CSE 141L Final

Jim Ji A17894775

Academic Integrity

Your work will not be graded unless the signatures of all members of the group are present beneath the honor code.

To uphold academic integrity, students shall:

- Complete and submit academic work that is their own and that is an honest and fair representation of their knowledge and abilities at the time of submission.
- Know and follow the standards of CSE 141L and UCSD.

Please sign (type) your name(s) below the following statement:

I pledge to be fair to my classmates and instructors by completing all of my academic work with integrity. This means that I will respect the standards set by the instructor and institution, be responsible for the consequences of my choices, honestly represent my knowledge and abilities, and be a community member that others can trust to do the right thing even when no one is watching. I will always put learning before grades, and integrity before performance. I pledge to excel with integrity.

Jim Ji

0. Team

Jim Ji

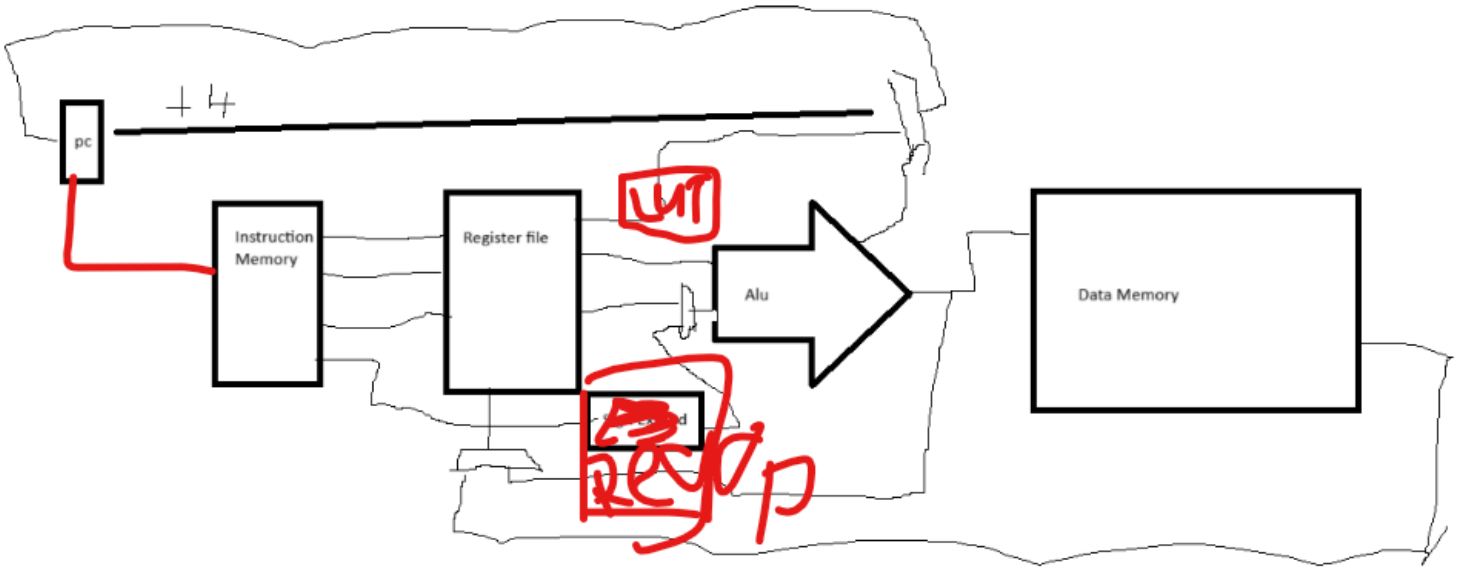
1. Introduction

TODO. Name your architecture. What is your overall philosophy? What specific goals did you strive to achieve? Can you classify your machine in any of the standard ways (e.g., stack machine, accumulator, register-register/load-store, register-memory)? If so, which? If not, devise a name for your class of machine. Word limit: 200 words.

My machine is going to be a load store machine following largely the same instructions as the MIPS architecture, im going to call it the MIC architecture. My philosophy or goal for this architecture is to implement a simplified version of the MIPS architecture with optimizations on the 3 programs I'm going to run, and the speicifc constraints I am given.

2. Architectural Overview

TODO. This must be in picture form. What are the major building blocks you expect your processor to be made up of? You must have data memory in your architecture. (Example of MIPS: https://www.researchgate.net/figure/The-MIPS-architecture_fig1_251924531)



General gist of design, omitted some wires and the control unit.

3. Machine Specification

Instruction formats

TYPE	FORMAT	CORRESPONDING INSTRUCTIONS
R	3 bits opcode, 3 bit dest operand, 3 bit supporting operand	Sub, flip, or, ldr , str
B	3 bits opcode, 1 bit operand register, 1 bit operand register, 4 bit address	beq, bne, etc.
I	3 bits opcode, 3 bits opearnd dest register, 3 bit immediate	mv
RegOp	0bits opcode (fixed 111), 3bit dest operand , 3 bit regop code	Neg1, neg0, sr, sl, carry, inc, chkNeg

Operations

An example row has been filled for you. When you submit, do not include the example type. In the name column, be sure to also add the definition of what the example actually does. For example, "lsl = logical shift left" would be an appropriate value to put in the name column. In the bit breakdown column, add in parenthesis what specific values the bits should be in order. X indicates that it will be specified by the programmer's instruction itself (i.e. specifying registers). In the example column, give an example of an “assembly language” instruction in your machine, then translate it into machine code. Add rows as necessary. In your submission, please delete this paragraph.

NAME	TYPE	BIT BREAKDOWN	EXAMPLE	NOTES
and = logical and	R	1 bit type (0) bits opcode (010), 1 bit funct (1), 1 bit operand register (X), 1 bit operand register (X), 2 bit destination register (XX)	# Assume R0 has `0b0001_0001` # Assume R1 has `0b1001_0000` `and R0, R1, R2` ⇔ `0_010_1_0_1_10` # after and instruction, R2 now holds `0b0001_0000`	This is a completely bogus example, since this implies that there are only 2 possible operand registers and 4 possible destination registers. Mention special things like implied destination register (i.e. stack) or special notes here.
Or	R	Op code (000), 3 bit destination register (XXX), 3bit other register (XXX)	Or r0 r1 R0: 00000000 R1: 10000000 R0: 10000000	Reg Or

Flip bit (flip)	R	Op code (001), 3 bit dest register, 3 bit pos reg	#Assume R1 has 0b1000_0001
			# Assume R0 has ob0000_0000
			R3 = 10
			Flip R0 r3
			Flip R1 r3
			R0 = 0000 0000
			R1 = 1000 0010
				Supports pos 1 - 16 (1 index pos)
				
				Will treat second (must be immediately) op call as the register containing the MSB

Branch	B	Op code(011), 1 bit MSB	Sub r1 r1	Must be used in conjunction with sub, sub

(b)		(branch addr) , 2 bit branch mode (XX), 3 bit branch pos	Sub r1 r1 B 000 000 //branch to lut idx 000	is used to generate the flag values for branching to succeed, Modes: 0 is if 2 consecutive sub instruction evaluates to 0 (so a implicit AND op) 1 is if prev rslt is neg 10 is if prev rslt is pos Branch codes translate to abs address from pc_Lut, msb for this is the 4th bit from the right so mach code 011 100 000 //means branch to lut idx 8 if cmp succeeds
sub	R	Op code (010), 3 bit dest, 3 bit other op, rslt = dest - otherOp	R1 has 16 R0 has 10 Sub r1 r0 R1 has 6	Sepcial thing is that this thing is the cmp of the branch opeartion, and it also checks for carry overflow which routes to a regop instruction
mv	I	Op code (100), 3 bit dest, 3 bit immed	R1 has X //can be anything Mv r1 3 R1 has 3	Clears entire register and asigns it the immediate value, very useful
ldr	R	Op code (101), 3 bit dest, 3 bit mem address	R1 has x //anything R2 has 3 Ldr r1 r2 R1 has mem[3]	Loads value to reg, not really special, but this style also limits mem access to a max 2^8 addresses.
str	R	Op code (110), 3 bit source, 3 bit dest address	R1 = 20 R2 = 3 Str r1 r2 Mem[3] = 20	The opearands a little flip in this one, but still it is just typical store
regop	regop	Op code (111), 3 bit	The operations supported are	Besides shift left and right, the rest are

		destination 3 bit regop code	000: check neg and 2's complement for MSB 001: check neg and 2's complement for LSB 011: increment plus add extra 1 if 001 op resulted in carry 010: regular shift right with carry memorized for next instruct 100: if prev is negative then flip msb 101: regular shift right without carry (did not use) 110: regular shift left with carry 111: if prev sub instruction carried over, then add 1 to cur value	more utility functionalities to help me implement the programs easier, the functionality are mostly oddly specific, treat as kind of bandaid to my current processor.

Internal Operands

8 general purpose 8 bit registers, has all normal behaviors

Control Flow (branches)

Equal to 0, smaller than 0, greater than 0 branches are supported through signals from alu saved to flipflops in toplevel. Branch modes accessed through middle 3 bits.

Addressing Modes

TODO. What memory addressing modes are supported, e.g. direct, indirect? How are addresses calculated? Give examples.

Address is all absolute, since there is relatively little branching involved (less than 10 for each of the programs) We just address from top to bottom 1-xxxx, makes it easy to reason about branches and easy to update the PC_LUT.

Memory address is also absolute, since the only times we should be accessing mem is when grabbing opearands and placing the answer, there is relatively little use in something more complicated.

4. Programmer's Model [Lite]

TODO. 4.1 How should a programmer think about how your machine operates? Provide a description of the general strategy a programmer should use to write programs with your machine. For example, one could say that the programmer should prioritize loading in the necessary values from memory into as many registers as possible, then perform calculations. Another approach could be loading and writing to memory in between every calculation step. Word limit: 200 words.

The programmer should think about registers 2 at a time as most instructions would be implied to be doing operations on 2 registers. And they should seek to load as much stuff from memory as possible, as doing actual operations would be quite cumbersome as those instructions do not support immediates.

TODO. 4.2 Can we copy the instructions/operation from MIPS or ARM ISA? If no, explain why not? How did you overcome this or how do you deal with this in your current design? Word limit: 100 words.

You cannot copy instructions from MIPS or ARM as we are dealing with the limit of only having 8 bit registers, and the 9 bit instruction length, a lot of things are implied with the most important being the assumption of we are looking at 2 registers when doing operations.

4.3 "Will your ALU be used for non-arithmetic instructions (e.g., MIPS or ARM-like memory

address pointer calculations, PC relative branch computations, etc.)? If so, how does that complicate your design?"

No, branches addresses are stored in PC_LUT, all operations will be purely arithmetic.

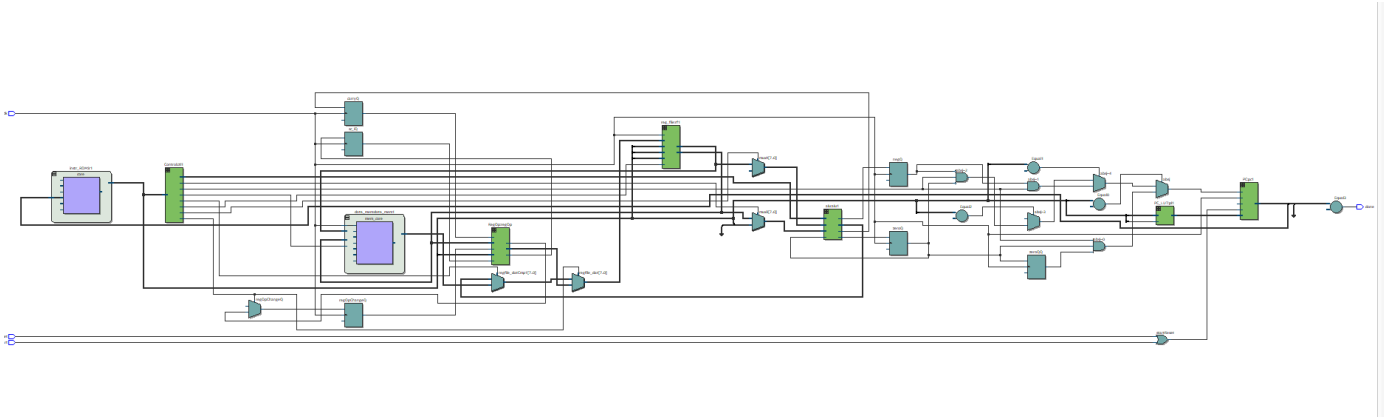
Individual Component Specification

TopLevel:

Description:

Connection of all modules, stores previous signals for modules to use (zeroQ, negQ, etc). Also raises done flag (specifying pc counter)

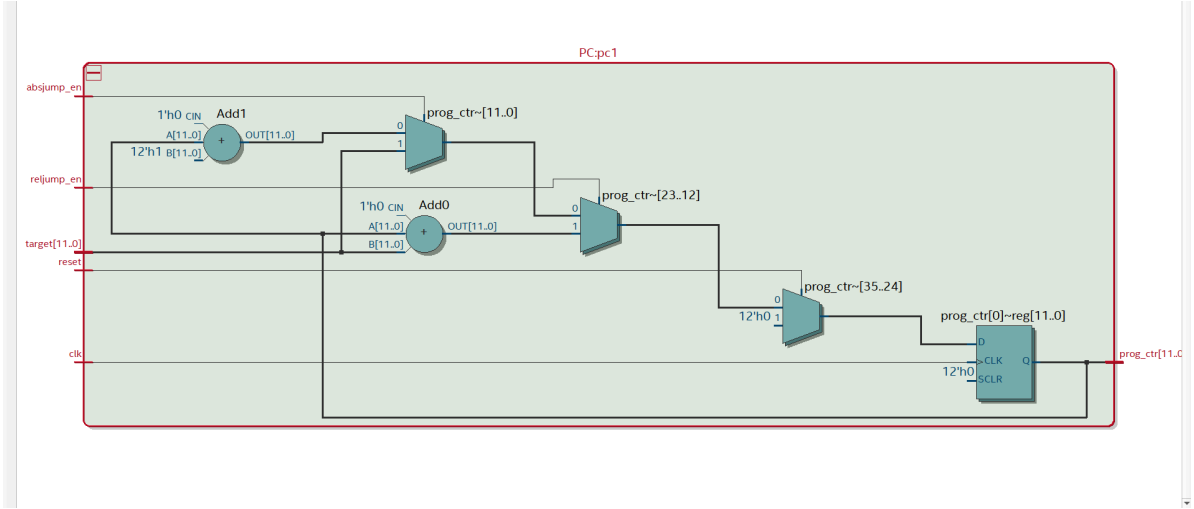
Schematic:



PC:

Advances the program counter and handles jump signals

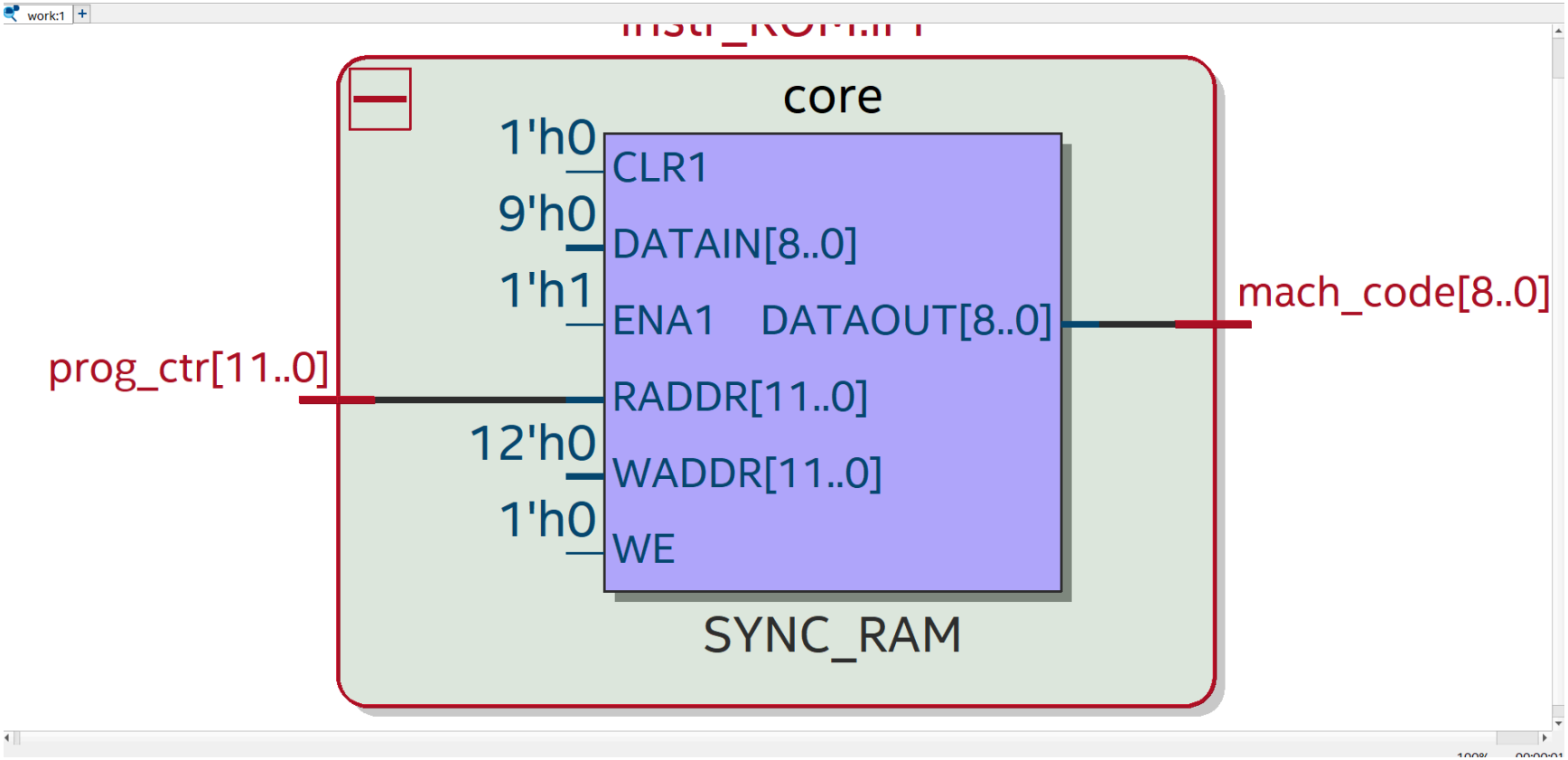
Schematic



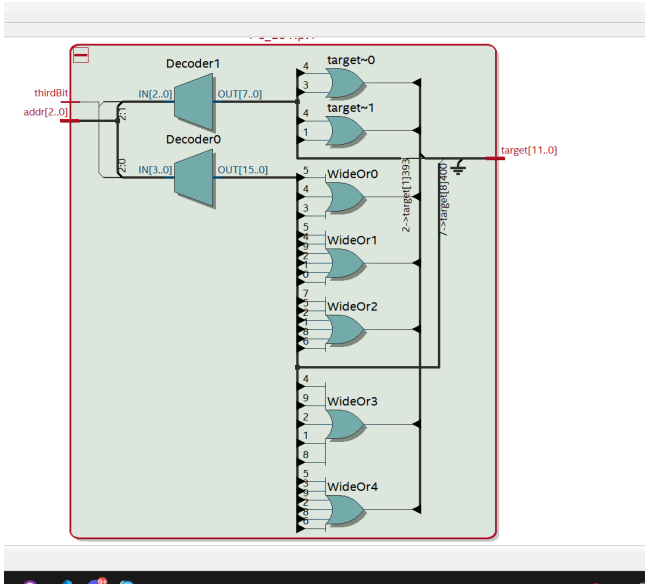
INSTR_ROM:

Reads machine code from file, and outputs machcode based on address from PC

Schematic:



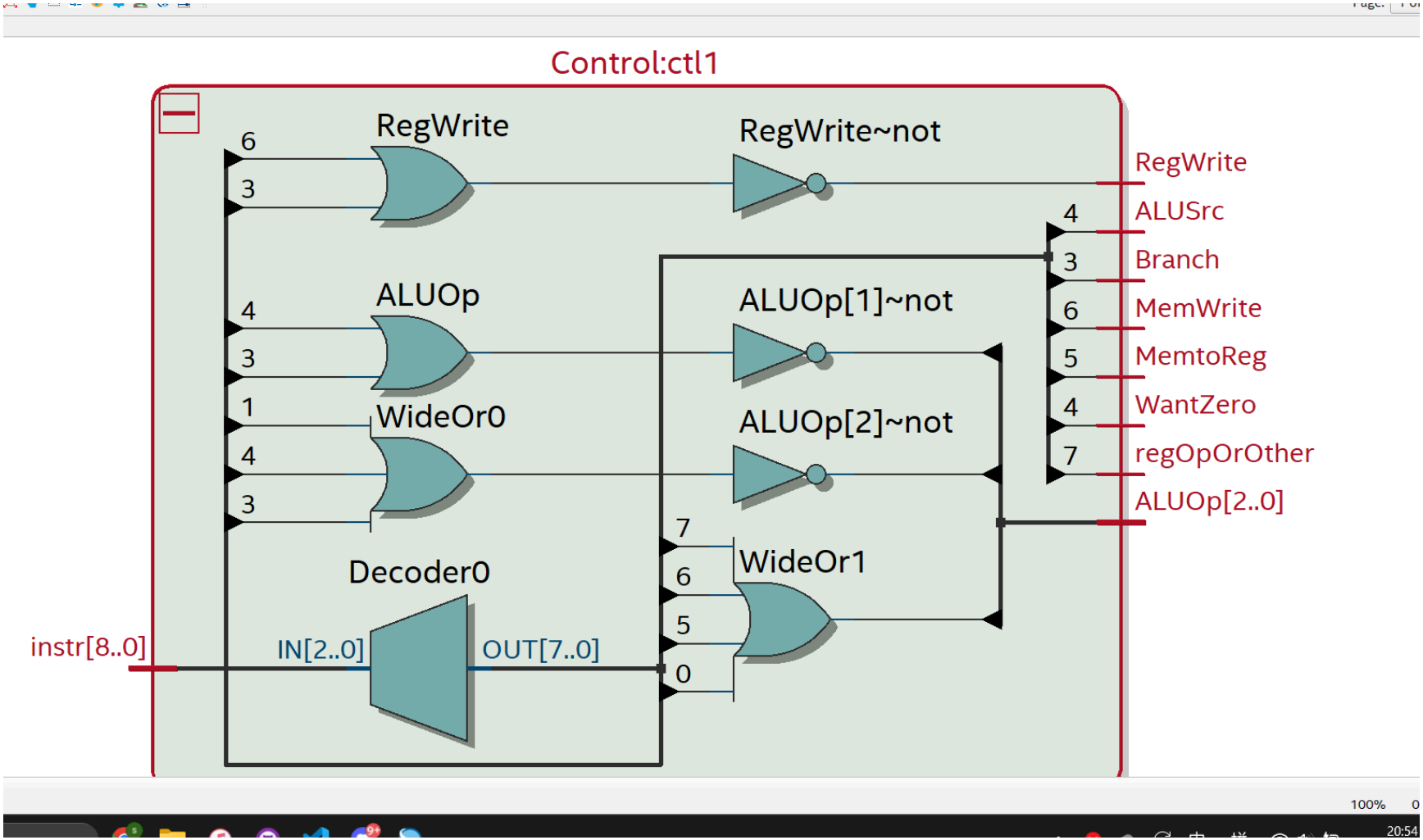
PC_LUT: stores all absolute branch locations (only have abs branches)

Schematic 

Control:

Processes mach code and gives signals to modules for read write enable, modes, etc. Also does a little of what control does by deciding when to branch.

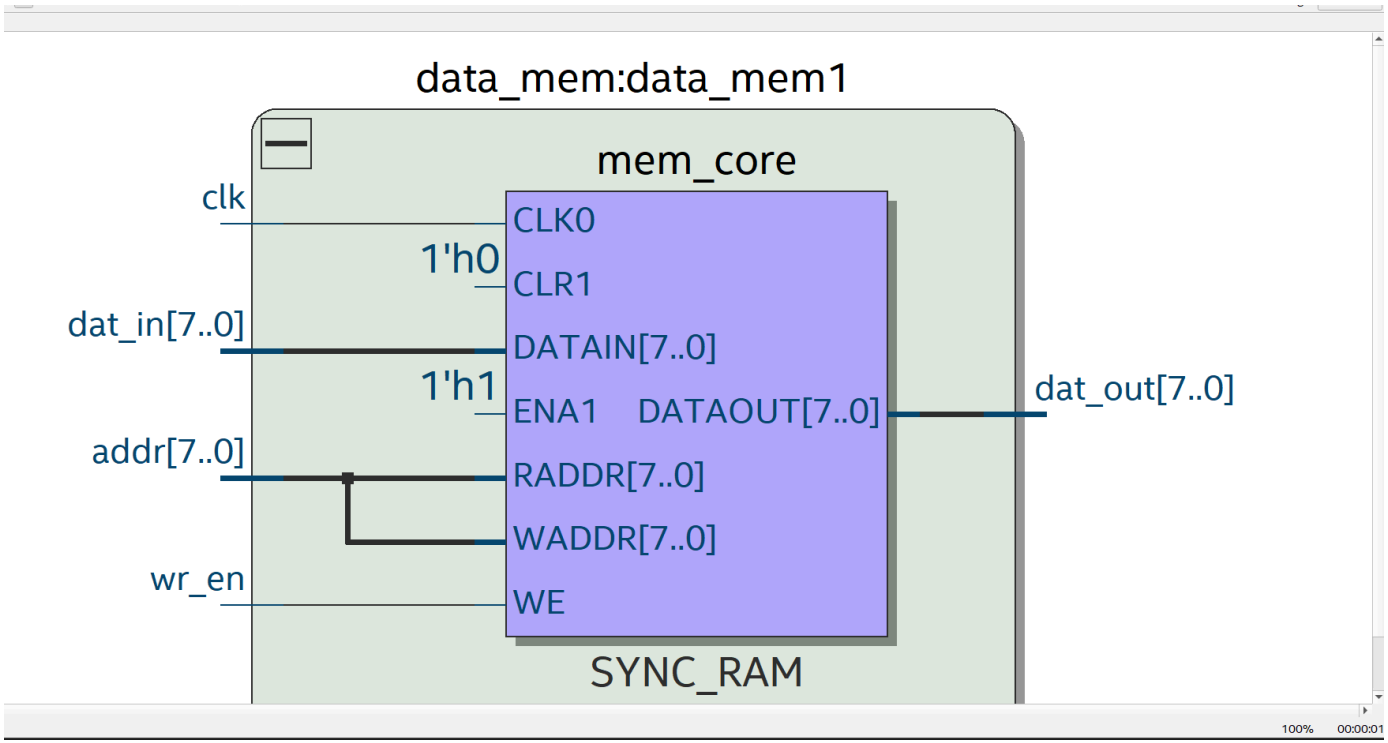
Schematic:



Dat_mem:

Stores the operands

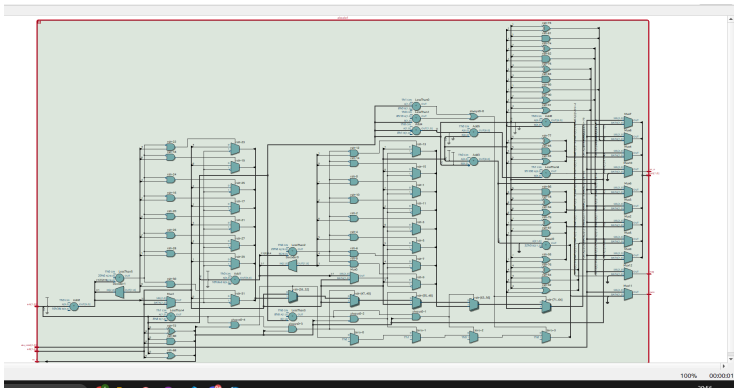
Schematic:



ALu:

Does operations based on control signals

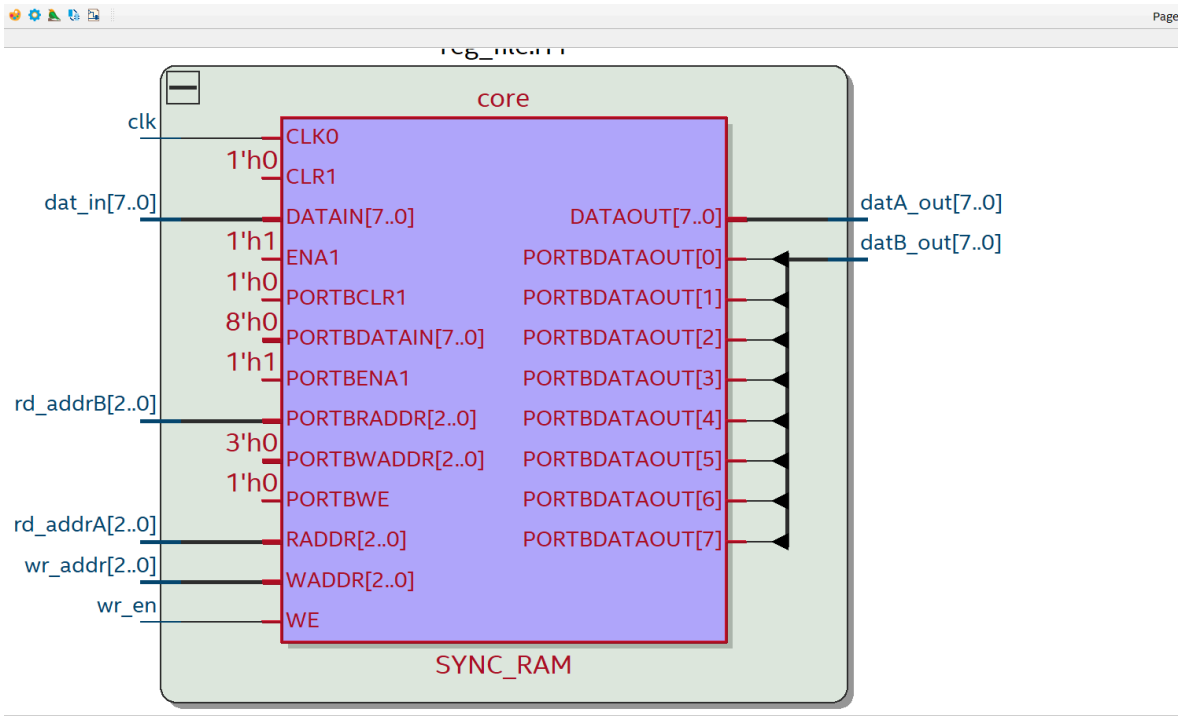
Schematic:



Reg_file:

Stores 8 usable registers, has 1 write and 2 reads.

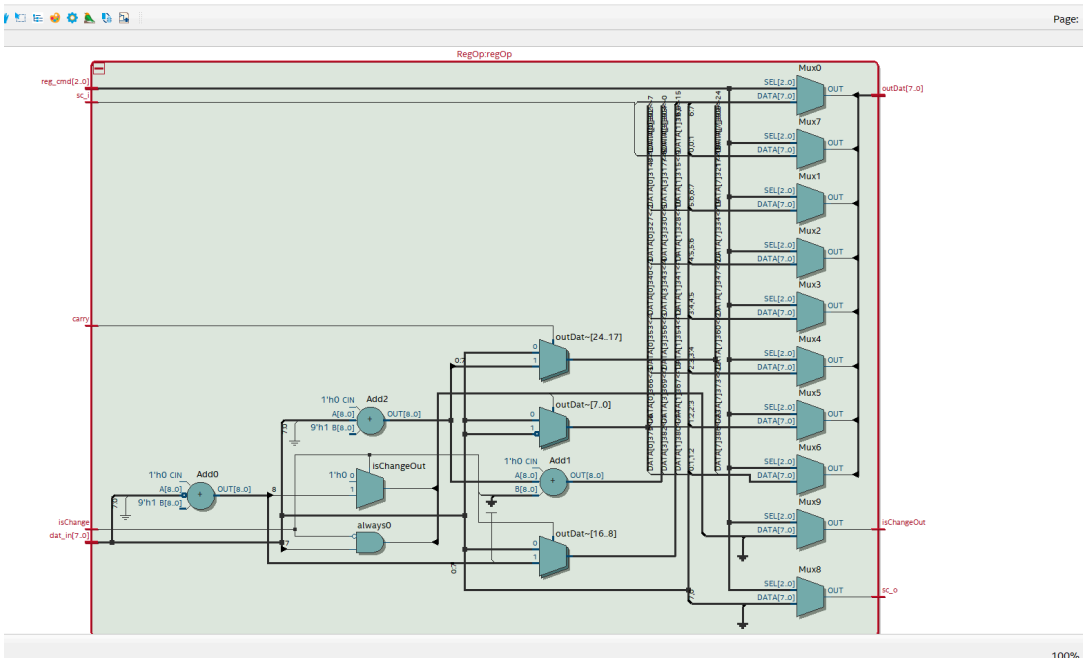
Schematic:



Reg_Op:

Does operation on 1 register only, has leftshift, rightshift, etc.

Schematic:


Page:
RegOp.regOp
reg_cmd[2..0]
sc_i
carry
isChange
dat_in[7..0]
Add0
Add1
Add2
isChangeOut
always0
outDat~[24..17]
outDat~[7..0]
outDat~[16..8]
Mux0
Mux1
Mux2
Mux3
Mux4
Mux5
Mux6
Mux7
Mux8
Mux9
SEL[2..0]
DATA[7..0]
OUT
outDat[7..0]
isChangeOut
sc_o
100%