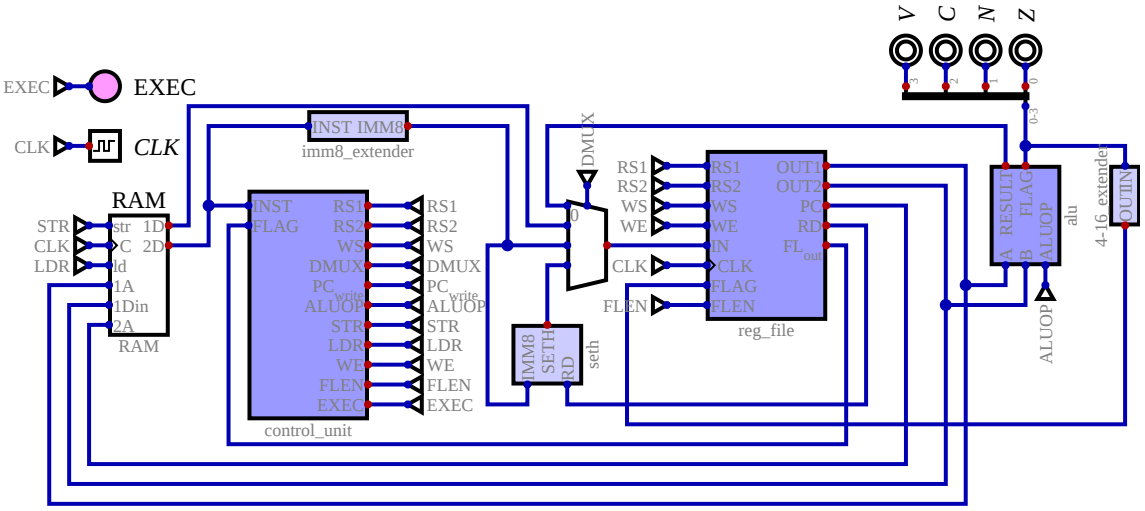
Assignment 1

Student Details

- **uid**: 'u7283652'
- **name**: 'Razeen Wasif'

CPUs

Basic CPU

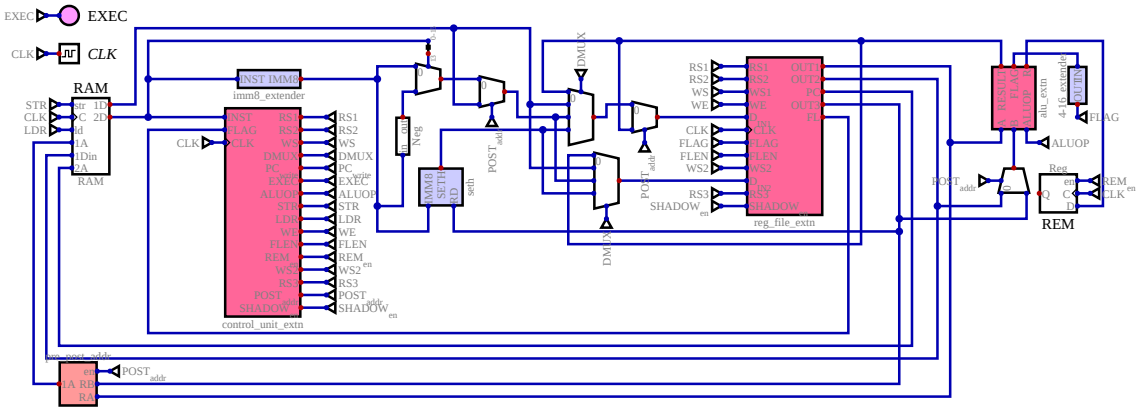


Test Results

Test: CPU - successful

ADD: passed
AND: passed
SUB: passed
XOR: passed
STR: passed
LDR: passed
SETH: passed
MOVL: passed
GP_PC: passed
READ_FLAGS: passed
FLAGS: passed
ALU: passed
FIB: passed
MEM: passed
REC: passed
JUMPS: passed
COND: passed

Extended CPU



Test Results

Test: CPU - successful

ADD: passed
AND: passed
SUB: passed
XOR: passed
STR: passed
LDR: passed
SETH: passed
MOVL: passed
GP_PC: passed
READ_FLAGS: passed
FLAGS: passed
ALU: passed
FIB: passed
MEM: passed
REC: passed
JUMPS: passed
COND: passed

Report

Report

For the extension, I wanted to primarily focus on the ALU, implementing multiplication and division functions such that my cpu could handle any form of arithmetic. To that end, I also designed a Carry Look-ahead Adder to increase the speed and efficiency of the cpu's arithmetic calculations. Additionally I also implemented pre / post offset memory operations and, given there's only 4 general purpose registers, it would be very convenient to have more registers to work with so I created shadow registers. In total, these were the following extensions that were added:

Extension	Description	Syntax
sign extension	stores a negative value into a register	movn rd, IMM8
CLA	Carry lookahead adder to increase arithmetic efficiency	n/a

Logical Bitshift	Custom barrel shifter that uses value in RB to left shift value in RA	lsl rd, ra, rb
Circular Shift	Circularly shifts / Rotates the bits in RA by the shift amount in RB	ror rd, ra, rb
Pre/Post offset memory addressing	Operations that can offset the base address before or after storing/loading the data	ldr/str rd, [ra, rb] ldrr/strr rd, [ra], rb
Dual Ported Register File	Expanded Register file to have two write ports and 3 read ports. There are two more outputs for PC and FL.	n/a
Shadow Registers	Added shadow registers for the 4 general purpose registers	shadowreg
Multiplication	Multiplies RA by RB and stores in RD	mult rd, ra, rb
Division	Divides RA by RB. Remainder stored in a dedicated register	div rd, ra, rb

Overview

The second write port on the register file is primarily used for writing the post offset address to the register. The first write port has been given the highest priority. I included a dedicated offset register in the register file to store the offset value for pre and post offset memory addressing. Storing the offset into a separate register allows for more flexibility and is simpler to keep track of and they can modified independently.

The shadow registers can be switched to using the 'shadowreg' instruction. This will only allow access to the shadow registers and the original registers won't be affected. This implementation of shadow registers were accomplised by using an extra unused opcode. The main decoder in the control unit uses a JK flip-flop to enable or disable the shadow registers.

Using the ARM book as reference, I designed the multiplier, divider and the16-bit CLA using four 4-bit CLA blocks. Each 4-bit CLA block takes in two inputs and a carry in. The block simultaneously generates the sum and determine if the carry out will be produced using AND and OR gates. Because logic gates have lower delay, the next 4-bit CLA block will receive the carry out much faster. This improvement makes a big difference especially for performing arithmetic on large 16-bit numbers.

The multiply component was impleneted using 4-bit multipliers to generate partial products which were shifted left accordingly and then added using the CLA to create the final product. This approach of parallel multiplication reduces the complexity of the operation compared to long multiplication, especially for a simple processor such as this cpu. Using the CLA rather than an adder also leads to better performance in terms of speed and lower power consumption. Since multiplying two 16 bit numbers can result in a 32-bit product, I decided to handle this by truncating and setting the overflow flag when more than 16 bits are used.

The Division component proved to be quite inefficient. It was implented using the long division algorithm which was the simplest way I found to build the division circuit. The only problem is that as the number of bits increases, the delay of an N-bit divider increases proportionally to N^2 [2]. This noticeably decreases the cpu's execution time having to wait atleast a second or two for the division tests in the ALU to be complete. The remainder that's generated from the division gets stored in a dedicated remainder register. To handle some edge cases of division, such as division by 0, when both inputs are 0, signed division I used multiplexers to hardcode the answers to either 0 or 1 as well as set the overflow flag. While this may not be an efficient implementation, it does pass the tests and produces correct answers.

ISA Changes

I added a few new instructions for the implemented operations. I added the 'MOVN' instruction which follows the same architecture as the 'MOVL' instruction. Next I added the instruction to switch to shadow registers labelled 'SHADOWREG'. All it does is sends the opcode '0011' to the control unit which does the enabling of the registers.

For post offset storing and loading, I figured it'd be easier to have them as separate instructions to pre offset ldr and str. That way, depending on the opcode I could either add the base with the offset before or after the memory operation. I used the last two opcodes to encode post offset storing as 'STRR' and loading as 'LDRR'. Other than that, I added the new alu operations.

Microarchitecture Changes

In the ALU, I replaced the subtractor with the CLA for faster performance. The register file has a few extra inputs, namely the second write port, third read port and the a input that enables the shadow register. An extra DMUX is also added to the CPU to send data in for the second write port.

There is also a 'pre_post_addr' component in the cpu which is used to add the base address with the offset or when the 'POST_addr' pin is enabled, it passes the base address through without modification and offsets it after the memory operation has been performed.

The control unit now has a clock input which is used to enable / disable the JK flip-flop for the shadow register. I've added a few extra control signals:

Control signal	Usage
REM_en	This signal enables the remainder register to be written to when division is being performed
WS2	Selects which register to write to on the second write port
RS3	Selects which register to read on the thrid read port
Post_addr	This signal acts as a enable input to a mux that chooses whether to send OUT2 or OUT3 as the B input to the ALU.
SHADOW_en	enables the shadow registers for use

When the load or store instruction is enabled, OUT2 become RD and OUT3 becomes RB, so the Post_addr signal essentially lets OUT3 (RB) into the ALU rather than OUT2 inorder to calculate the post offset. It also overwrites the DMUX input and the lets the post offset value be written into the register.

Analysis and Tradeoffs

Overall I'm happy with the implementation of my extensions. The CPU has a max frequency of 0.002 GHz which might seem paltry but given the CPUs simplicity, such a low max frequency makes sense and is fine for what this cpu is worth. Having the CLAs improves the overall performance of the CPU since both subtraction and multiplication uses them as well. Having the pre/post offset memory addressing also prevents the use of mutiple cycles to perform those operations. The overall style and design might leave much to be desired including the excessive number of control signals. I tried my best to apply abstraction wherever possible to clean things up. Lastly I'm somewhat unsatisfied with my division circuit. Despite it working perfectly fine and being more accurate than some faster division algorithms which relies on estimates, its signifcantly slow.

Testing

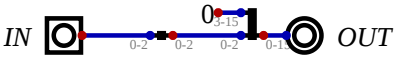
In the extended_cpu file I added some additional tests for the ALU operations. I also wrote a program in the demo.quac file which runs some code to showcase the extensions.

References

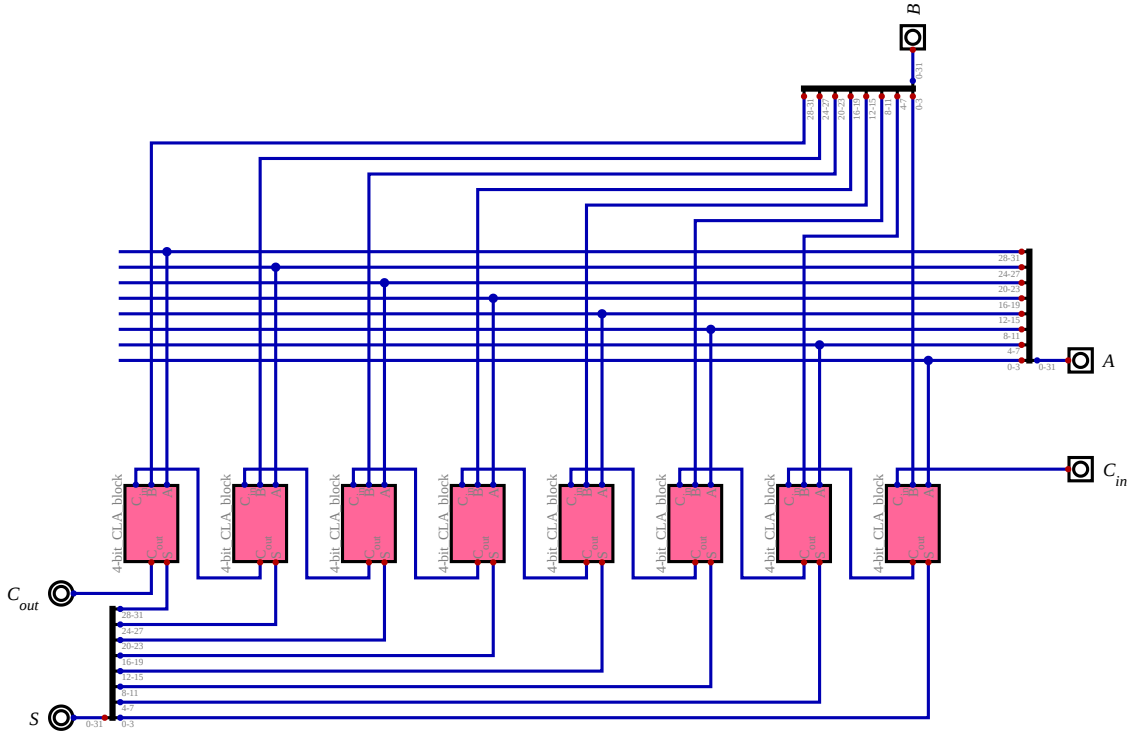
- [1] Re-used components from labs include: 3-16_extender.dig, 4-16_extender.dig, alu.dig, control_unit.dig, counter.dig, DMUX.dig, flags_setter.dig, imm8_extender.dig, reg_file.dig, seth.dig, zero_comparator.dig.
- [2] Harris, S.L. and Harris, D.M. (2016) *Digital Design and computer architecture: ARM edition* . ‘Chapter 5’. Amsterdam etc.: Elsevier.

Additional Circuits

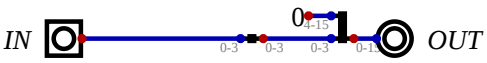
3-16_extender.dig.svg



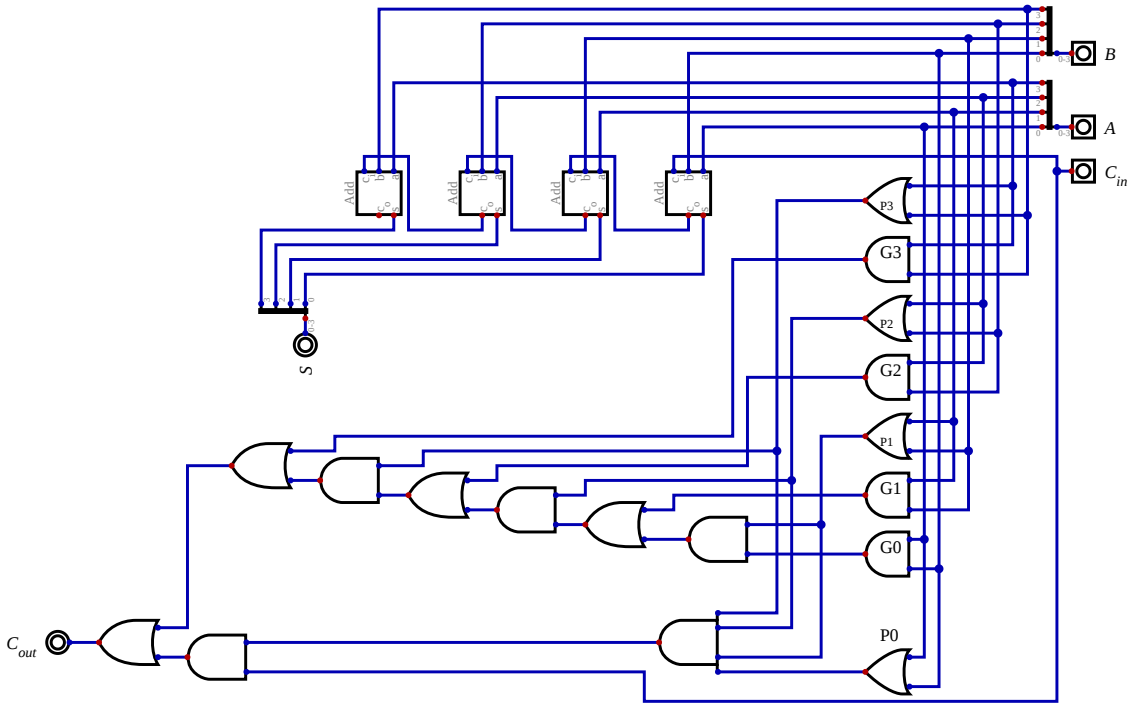
32-bit_CLA.dig.svg



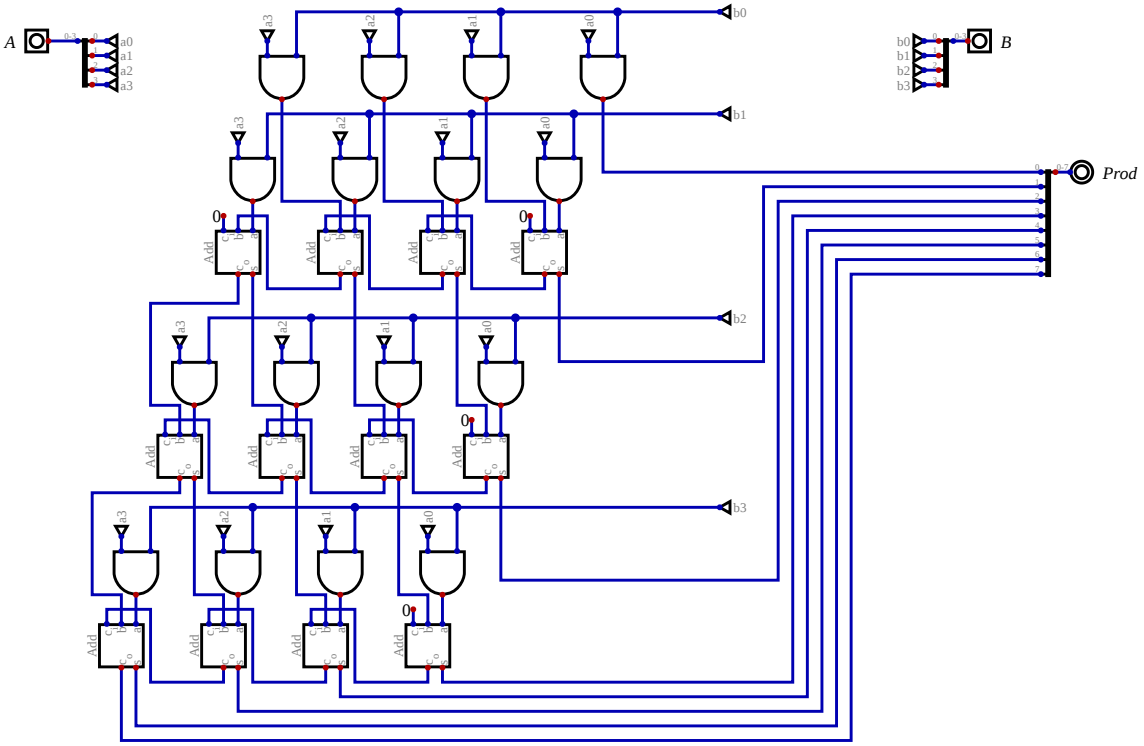
4-16_extender.dig.svg



4-bit_CLA_block.dig.svg

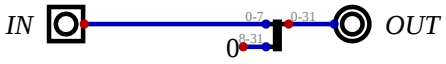


4-bit_multiplier.dig.svg

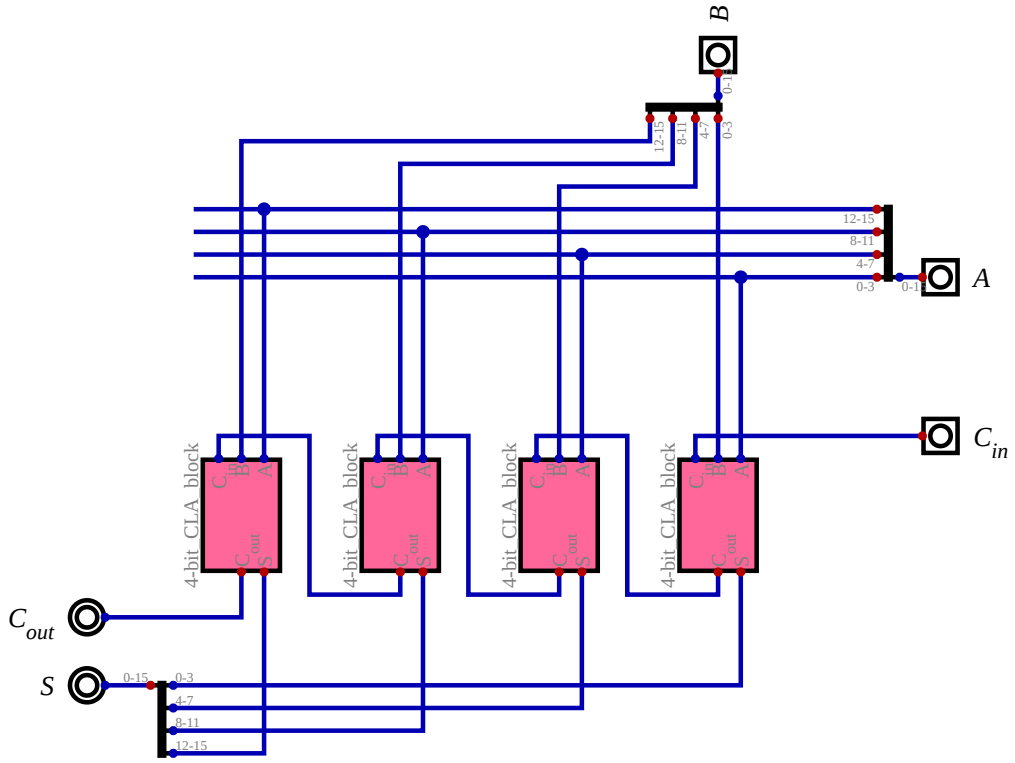


8-16_extender.dig.svg

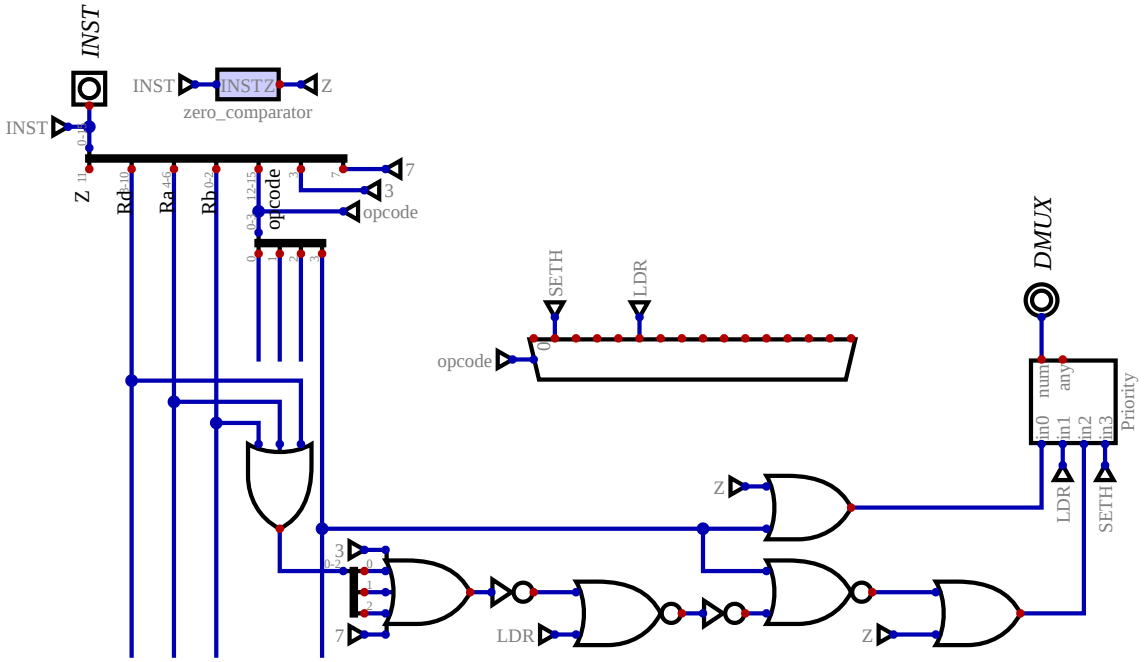
8-32_extender.dig.svg



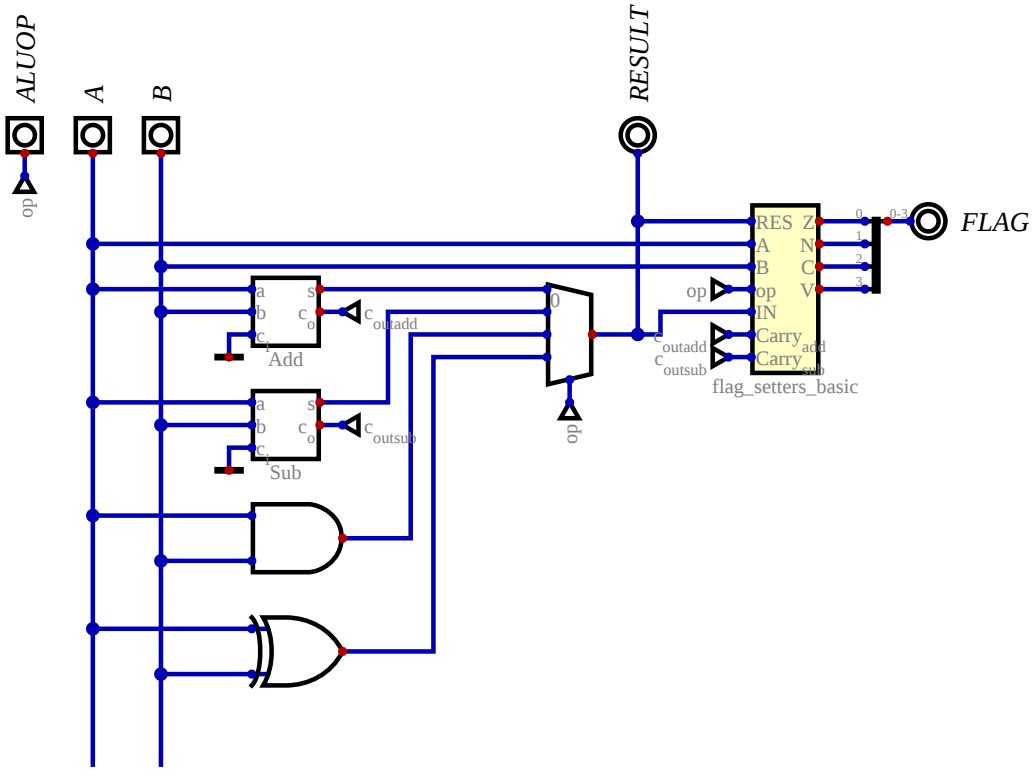
CLA.dig.svg



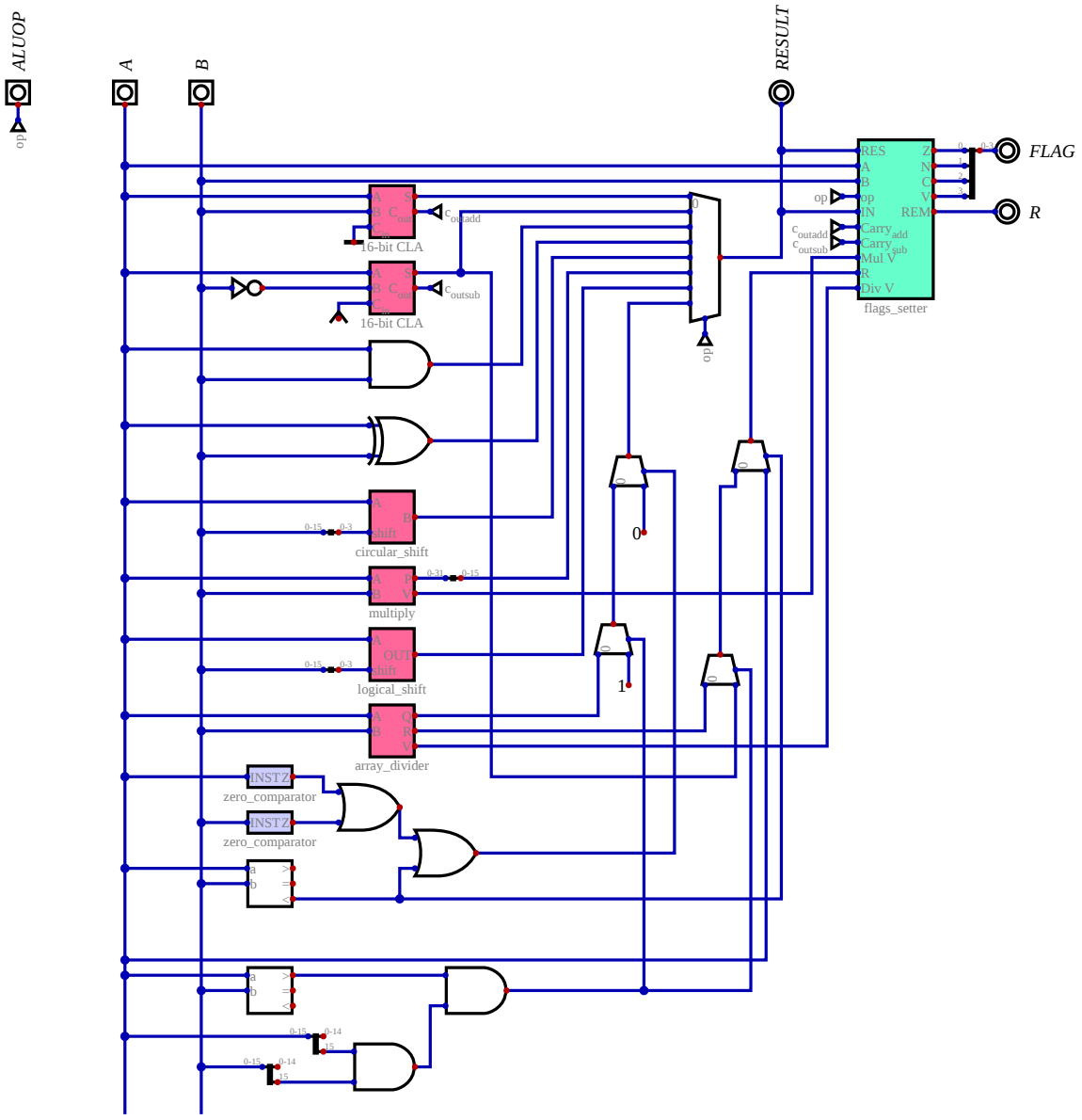
DMUX.dig.svg



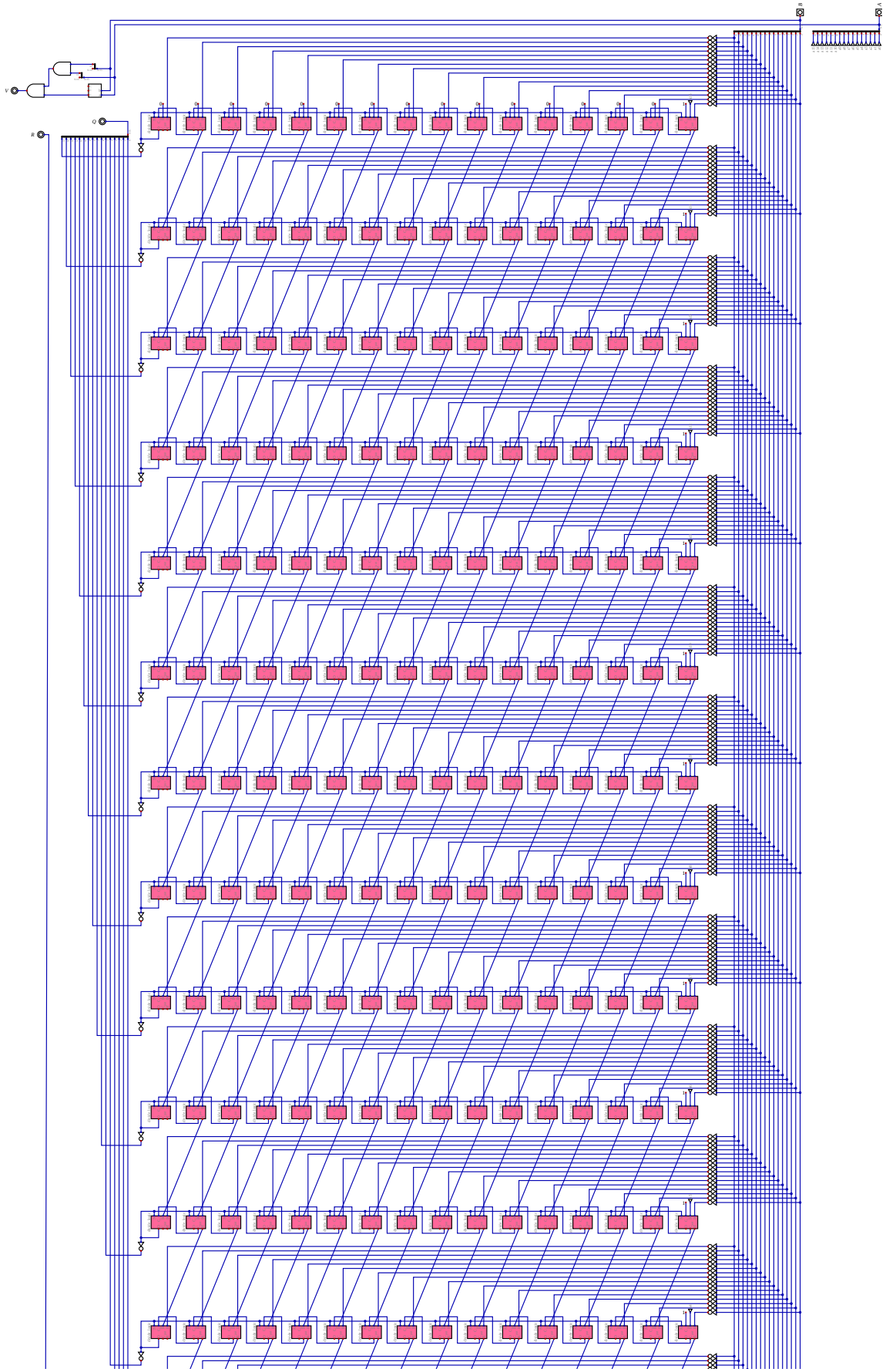
alu.dig.svg

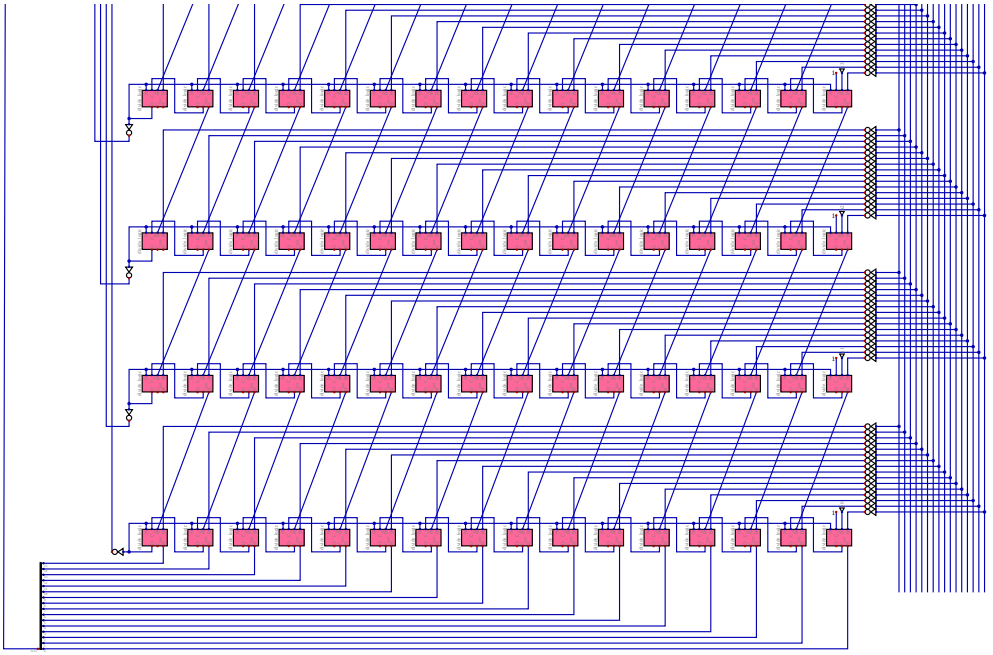


alu_extn.dig.svg

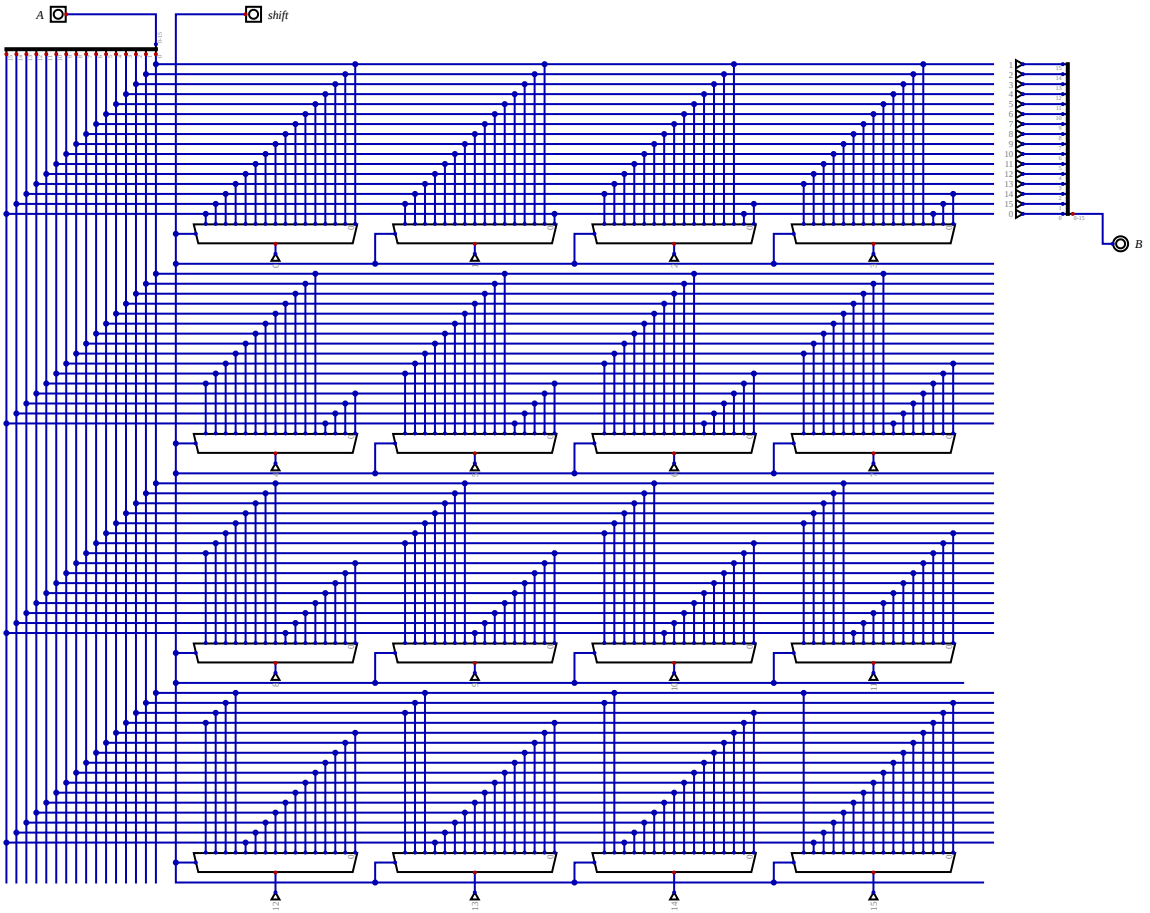


array_divider.dig.svg

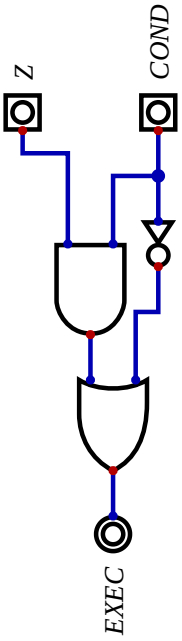




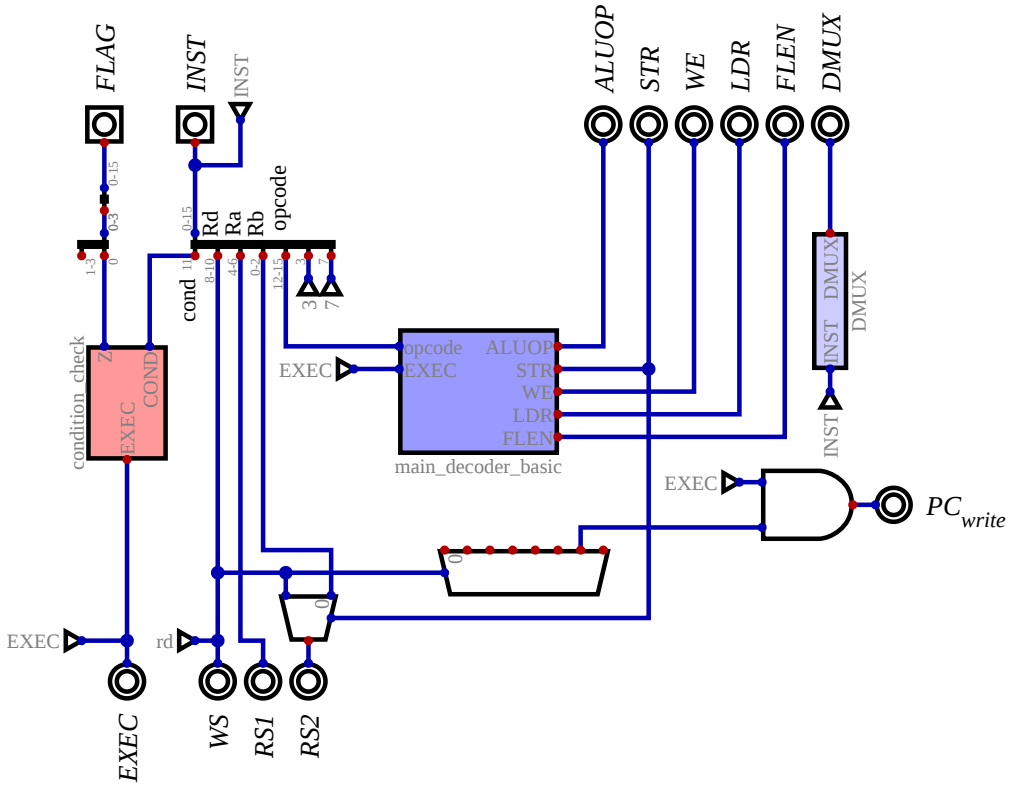
circular_shift.dig.svg



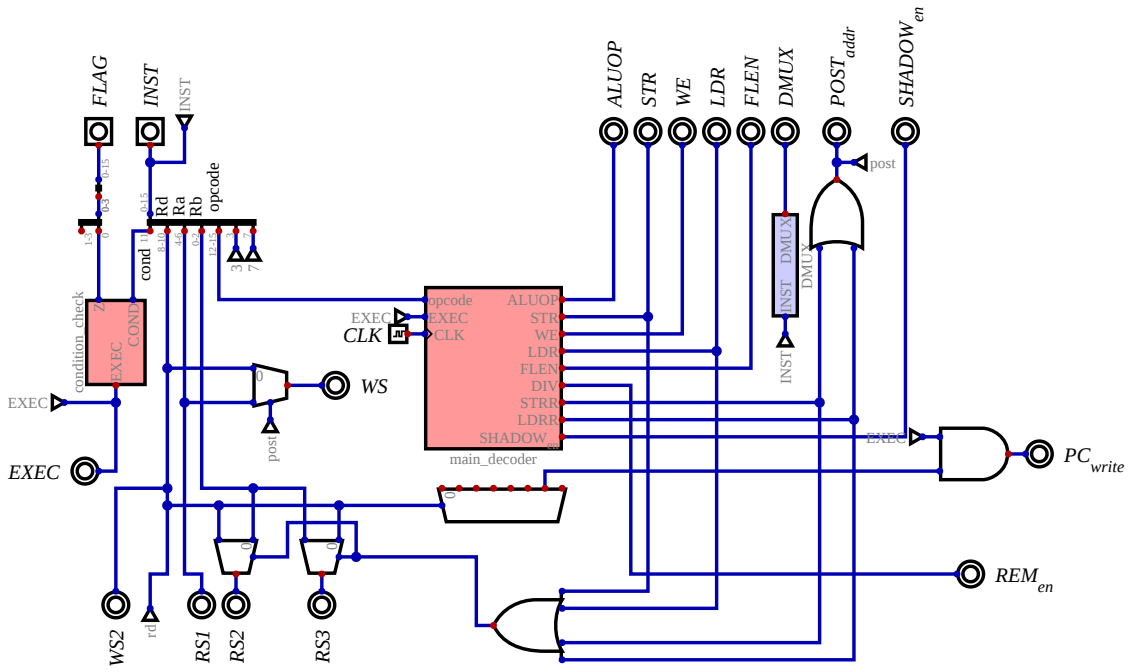
condition_check.dig.svg



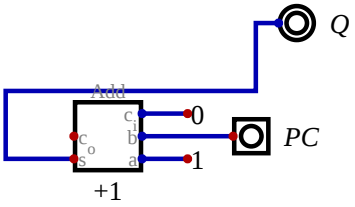
control_unit.dig.svg



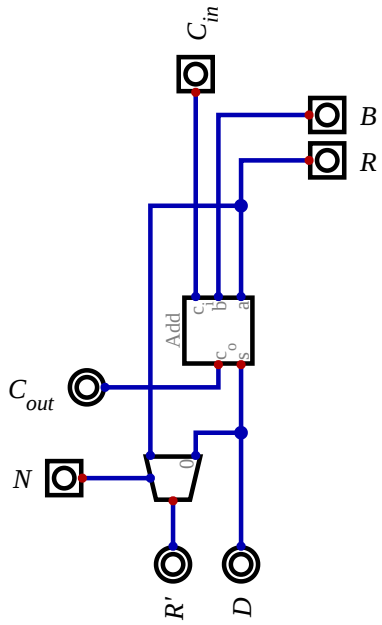
control_unit_extn.dig.svg



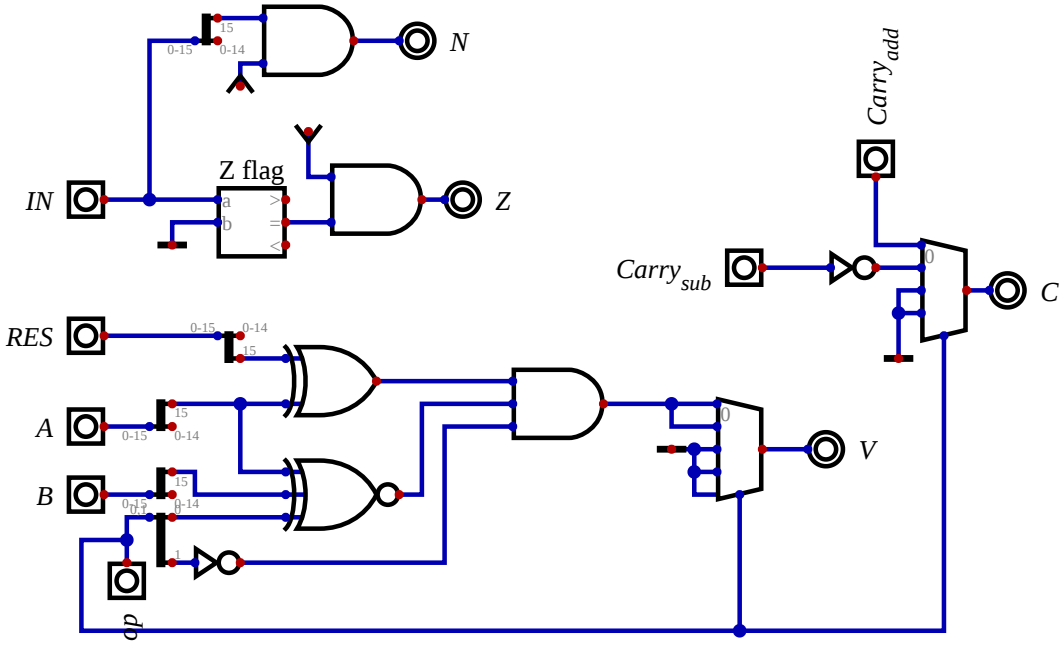
counter.dig.svg



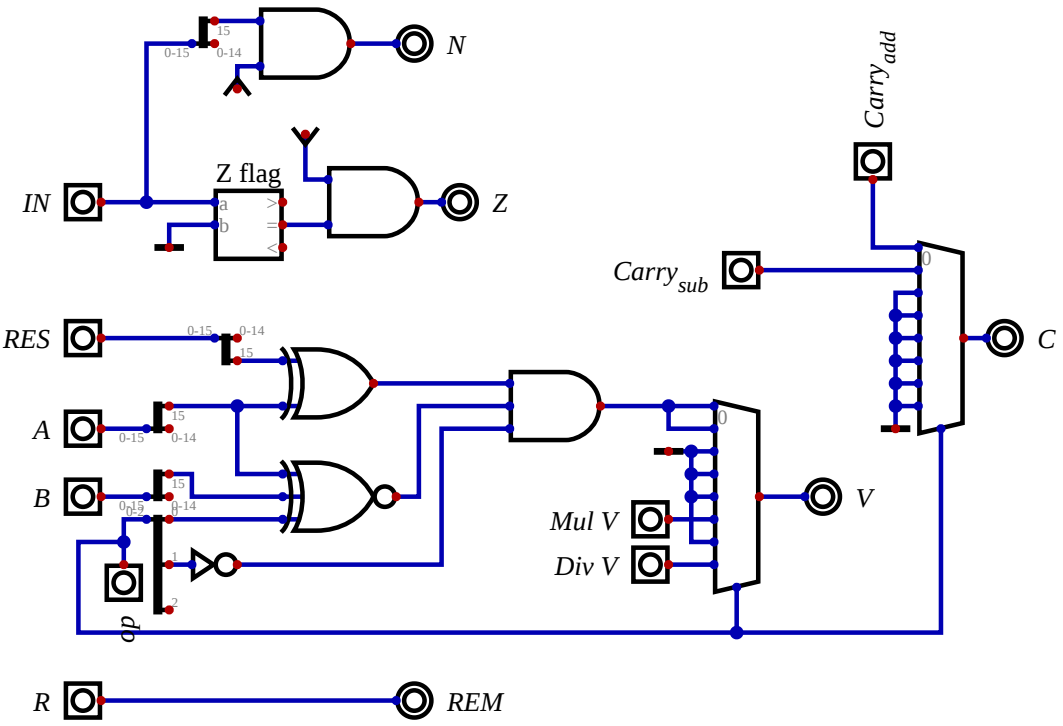
divide_logic.dig.svg



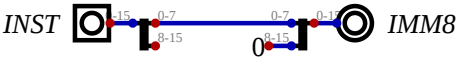
flag_setters_basic.dig.svg



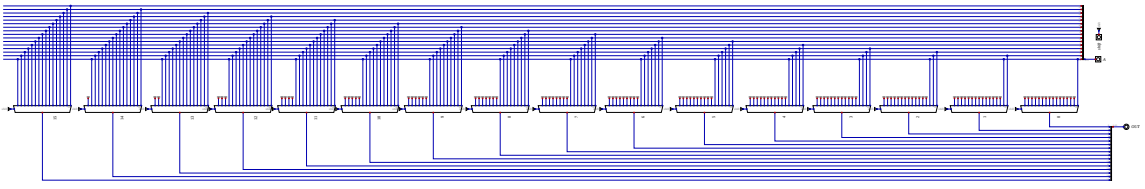
flags_setter.dig.svg



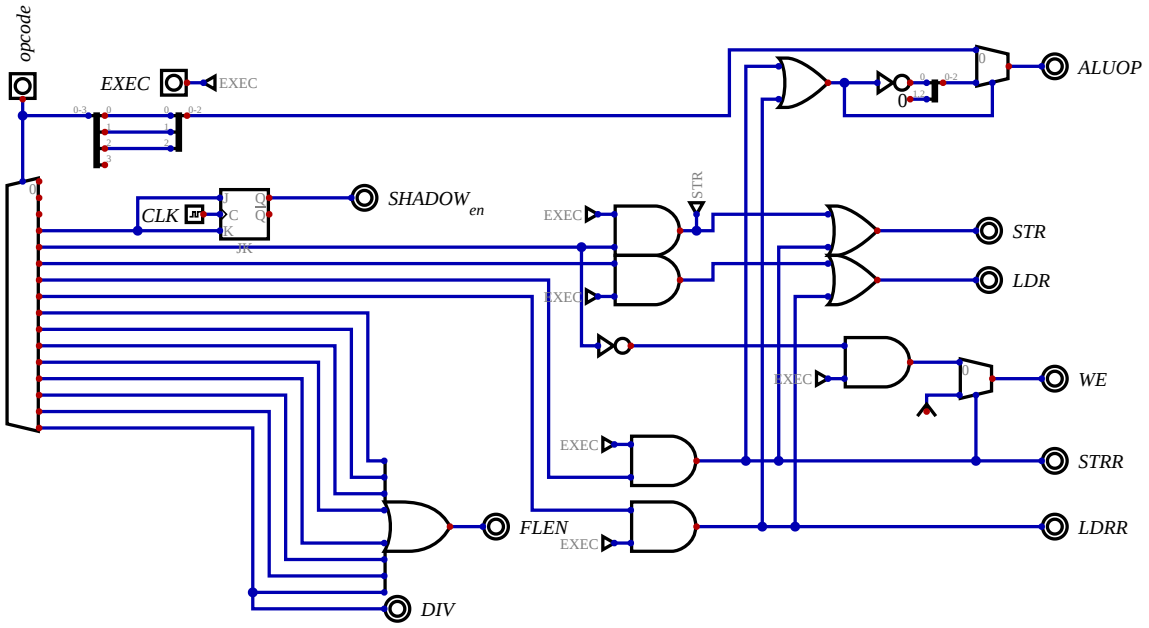
imm8_extender.dig.svg



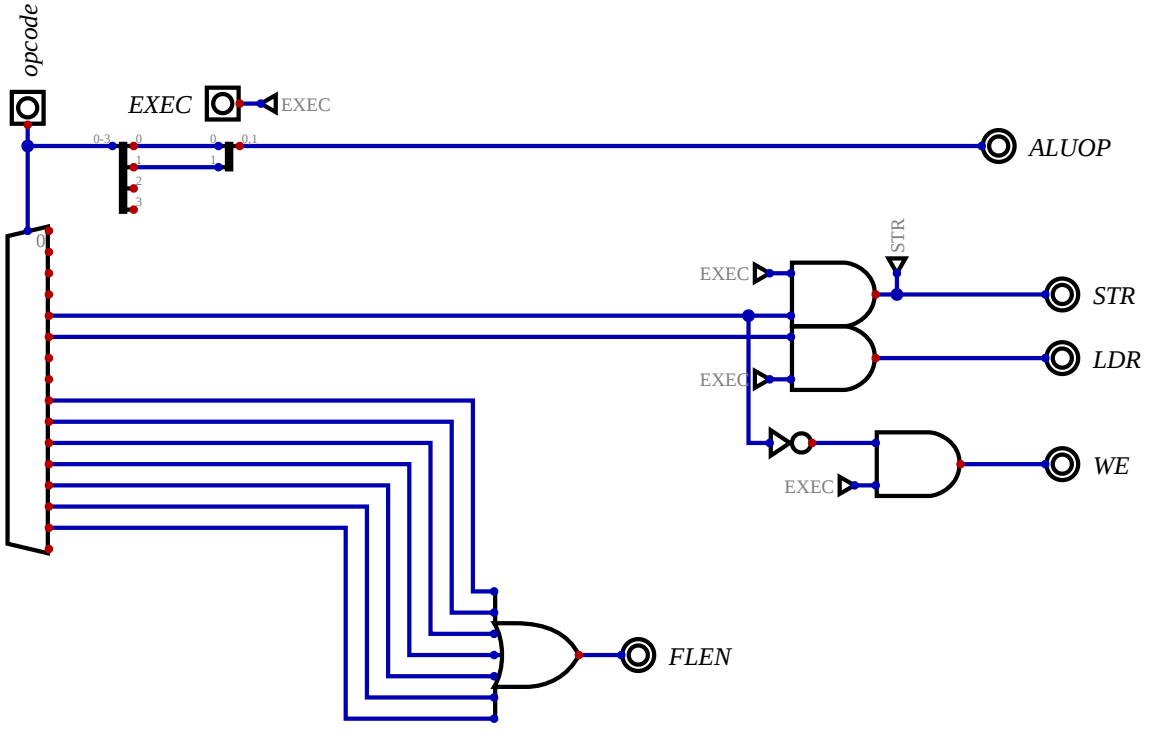
logical_shift.dig.svg



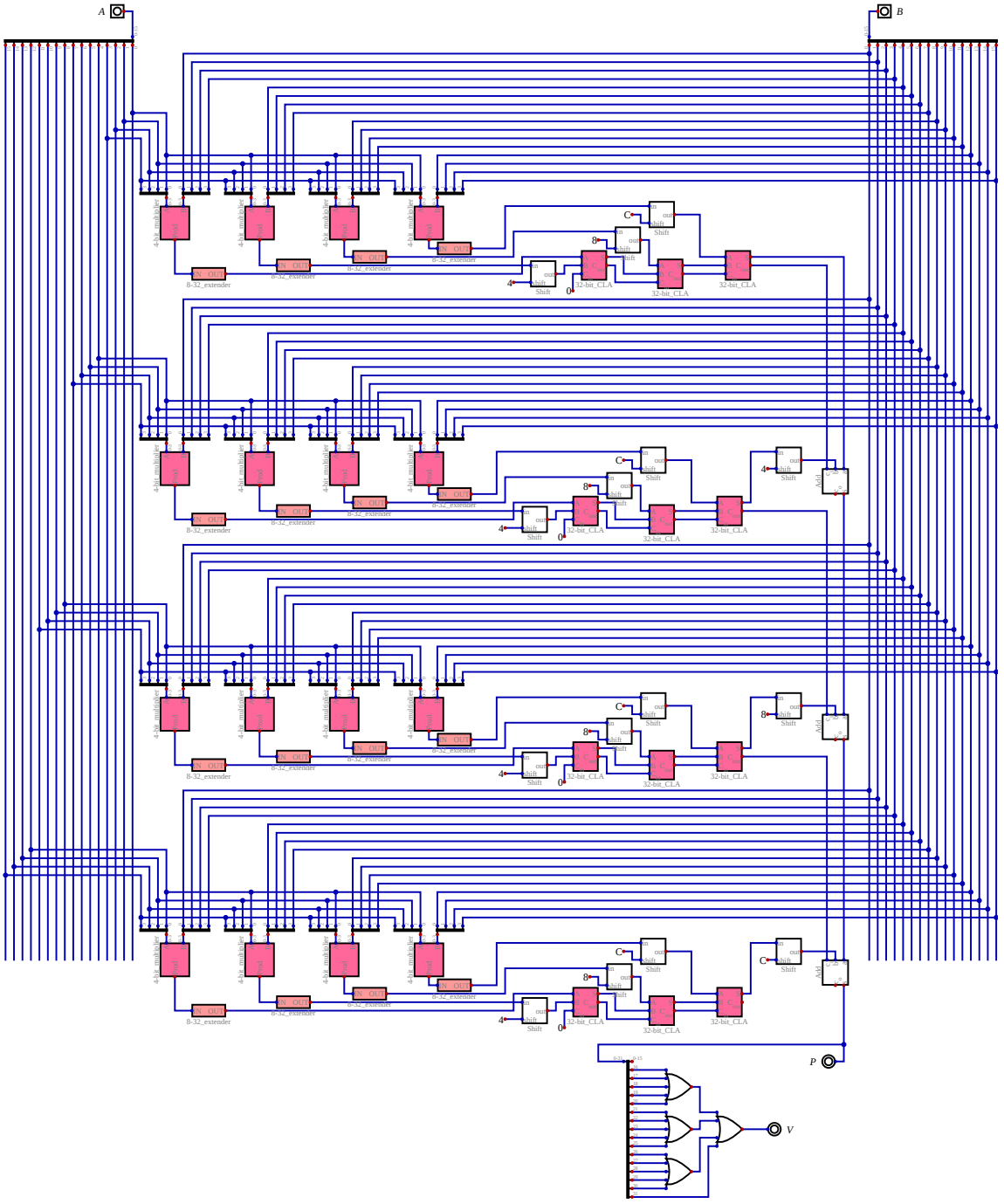
main_decoder.dig.svg



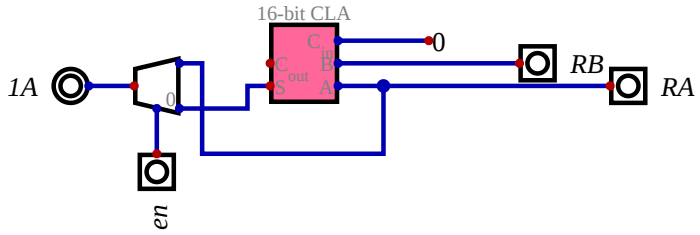
main_decoder_basic.dig.svg



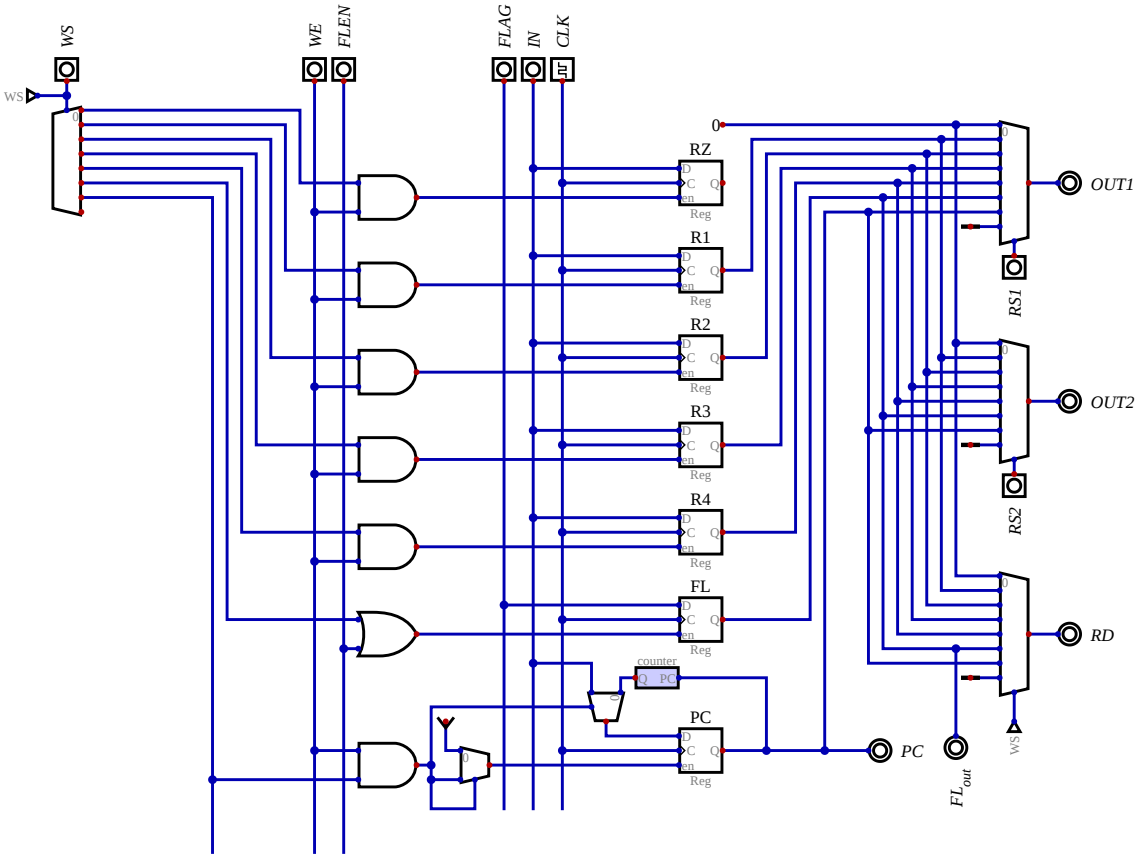
multiply.dig.svg



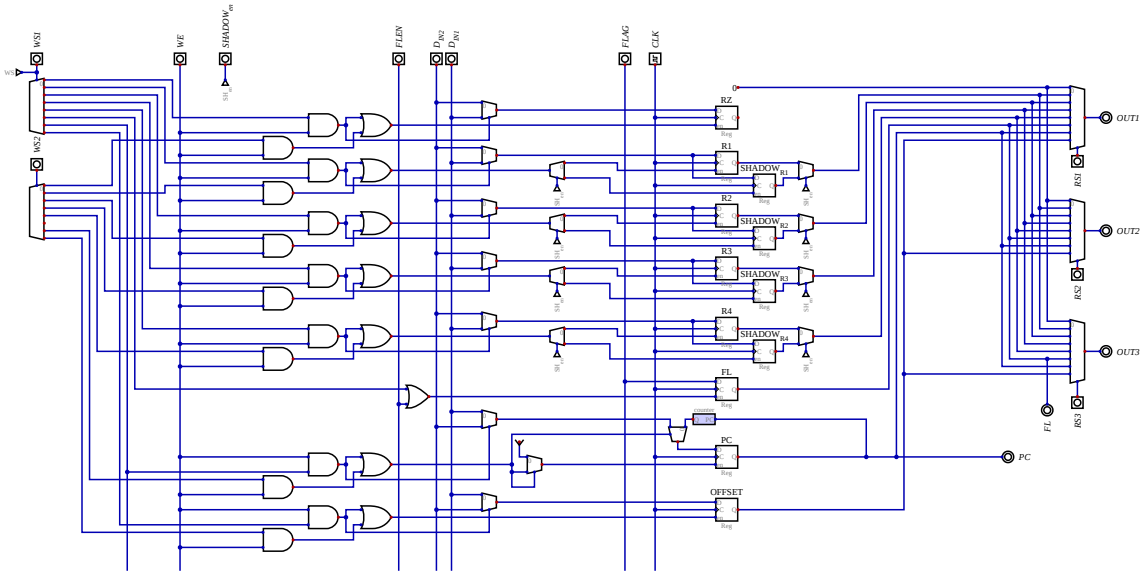
pre_post_addr.dig.svg



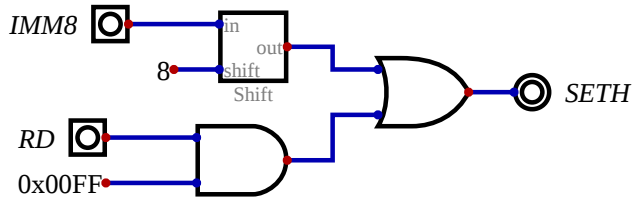
reg_file.dig.svg



reg_file_extn.dig.svg



seth.dig.svg



zero_comparator.dig.svg