جامعه الزقازيق _كلية الهندسة

الفرقة : الثالثة هندسة الحاسبات والمنظومات

المقرر: دوائر الحاسب المتكاملة

Name: Eman Mohamed Ahmed Gomaa

Number in section: 29

Group Number:1

Name of project: 8-bit RISC Processor

Introduction

The objective of this course (Integrated Circuit Design) is to design an 8-bit microprocessor**.**

Using me as a student is an application to the ic and orgnization course**.**

Requirements:

- 8-bit data bus

- 16-bit address bus

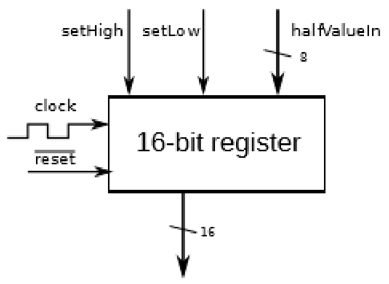
- 8*8-bit general purpose register

Module:

1- Generic 16-bit register:

A 16-bit register is used three times in the design: the jump register, the memory address register, and the instruction register. The same module is instantiated in all three of these cases.

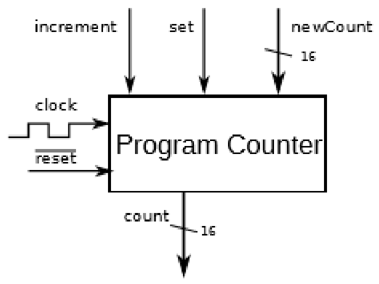
The module has an 8-bit input, which in all three instantiations comes from the data bus. Two signals, setHigh and setLow determine whether this input is stored in the top half or bottom half of the register.



2- Program counter

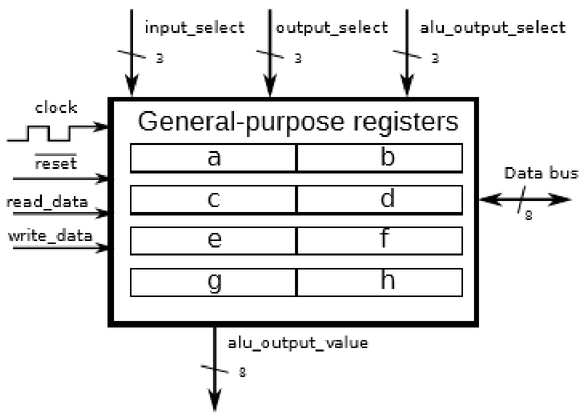
The program counter holds the address of the next instruction byte

and is used to index ROM when fetching instructions.



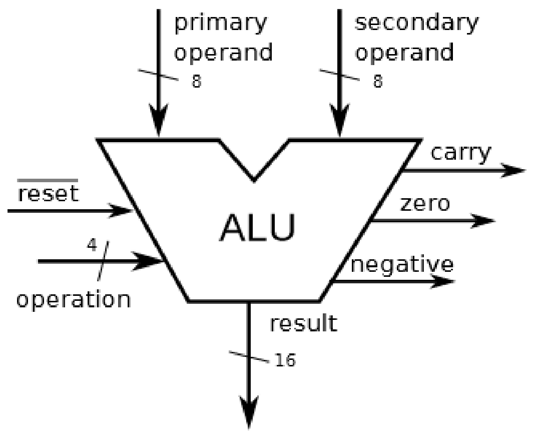
3- General-purpose register block

The general-purpose register block contains 8 eight-bit registers which are used for data manipulation. They are grouped into pairs, creating 4 sixteen-bit registers which can receive the result of a multiply operation.



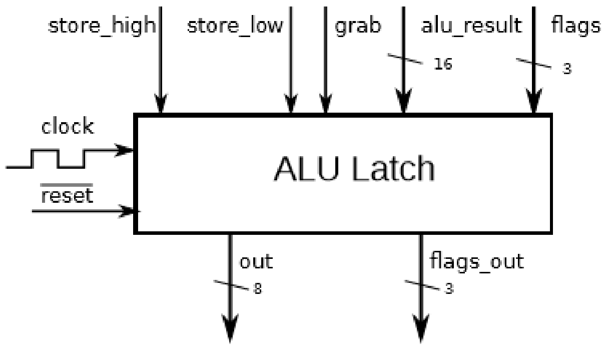
4-Arithmetic logic unit (ALU)

The arithmetic logic unit implements all of the arithmetic operations speci_ed: addition, subtraction, multiplication, logical AND and OR, left and right logical shifts, left and right arithmetic shifts, bitwise complement, and negation. It also contains a passthrough instruction so that the ALU latch can be used as a temporary register for the MOVE operation. Output _ags are set based on the results of the operation.



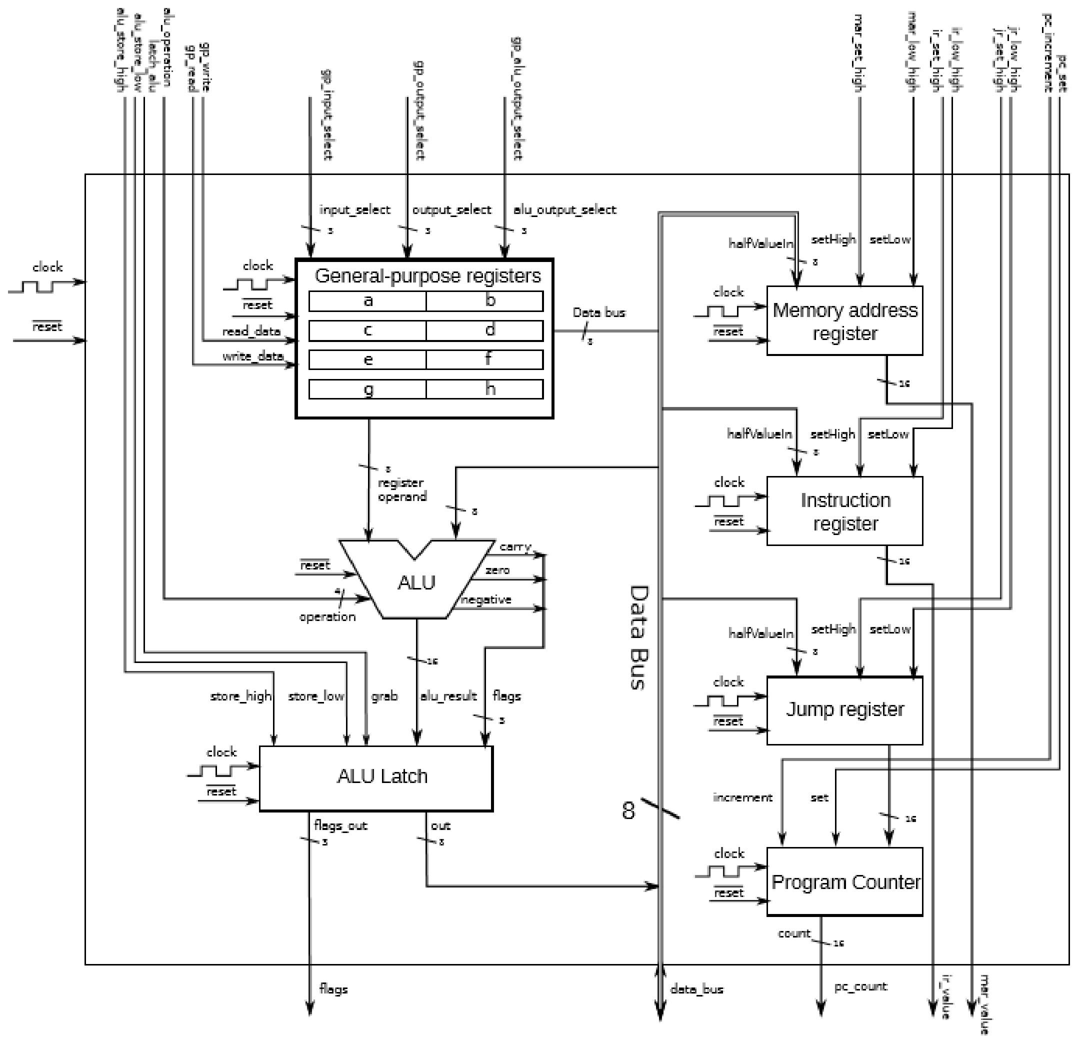
5- ALU Latch:

The ALU latch grabs the result of the ALU operation, holds it, and then puts it on the databus when the store signals are asserted. It also latches the _ags, so that a jump operates based on the last time the result was grabbed.



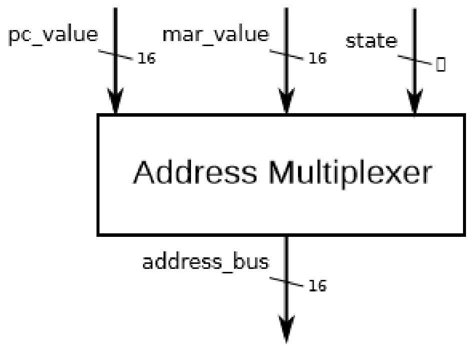
6- Datapath

The datapath module combines the program counter, jump register, general-purpose registers, ALU, ALU latch, memory address register, and instruction register into a single unit connected by a data bus. The data bus is a bidirectional module port, so data can be brought in and out of the chip. A block diagram is:



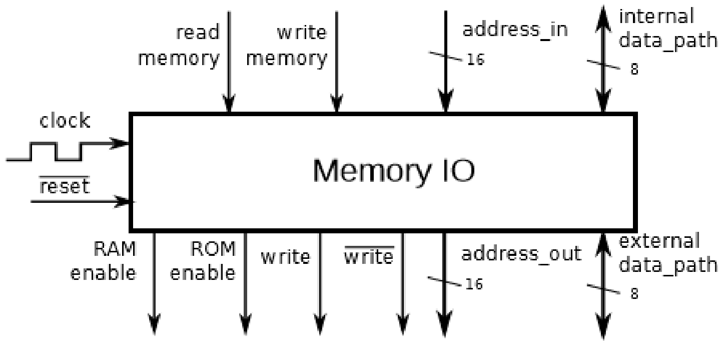
7- Address multiplexer

The address multiplexer switches the output address between the program counter and memory address register based on the state. If the processor is performing a load or store using a memory location, the MAR address is used; otherwise, the program counter is used.



8-Memory IO

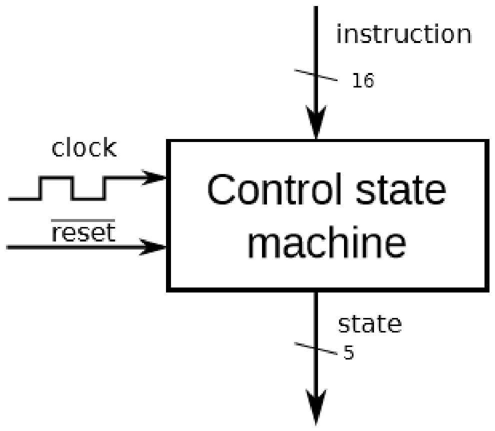
The Memory IO is the main communication block between the memory and the internal CPU modules.



9-Control module

The control module consists of two separate modules: a **state machine** which reads the output of the instruction register and determines what to do on the next clock cycle, and a **signal translation module** which maps the control state into controls signals for all of the other modules.

a. State machine:

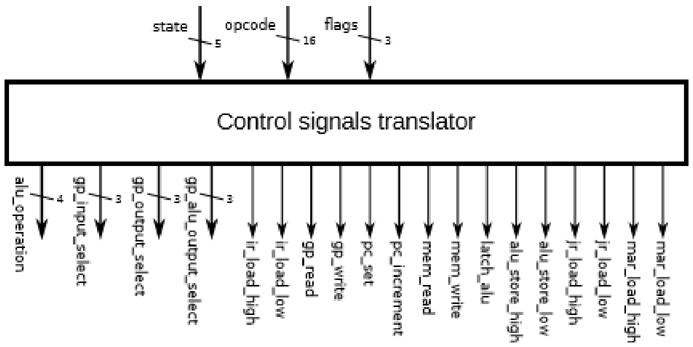


b. Control signal translation

What is the Control signal translation?

Module which translates the control module state into the set of control signals

-Block Diagram:



Mapping of states to control signals

It defines the signal to be transmitted by the translator in each state input of the control state

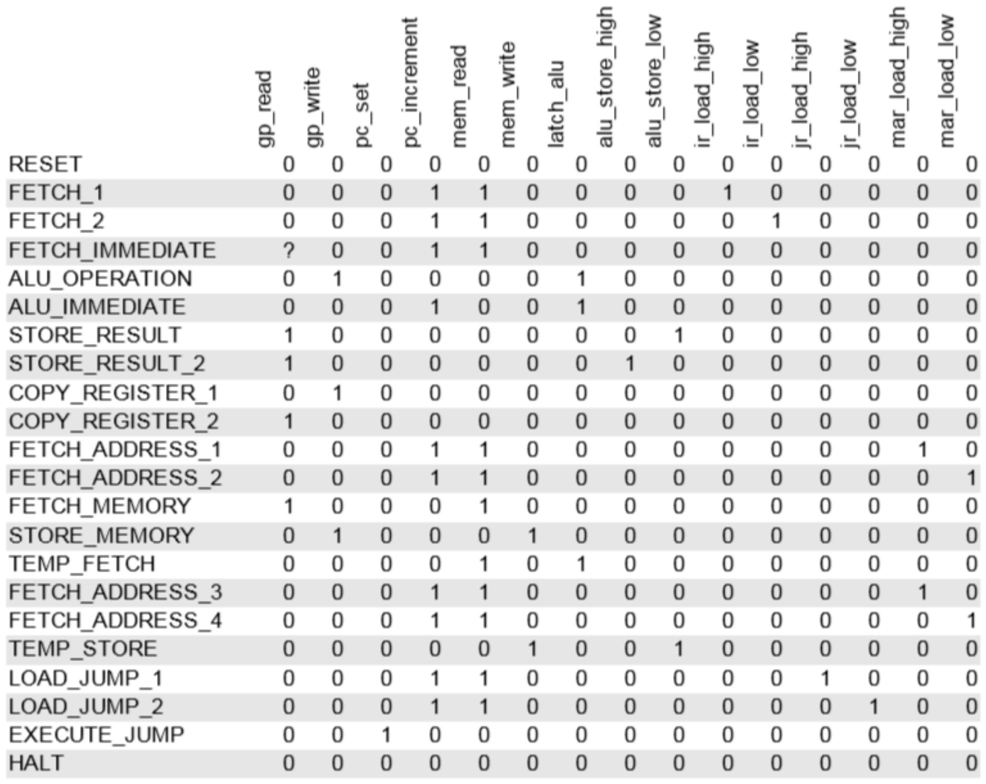
example:

Reset	all signals is zero

Fetch_1 signal in pc_increment= mem_read=ir_load_high =1

Fetch_2 signal in pc_increment= mem_read =ir_load_low=1

and so on

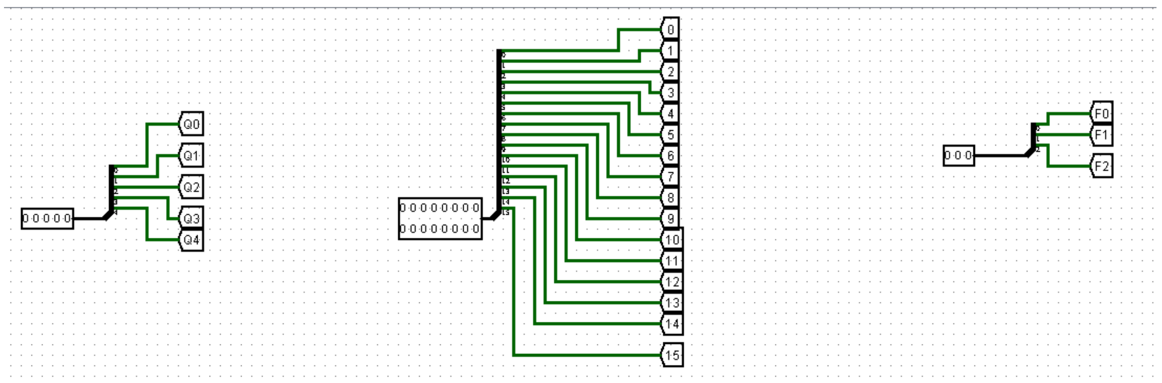


	gp_read	gp_write	pc_set	pc_increment	mem_read	mem_write	latch_alu	alu_store_high	alu_store_low	ir_load_high	ir_load_low	jr_load_high	jr_load_low	mar_load_high	mar_load_low
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FETCH_1	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0
FETCH_2	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0
FETCH_IMMEDIATE	?	0	0	1	1	0	0	0	0	0	0	0	0	0	0
ALU_OPERATION	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0
ALU_IMMEDIATE	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
STORE_RESULT	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
STORE_RESULT_2	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0
COPY_REGISTER_1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
COPY_REGISTER_2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
FETCH_ADDRESS_1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
FETCH_ADDRESS_2	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
FETCH_MEMORY	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
STORE_MEMORY	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
TEMP_FETCH	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
FETCH_ADDRESS_3	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0
FETCH_ADDRESS_4	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
TEMP_STORE	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0
LOAD_JUMP_1	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0
LOAD_JUMP_2	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0
EXECUTE_JUMP	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
HALT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Logisim Implementation:

Input:

- [4:0] State from the control state machine
- [15:0] opcode, Full 16-bit opcode from the instruction register
- [2:0] alu_flags, Carry/Zero/Negative flags from ALU



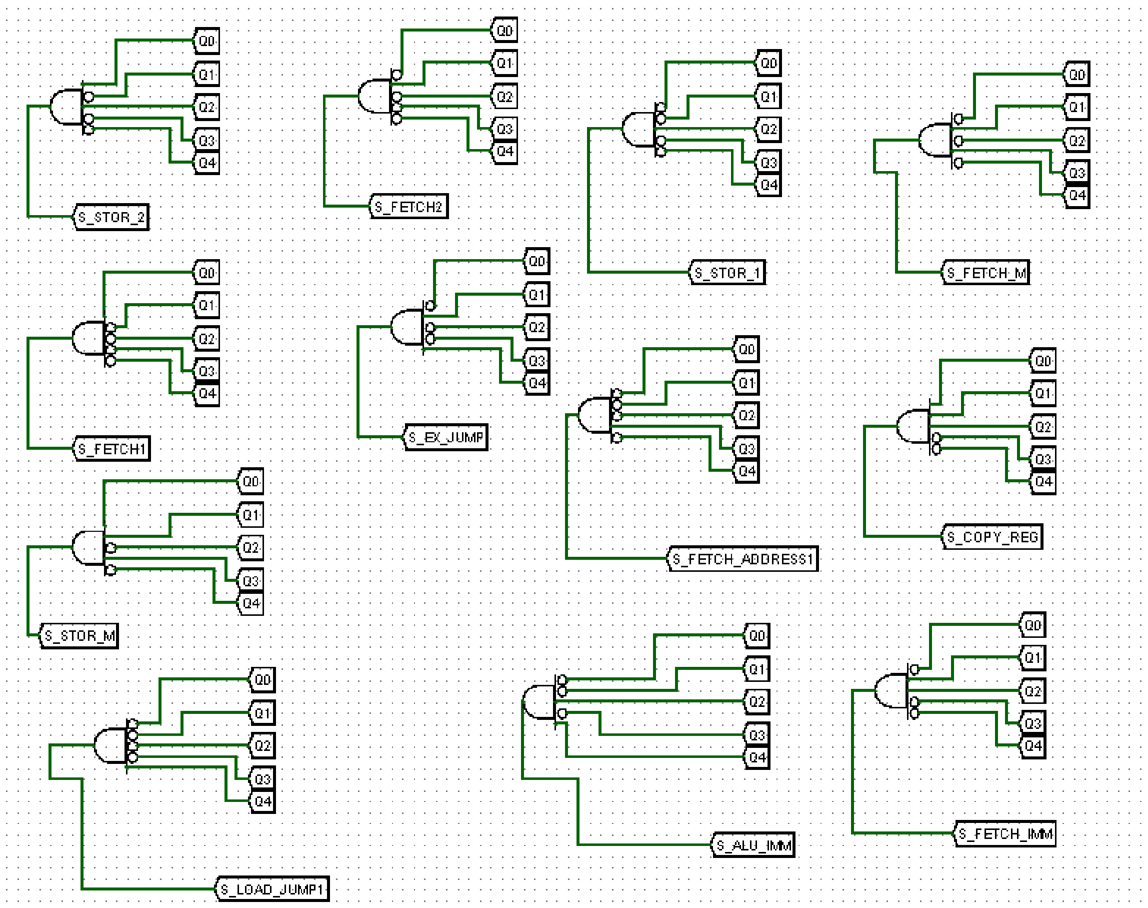
Input check:

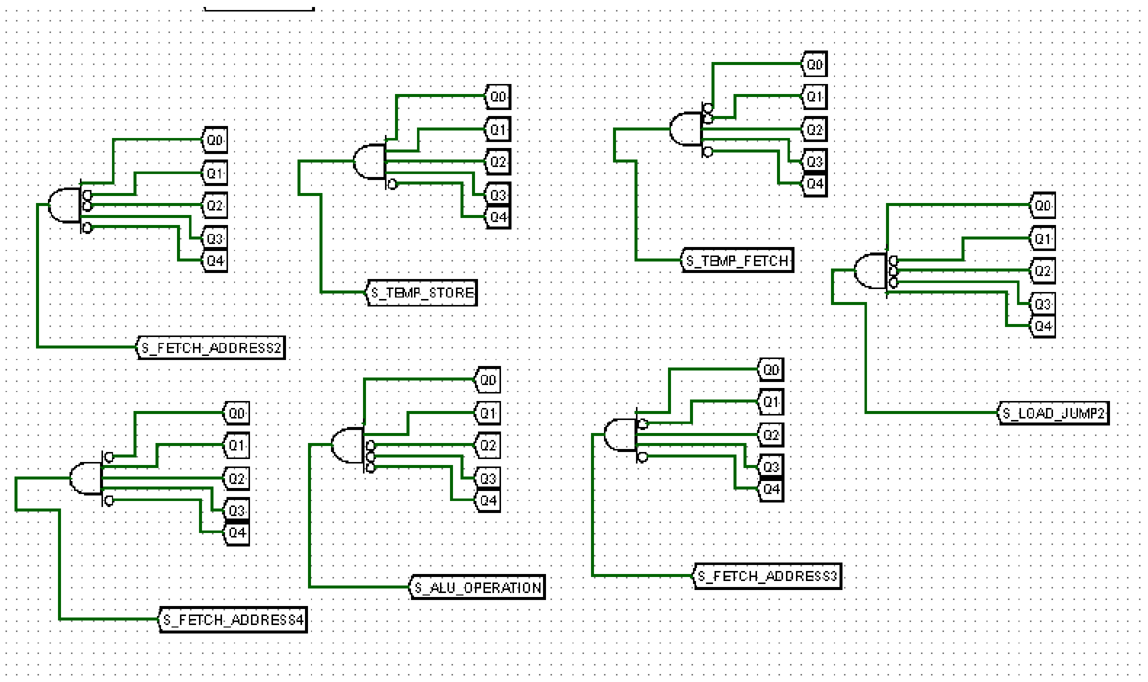
First: What is the state of State machine:

The input is tested from state machine by logic and gate

S_FETCH_1	5bit	d1	b00001
S_FETCH_2	5bit	d2	b00010
S_ALU_OPERATION	5bit	d3	b00011
S_STORE_RESULT_1	5bit	d4	b00100
S_STORE_RESULT_2	5bit	d5	b00101
S_FETCH_IMMEDIATE	5bit	d6	b00110
S_COPY_REGISTER	5bit	d7	b00111
S_FETCH_ADDRESS_1	5bit	d8	b01000
S_FETCH_ADDRESS_2	5bit	d9	b01001
S_FETCH_MEMORY	5bit	d10	b01010
S_STORE_MEMORY	5bit	d11	b01011
S_TEMP_FETCH	5bit	d12	b01100
S_FETCH_ADDRESS_3	5bit	d13	b01101
S_FETCH_ADDRESS_4	5bit	d14	b01110

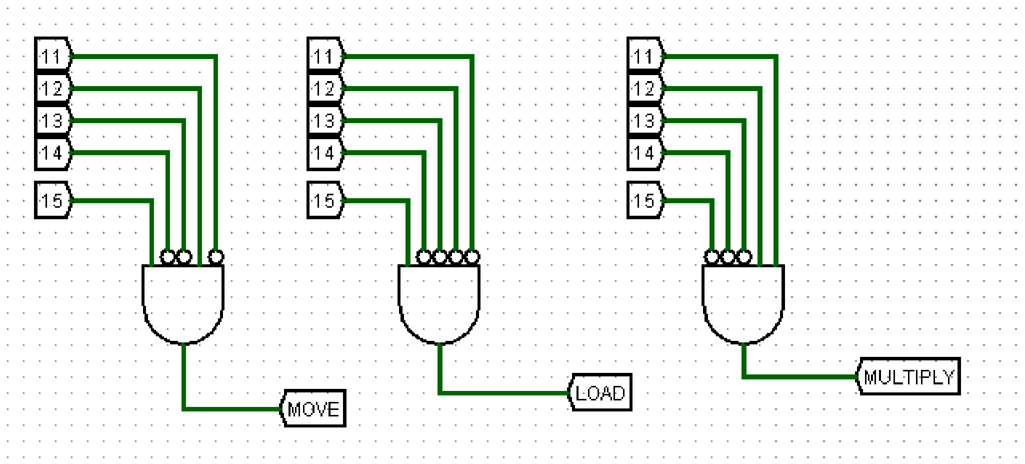
S_TEMP_STORE	5bit	d15	b01111
S_LOAD_JUMP_1	5bit	d16	b10000
S_LOAD_JUMP_2	5bit	d17	b10010
S_EXECUTE_JUMP	5bit	d18	b10011
S_ALU_IMMEDIATE	5bit	d20	b10100





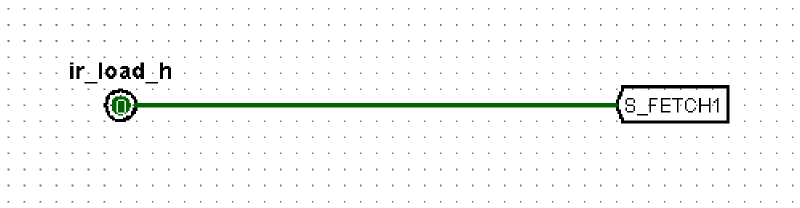
Secnod: What is the type of instruction represented in the opcode[11:15]

MOVE	5bit	b10010
LOAD	5bit	b10000
MULTIPLY	5bit	b00011



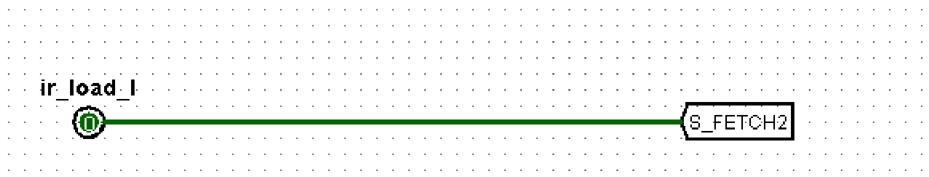
The Output circuit:

- ir_load_high: Load the high 8 bits of the instruction from the data bus



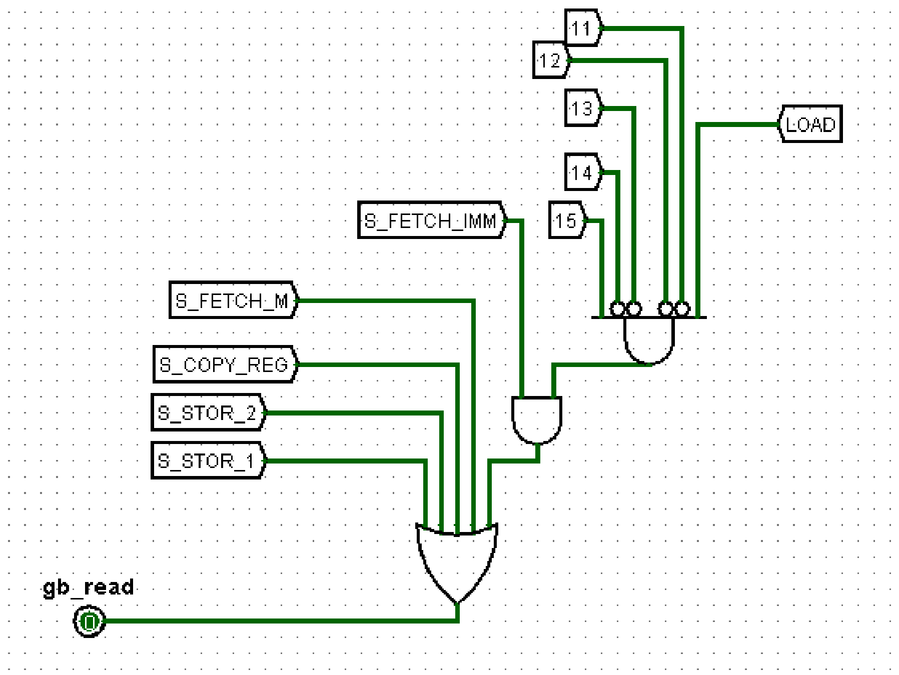
ir_load_high will activate if state == S_FETCH_1

- ir_load_low: Load the low 8 bits of the instruction



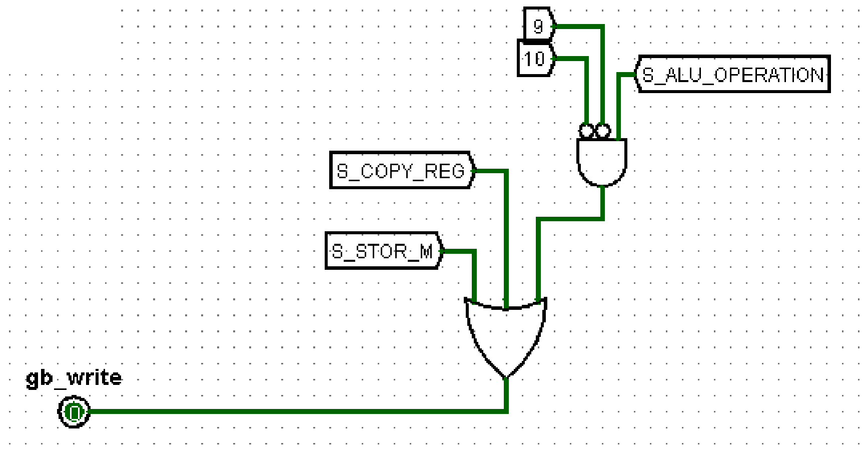
ir_load_low will activate if state == S_FETCH_2

- gp_read: Read a value from the data bus into a register. Read into the registers if we have a store, a copy, or are loading an immediate into a register



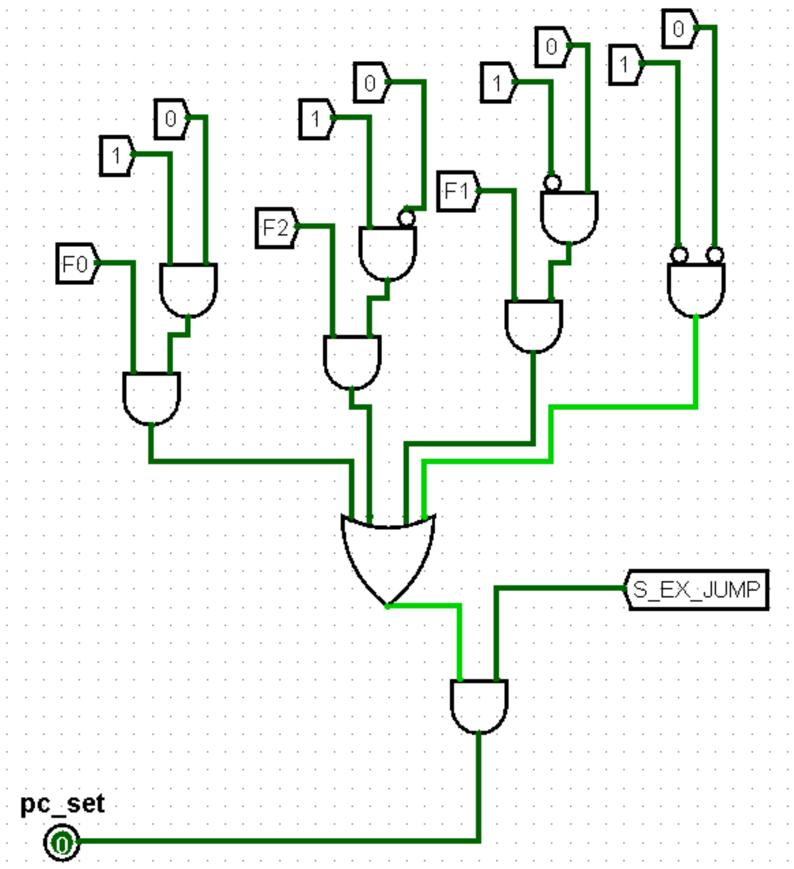
gp_read will activate if state ==S_STORE_RESULT_1 OR state ==S_STORE_RESULT_2 OR state ==S_COPY_REGISTER ORstate ==S_FETCH_MEMORY OR (state ==S_FETCH_IMMEDIATE AND opcode[15:11] ==LOAD)

- gp_write, Write a value from the register onto the data bus. Write from the registers if we have a register ALU operation or a store



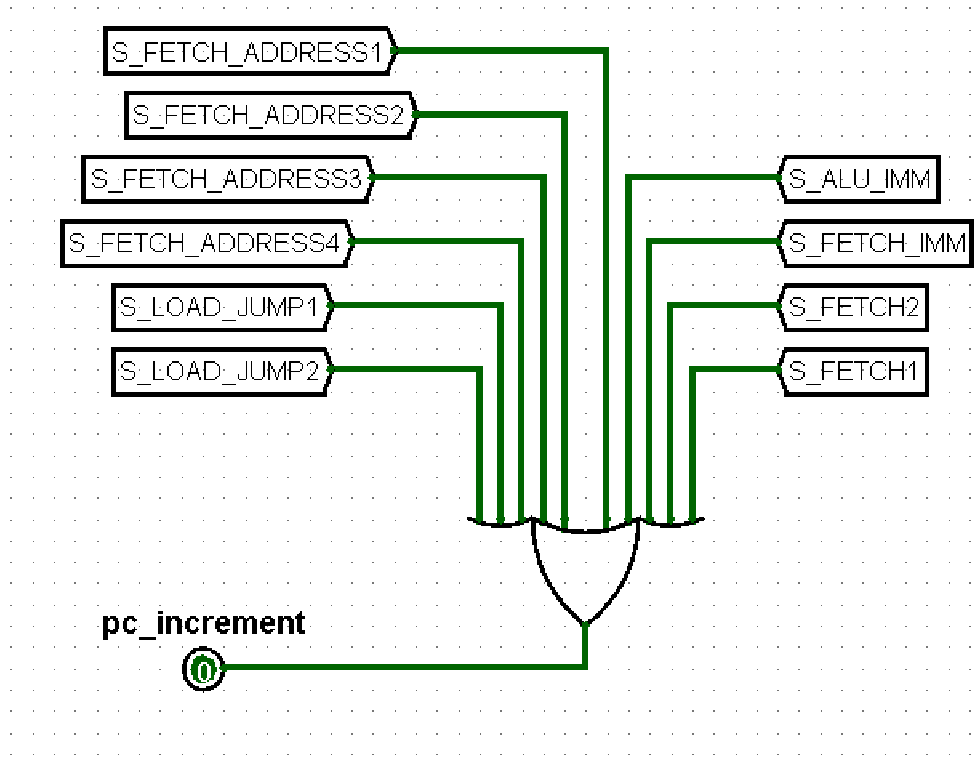
gp_write will activate if (state ==S_ALU_OPERATION AND opcode[10:9] ==b00) OR state ==S_STORE_MEMORY OR state ==S_COPY_REGISTER

- pc_set, Set the program counter from the jump register



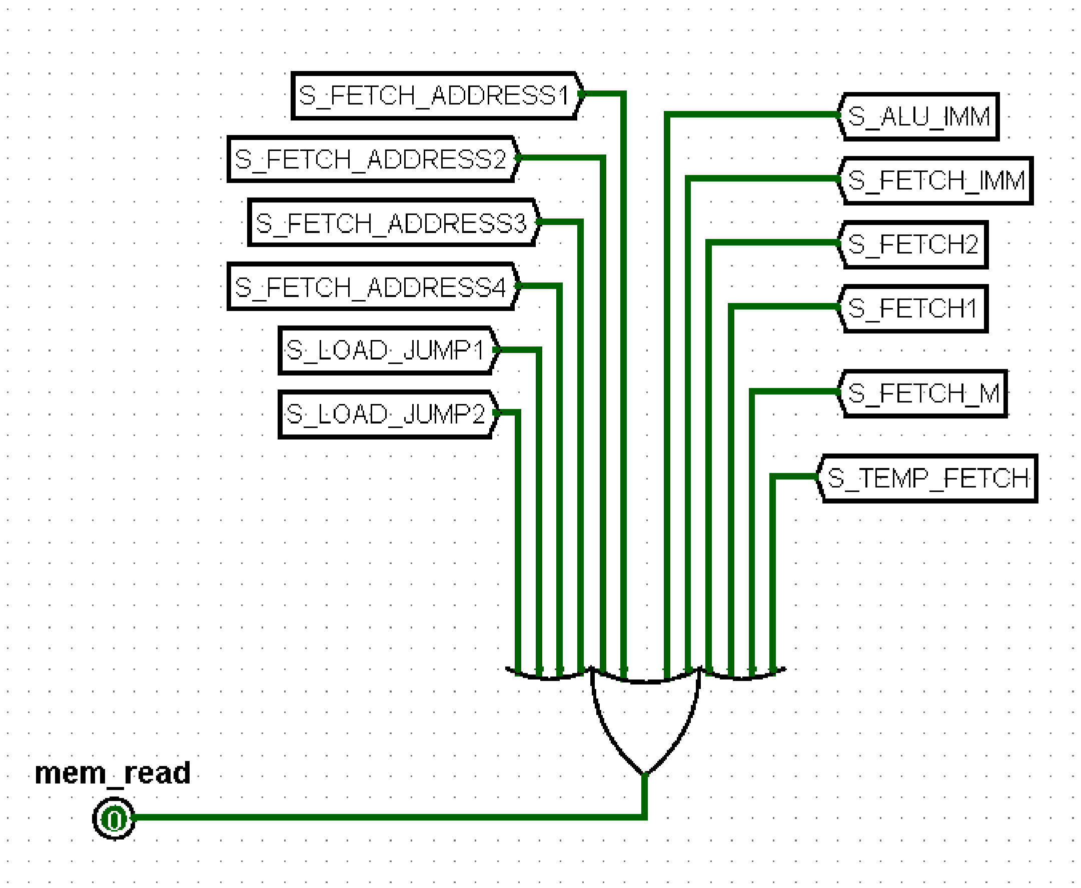
pc_set will activate if state ==S_EXECUTE_JUMP AND (opcode[1:0] ==b00 OR (opcode[1:0] ==b01 AND alu_flags[CARRYFLAG] == b1) OR (opcode[1:0] ==b10 AND alu_flags[ZEROFLAG] == b1) OR (opcode[1:0] ==b11 AND alu_flags[NEGFLAG] == b1))

- pc_increment, Increment the program counter



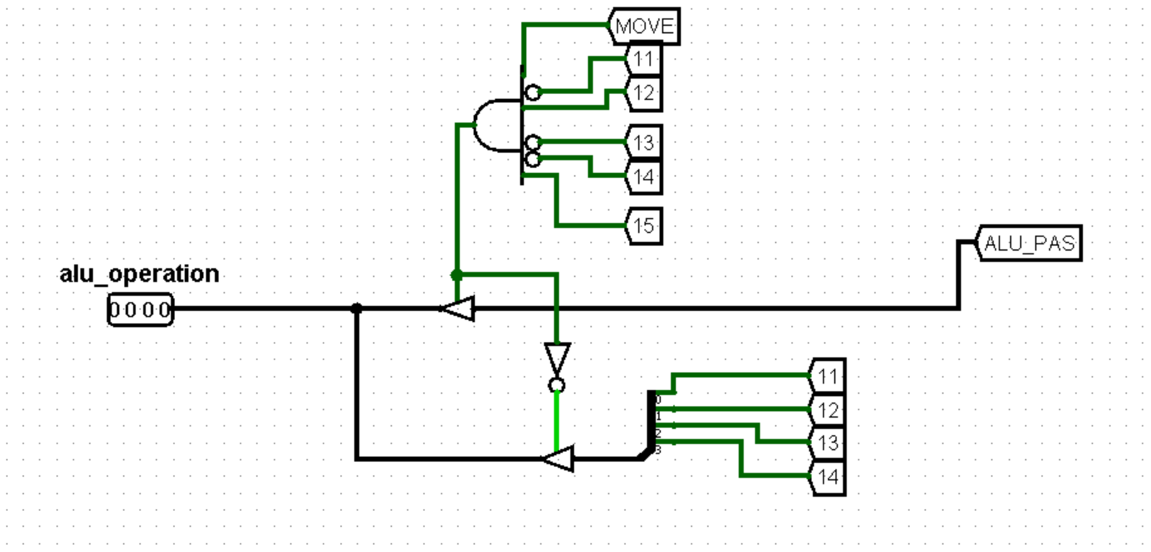
pc_increment will activate if state ==S_FETCH_1 OR state ==S_FETCH_2 OR state ==S_FETCH_IMMEDIATE OR state ==S_ALU_IMMEDIATE OR state ==S_FETCH_ADDRESS_1 OR state ==S_FETCH_ADDRESS_2 OR state ==S_FETCH_ADDRESS_3 OR state ==S_FETCH_ADDRESS_4 OR state ==S_LOAD_JUMP_1 OR state === 'S_LOAD_JUMP_2

- mem_read, Read a value from memory onto the data bus

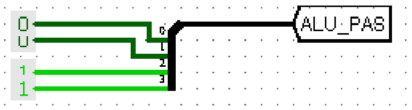


mem_read will activate if state ==S_FETCH_1 OR state ==S_FETCH_2 OR state == S_FETCH_IMMEDIATE OR state == S_ALU_IMMEDIATE OR state == S_FETCH_ADDRESS_1 OR state == S_FETCH_ADDRESS_2 OR state ==S_FETCH_MEMORY OR state ==S_TEMP_FETCH OR state ==S_FETCH_ADDRESS_3 OR state == S_FETCH_ADDRESS_4 OR state == S_LOAD_JUMP_1 OR state == S_LOAD_JUMP_2

- [3:0] alu_operation:Signals directly from the opcode.

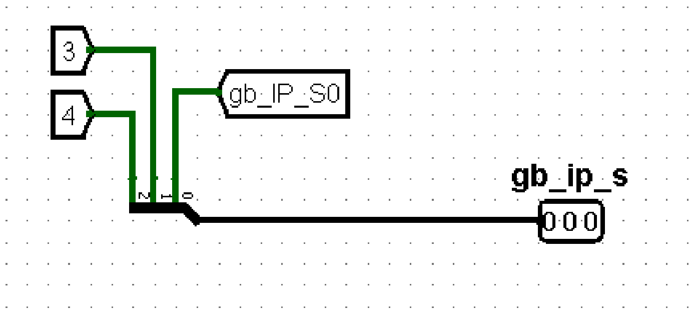


ALU_PASSTHROUGH 4bit b1100

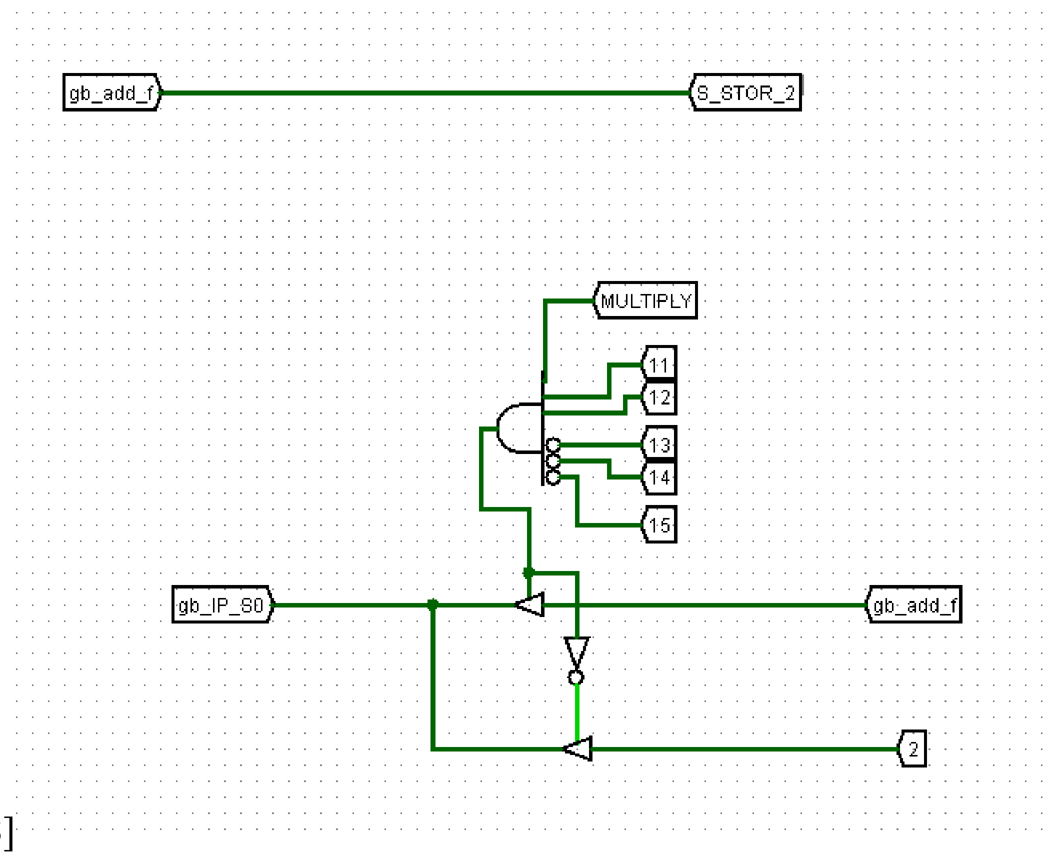


alu_operation = ALU_PASSTHROUGH if (opcode[15:11] ==MOVE) is true if not true alu_operation = opcode [14:11]

- [2:0] gp_input_select is the "primary operand" used for unary operations



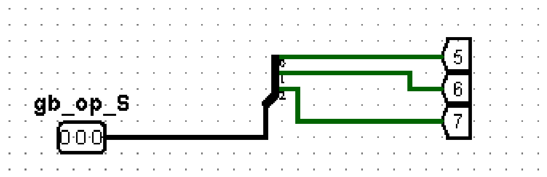
gp_input_select[2:1] =



opcode[4:3]

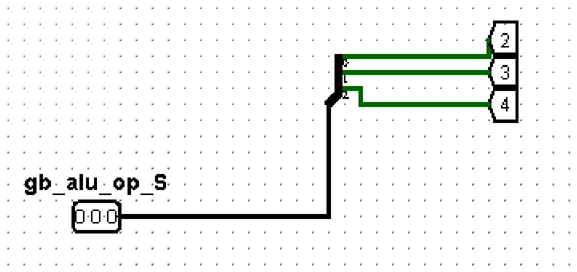
gp_input_select[0] = gp_address_force if (opcode[15:11] == MULTIPLY)is true else gp_input_select[0] = opcode[2]gp_address_force: Bit used to force a particular address when using a multiply. gp_address_force = S_STORE_RESULT_2

- [2:0] gp_output_select: Register select for GP registers to data bus.



gp_output_select = opcode[7:5]

- [2:0] gp_alu_output_select : Register select for GP registers directly to ALU

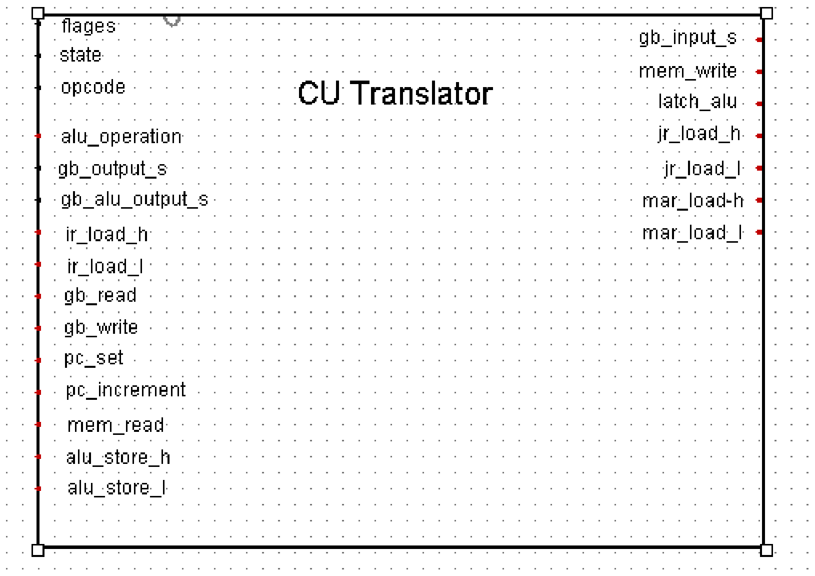


gp_alu_output_select = opcode[4:2]

The output remains and is explained by someone else in the Team:

- mem_write, Write a value from the data bus out to memory (RAM)
- latch_alu
- alu_store_high, Write the high 8 bits of the ALU result to the data bus
- alu_store_low, Write the low 8 bits of the ALU result to the data bus
- jr_load_high, Load the high 8 bits of the jump destination into the jump register
- jr_load_low, Load the low 8 bits of the jump destination
- mar_load_high, Load the high 8 bits of the memory address into the MAR
- mar_load_low, Load the low 8 bits of the memory address

Final Logisim Block:



Reference:

https://stanford.edu/~sebell/oc_projects/ic_design_finalreport.pdf