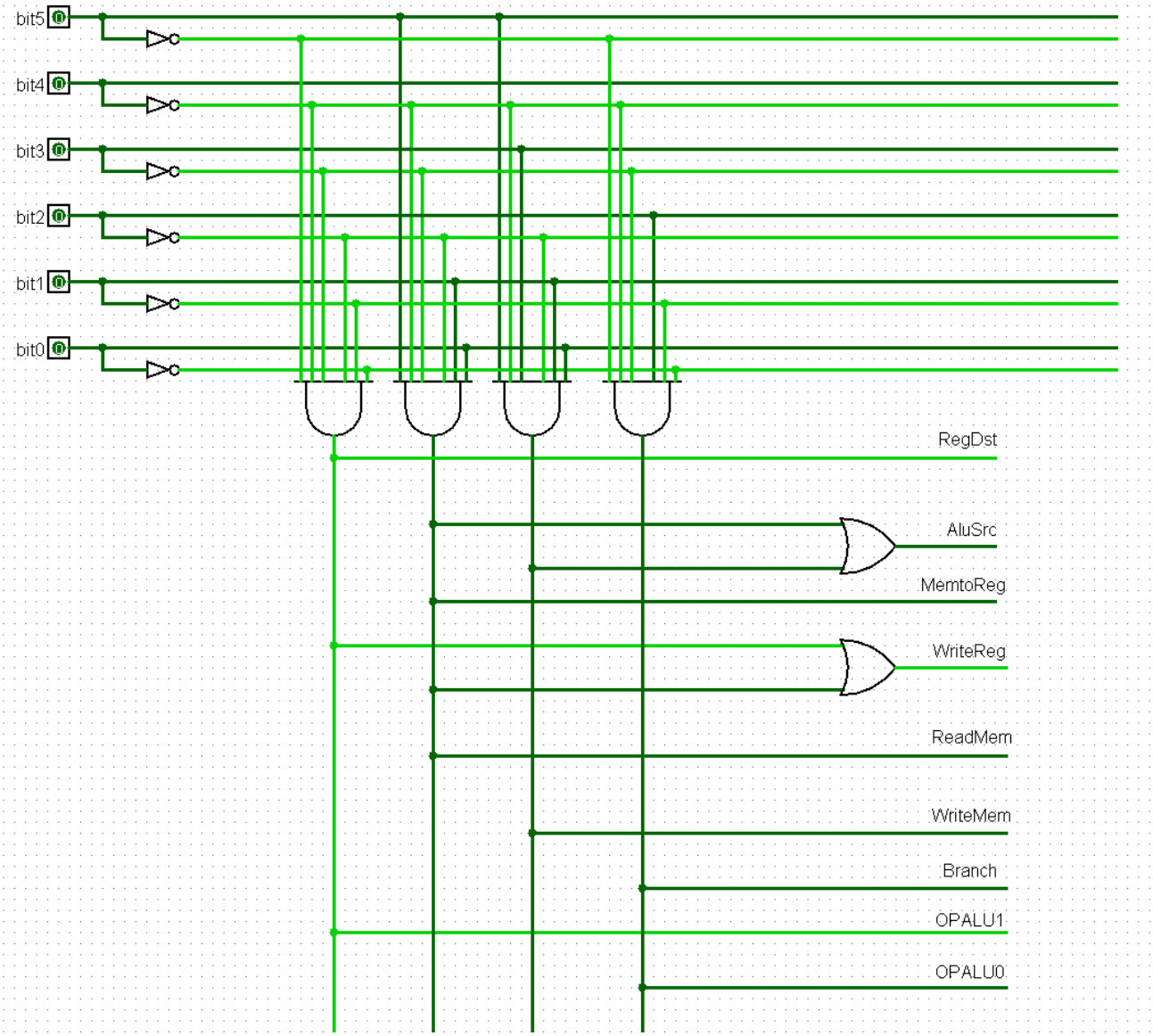
9) Set If Less Than Immediate (I-Type)



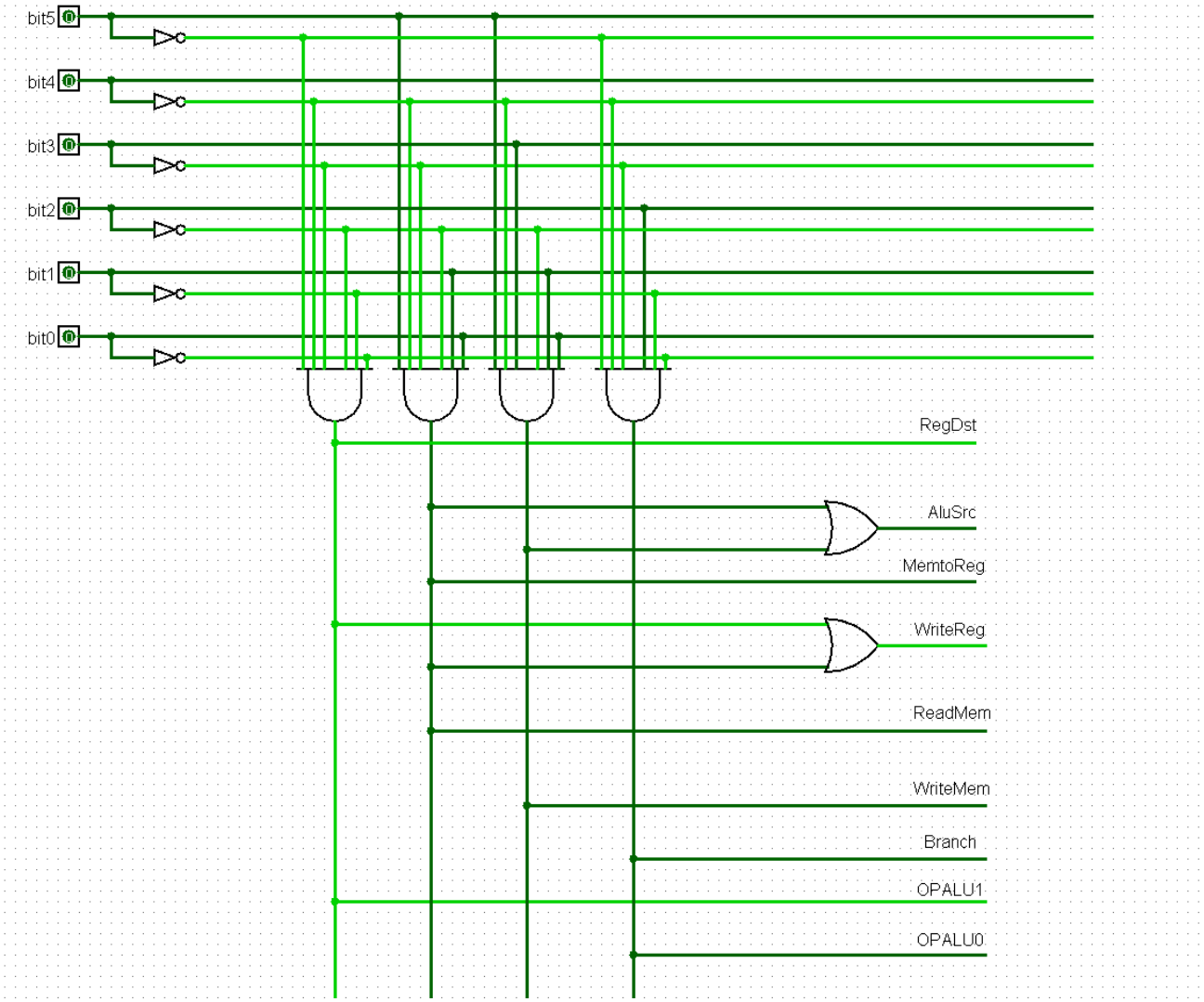
10) Load Byte Unsigned Instruction (I-Type)



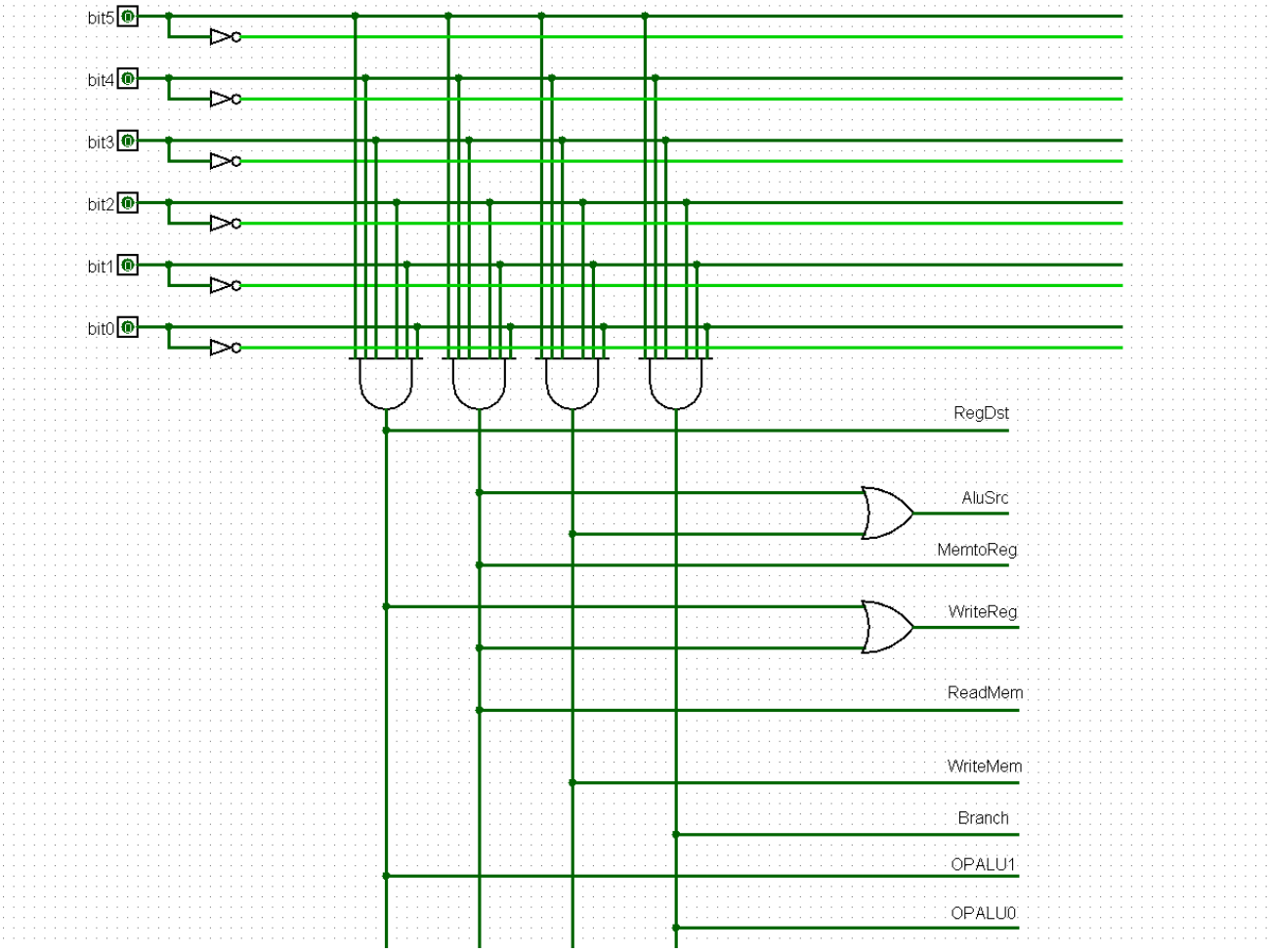
11) Shift Left Logical Instruction (R-Type)



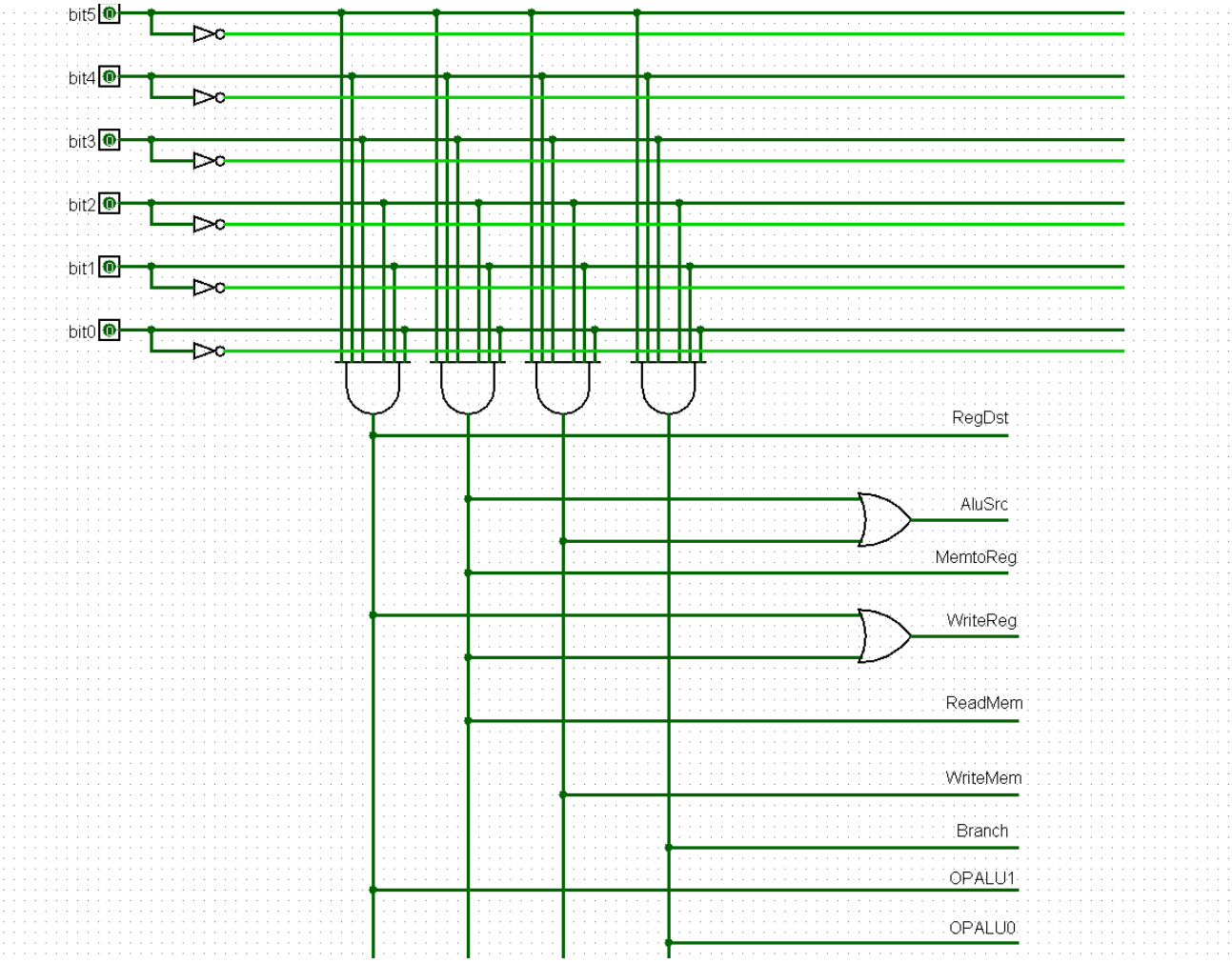
12) NOR Instruction (R-Type)



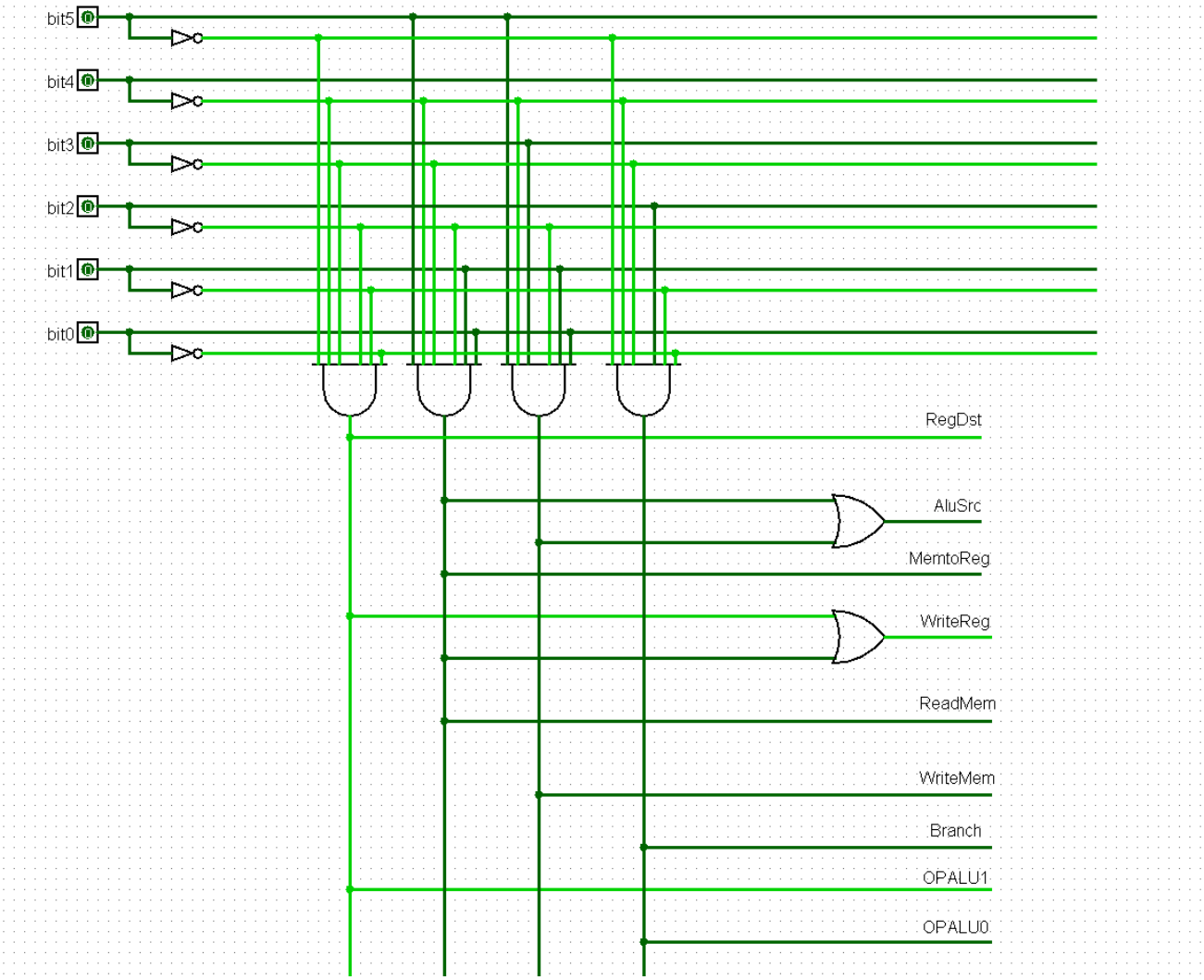
13) Jump Instruction (J-Type Instruction)



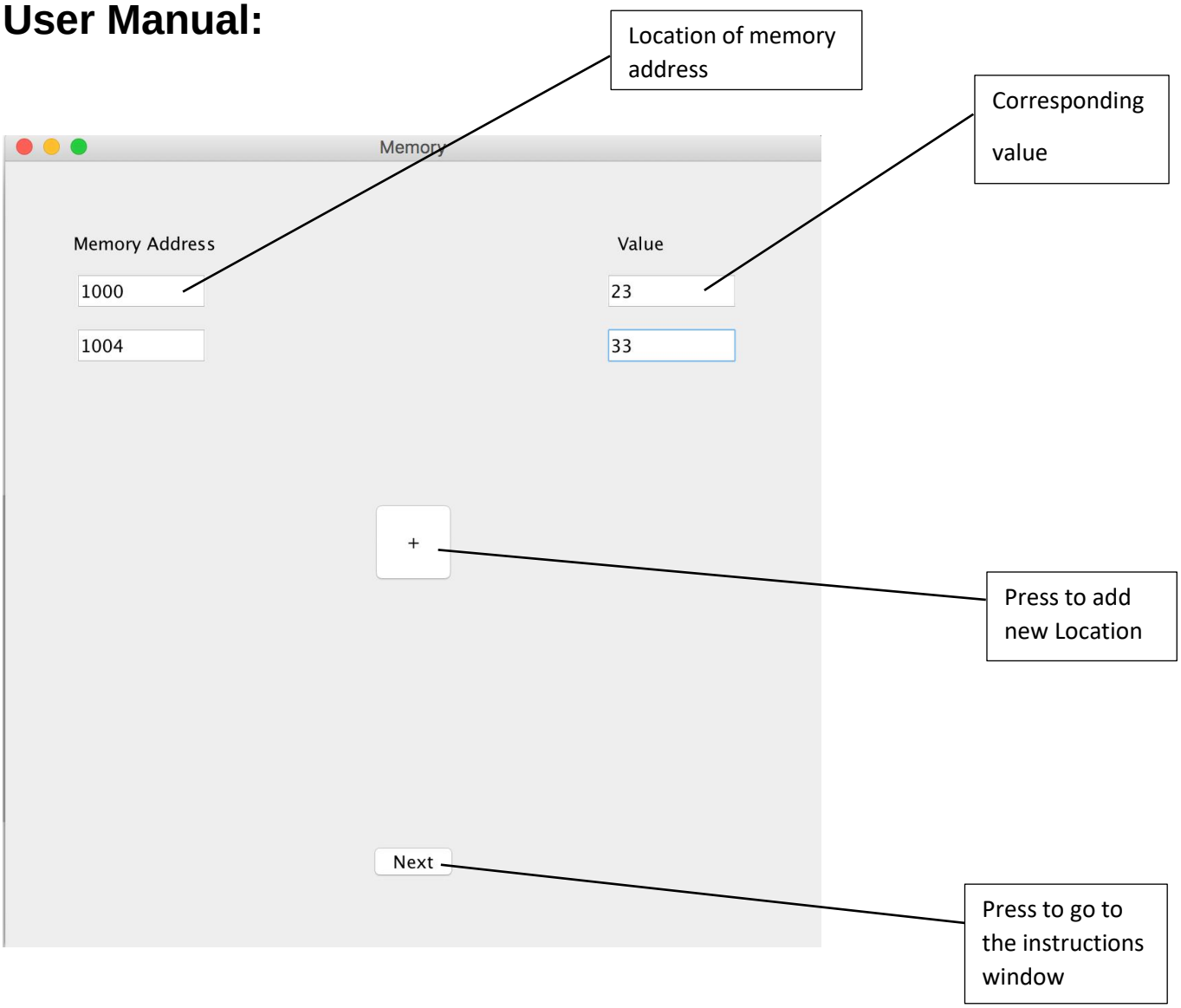
14) Jump and Link Instruction (J-Type Instruction)

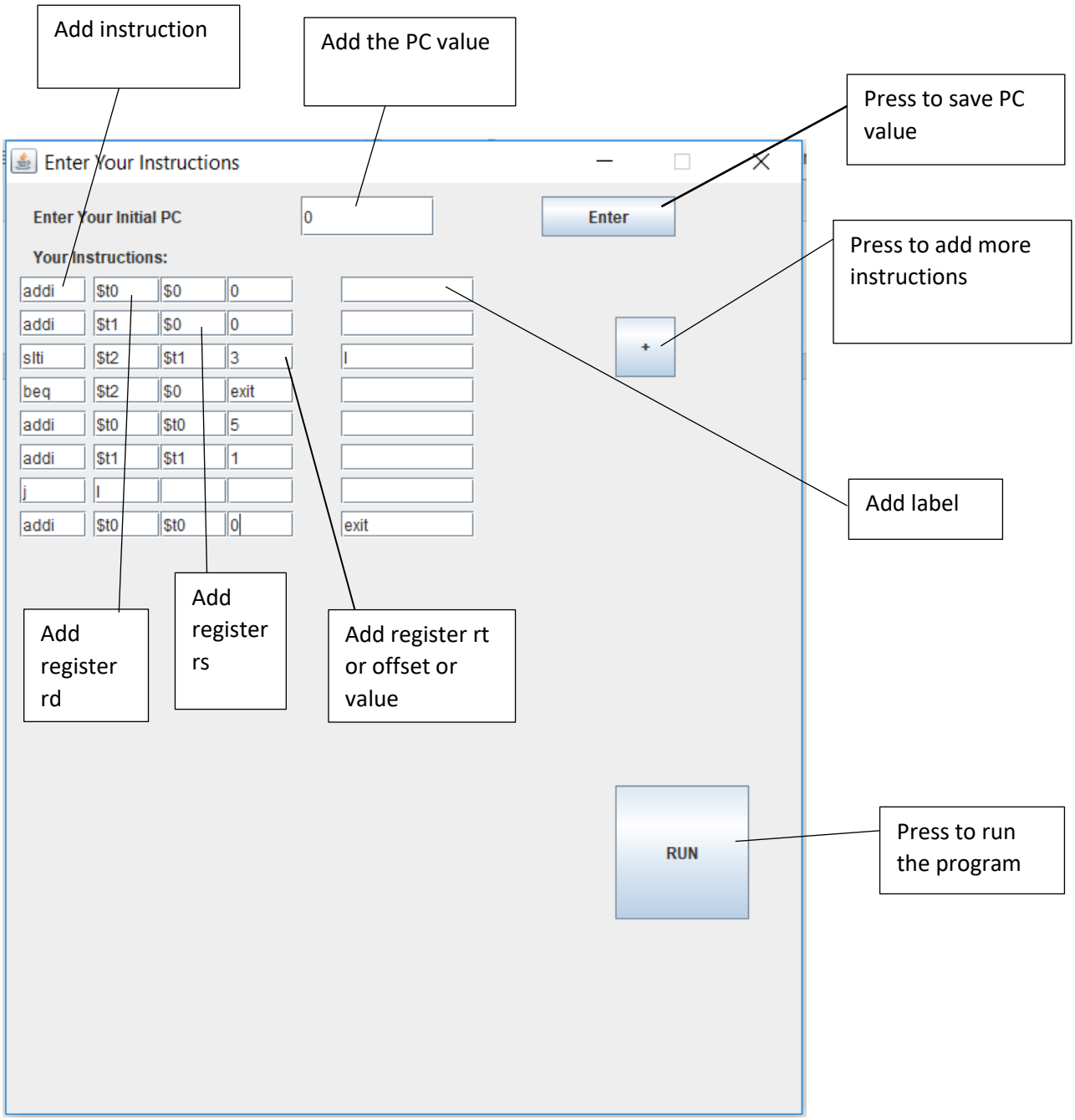


15) Jump Register Instruction (R-type Instruction)



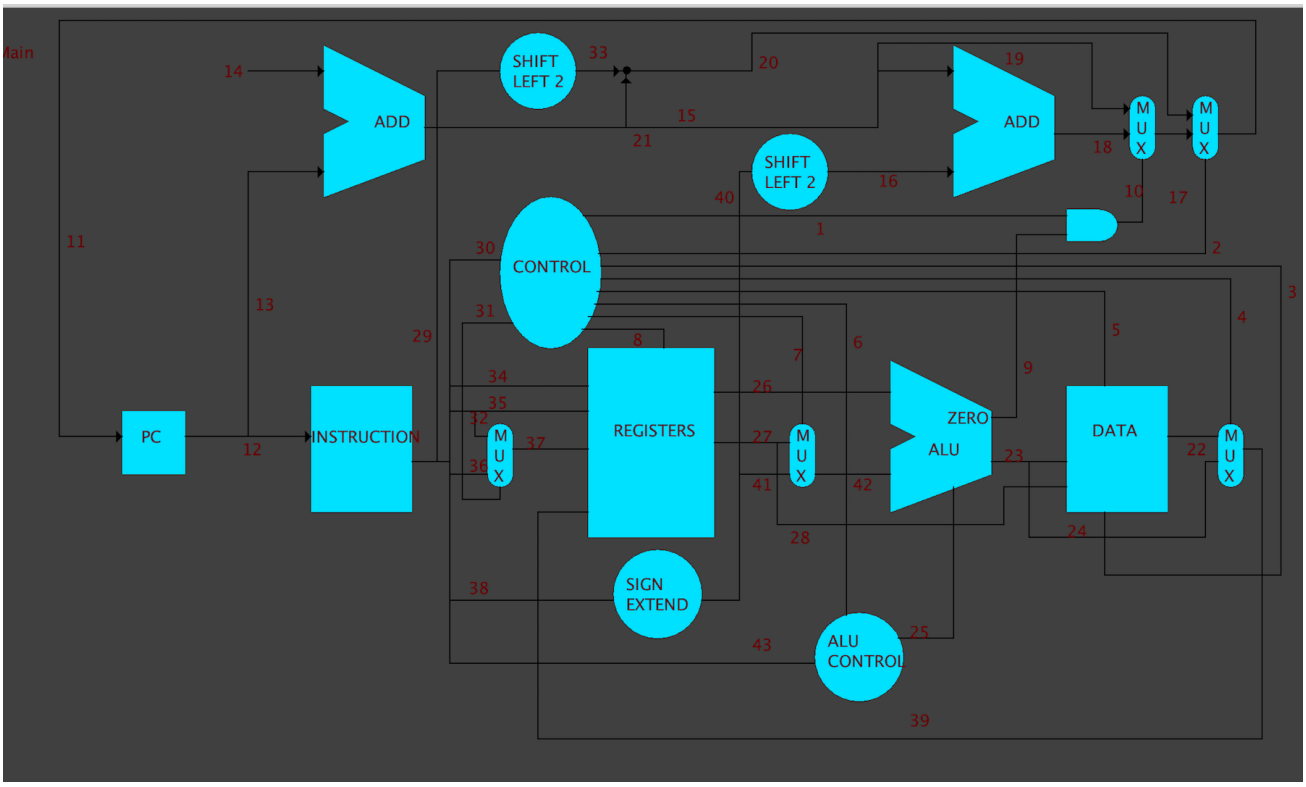
User Manual:


Location of memory address
Corresponding value
Memory
Memory Address
Value
1000
23
1004
33
+
Press to add new Location
Next
Press to go to the instructions window

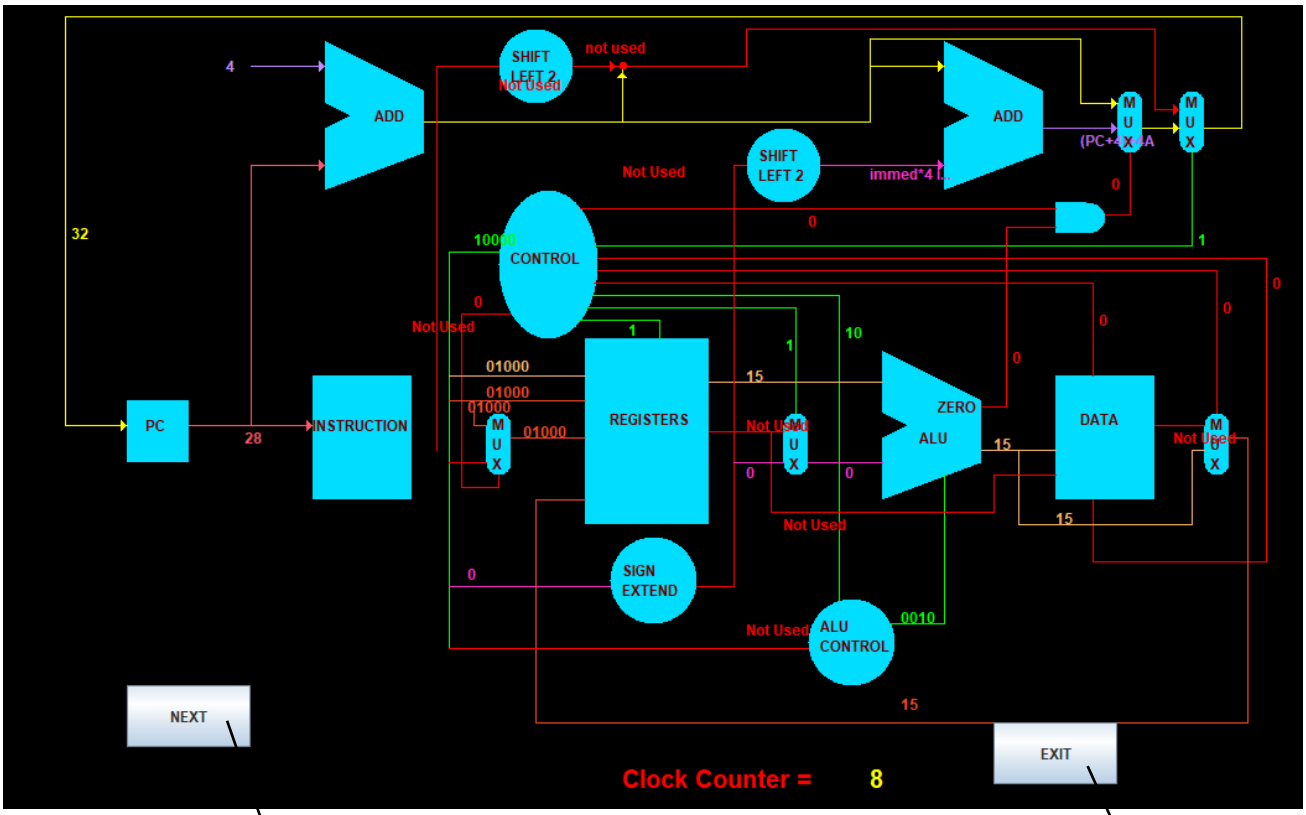


WARNING : IF THE PROGRAM DIDN'T RUN AFTER PRESSING THE "RUN" YOU NEED TO RECHECK THE TEXT YOU ENTER AGAIN IN ALL INSTRUCTIONS !

The Datapath circuit with no instructions:



The data path for the program:



Press to represent next instruction.

Press to exit the program

Warning :

If the next button didn’t change the frame after pressing.. it means that you have reached the end.

The register file with final values for each register :

NAME	NUMBER	VALUE
$0	0	0
$at	1	0
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	15
$t1	9	3
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	0
$sp	29	0
$fp	30	0
$ra	31	0

How the project was split among the team:

1.Ahmed Bahaa

Classes:

1.jr

2.addi

3.slti

4.home

5.animation

6.instruction_ar

2.Mario Sherif

Classes:

1.sw

2.lb

3.nor

4.animation

5.register

6.memory

3.Kirolos Raouf

Classes:

1.sll

2.sb

3.lw

4.home

5.animation

6.instruction_ar

4.Andrew Sameh

Classes:

1.beq

2.jal

3.lbu

4.animation

5.rf

6.registerfile

7.window

5.Basma Magdy

Classes:

1.j

2.add

3.slt

4.animation

5.rf

6.registerfile

7.memory