Name: Xinyi Alexis Wu

Note: I started working on this since sunday and this one has been taking a majority of my time.

By ddl I only accomplished the whole alu (see github history at

https://github.com/AlexisWu-01/olin-cafe-f22/tree/lab2/hws/6)

I wished to complete it instead of submitting what I have done by ddl. (Since this one builds up to wiring cpu)

Homework 6

The git repository hws/6 directory is the basis for this homework. Space for answers intentionally left out - include these in your single pdf in the submission zip file.

Reading

- Reread:
  - H&H 3.5: Timing, Metastability, except 3.5.6
    - Definitely read sections 3 and 4 of this guide (the rest is great too, read it if you have time!) Synchronization and Metastability, Steve Golson, Synopsys Users Group Conference 2014
  - H&H 5.2.5: Shifters and Rotators
  - H&H 5.5: Memory Arrays
- New:
  - H&H 6.1 to 6.4
  - H&H 7.1-7.2

0. Spiral 2 Feedback

Please fill out this form to give me feedback on how the last spiral through the course went.

1. Metastability and Designing for Failure

(a) Reasonable MTBFs (Mean Times Between/Before Failure)

For each of the following systems, pick an MTBF (specify your time unit!) that you think makes sense as a design requirement for the entire system. Note that for systems with constant probability of failure rates ~66% of units will have failed at least once before the MTBF has elapsed, and that in the case of a metastability failure a full system reset (power cycling) is required. Write a sentence explaining why you chose your number for the application.

Toddler Toy Piano (ages 2 to 4):

Toddlers' concentration time would be no more than 12 minutes (according to google). Therefore an MTBF of 15 minutes is good enough. (The users might not even notice if it really fails.)

Industrial Robot Arm:
An industrial robot arm should not make too many mistakes, but we could restart it every day/ week based on the type the work robot. In that case it could be from 24 hours to 7 days

Vehicle ADAS (Automated Driver Assistance System):
This cannot go wrong. However, it is recommended that drivers takes a break every 4 hours. So it should work more than that in case of malfunction and sleepy driver happen at the same time. (The car would be restarted at least once everyday since even super drivers would need to take breaks for fuel/ charge). If the normal reset counts for the failure reset, an MTBF of 24 hours could be enough. If that does not count, then we want this to be as long as possible, as a consumer, once a year is acceptable.

(b) Case Study: Toy Piano

Our toy piano has 24 different digital asynchronous inputs (keys) that an eager toddler can hit very quickly. Using Equations (24) and (25) from the Synchronization and Metastability guide (page 22), and the following device parameters, determine the fastest clock speed $\boldsymbol{f_c}$ that lets the **full system** meet the MTBF you found in part (a). You can assume the following device parameters:

- $\boldsymbol{\tau}$= 175 *ps*
- $\boldsymbol{T_o}$ = 225 *ps*
- $\boldsymbol{f_d}$ = 10 *Hz* (these are very eager toddlers)

- $t_R = t_{setup} - Tc$, *where* $T_c = 1/f_d$ *and* t_{setup} = 200 *ps*

This problem isn't supposed to be about annoying algebra, so there is a python script in the homework folder that shows you how to sweep some frequencies on a log scale to see what works (there's no analytical solution for this, so you have to guess and check).

Setting the f_c to 3.5e8 Hz would meet the required MTBF of 15 mins. (See modified py file)

2. Combinational Review: A complete 32-bit ALU

The "thinky" element in our custom RISC-V CPU is the component that can do just about any math we need for reasonable computation. This is called an Arithmetic Logic Unit, or ALU. You should have all the building blocks for this module.

Use only structural combinational logic (no flops, no ifs/elses, etc.). That means use `always_comb` statements with only ~&|^? operators for these modules. Working adders and muxes are included in the stub folder. There are many opportunities to be clever with submodule re-use, but remember that it is better to have a working large or slow ALU than a very fast but incomplete one. The only new features that shouldn't be in prior examples/solutions are:

- Implement an SLL (shift left logical) operation that shifts input a to the left by b bits. The result should be padded with zeros.
- Implement an SRL (shift right logical) operation that shifts input a to the right by b bits. The result should be padded with zeros.
- Implement an SRA (shift right logical) operation that shifts input a to the right by b bits. This result should be "sign extended" - that means that you need to copy the most significant bit.
- **STRETCH** - add correct overflow logic so that the ALU reports if an operation resulted in a 32 bit overflow. Only do this if you've finished the rest of the assignment.

You should augment the test_alu.sv example with more test cases to give you confidence that you have implemented the different operations correctly. Include descriptions **and** schematics in the top level PDF that show your approach to the shifters.

Confidence/Skills Check

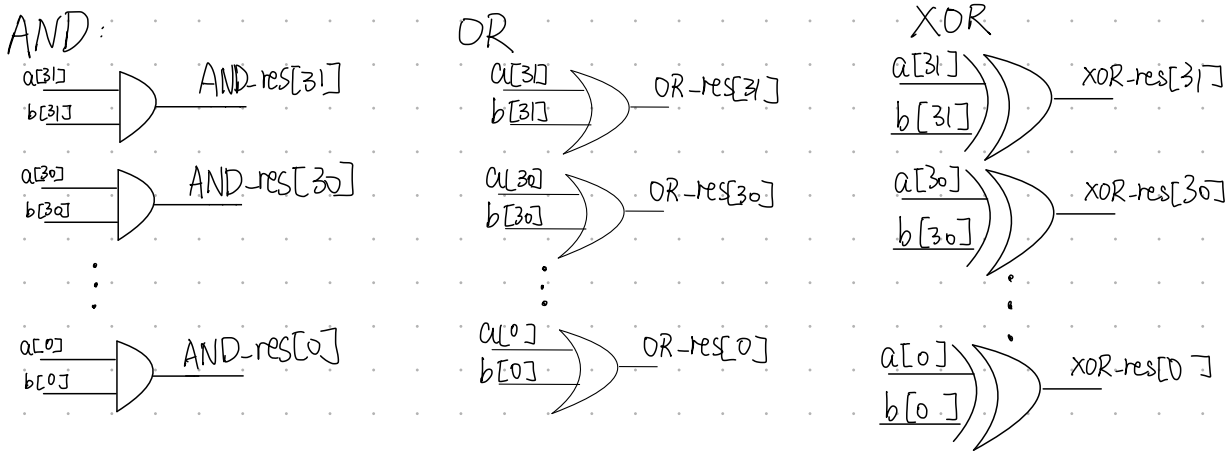
With the hints from the reading this should be starting to feel straightforward in theory but a little tricky in execution (there are a lot of wires to deal with in 32 bit systems). See the solution muxes for examples of using python to auto generate long connection lists.

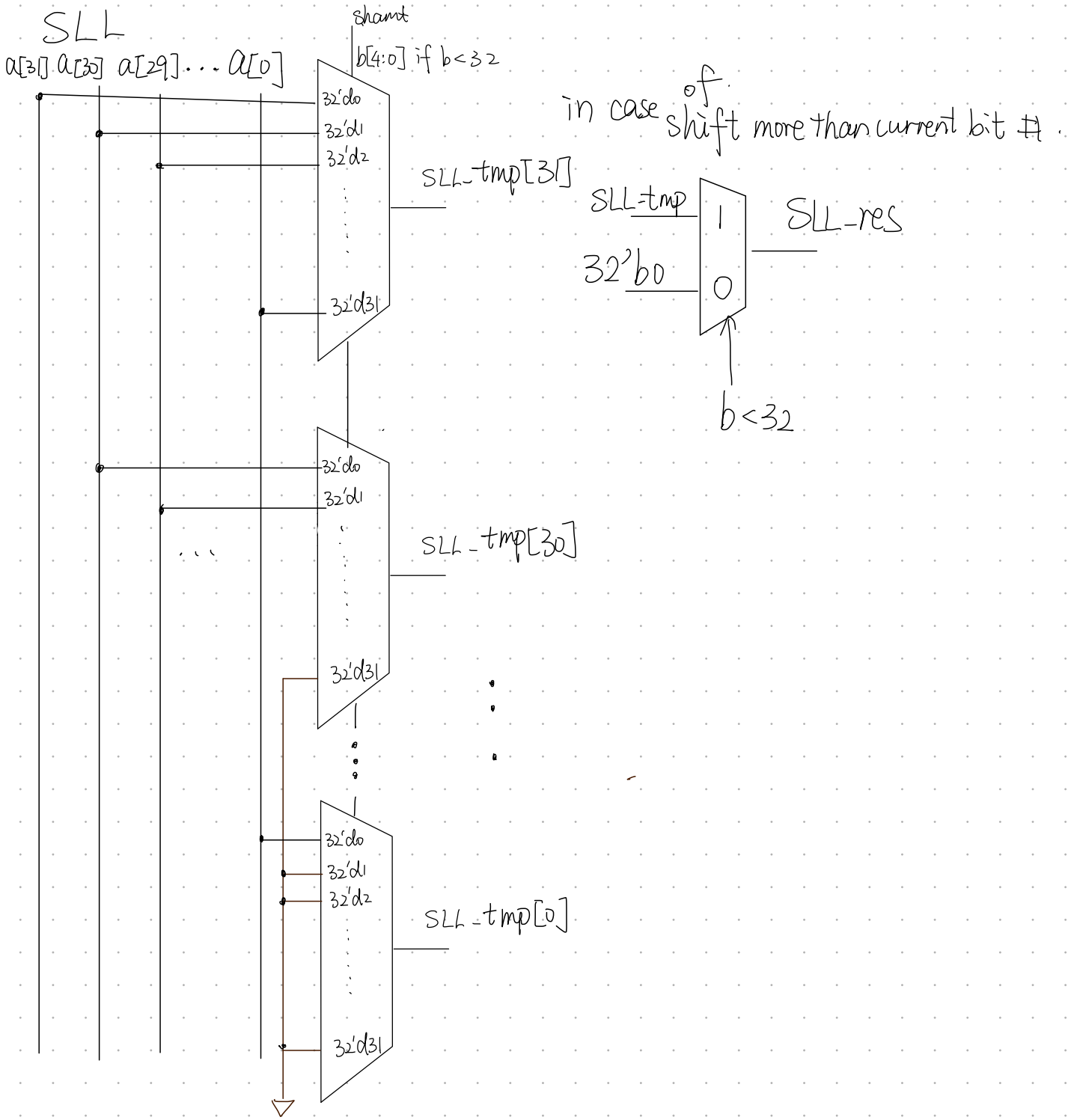
3. Sequential Review - Register File

Implement a 32-bit register file using **structural synchronous** and **combinational** logic (i.e. only `always_ff`, `always_comb` with basic ops ~&|^? allowed). A working register.sv file is provided, but you will need to add the other modules together to create this file. Hint, there are some combinational modules not included in the folder that you might need for this one.

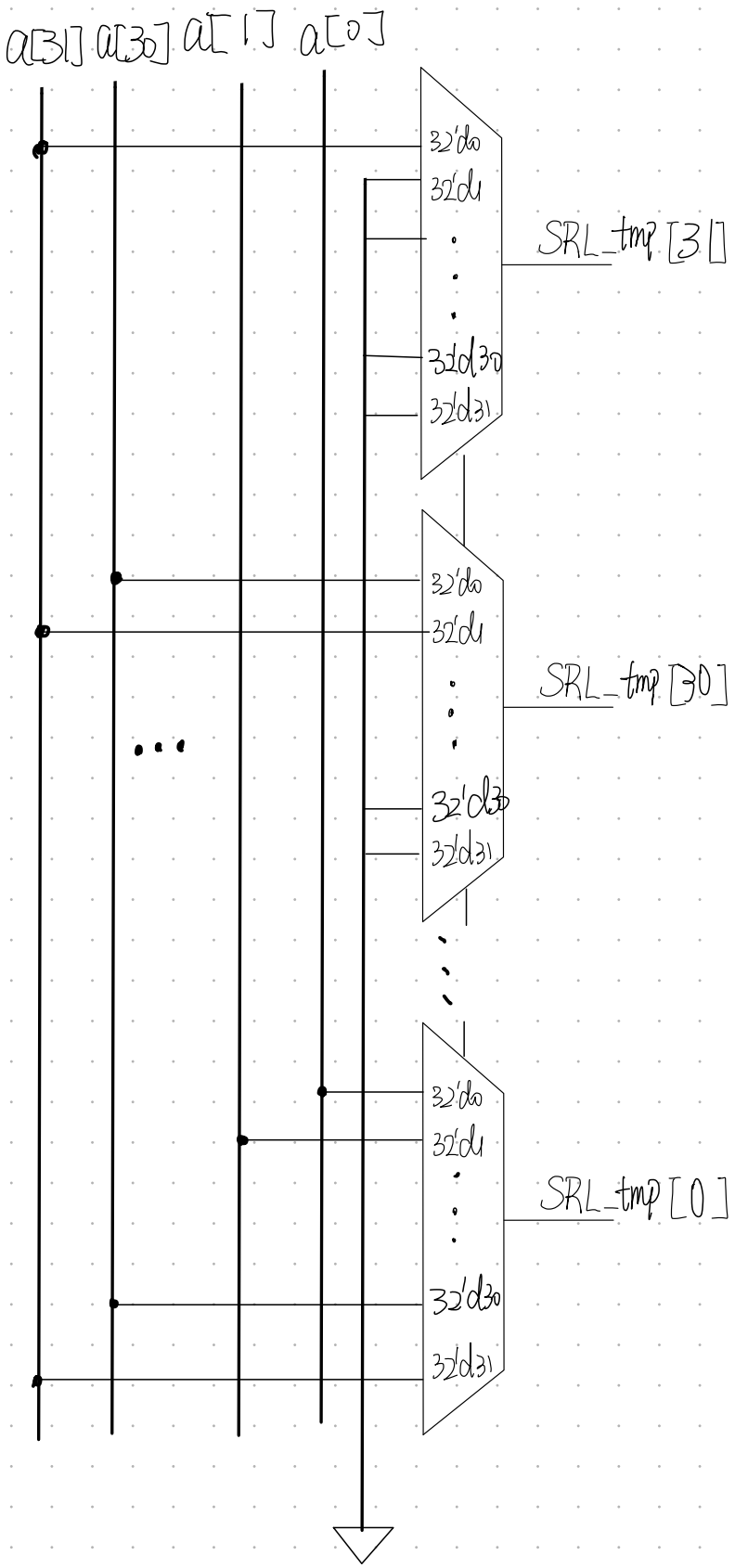
Confidence/Skills Check

This is more about understanding how a register file is supposed to work and getting better at wiring large systems together - the underlying circuitry is simple once you understand how a register file is supposed to work.

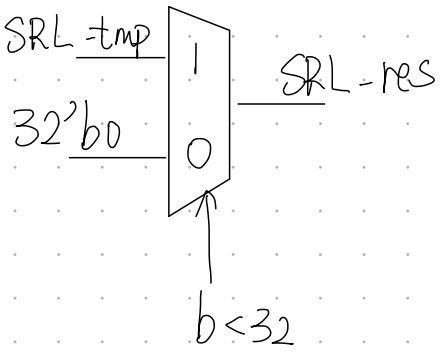

AND
a[31]
b[31]
AND_res[31]
a[30]
b[30]
AND_res[30]
a[0]
b[0]
AND_res[0]
OR
a[31]
b[31]
OR_res[31]
a[30]
b[30]
OR_res[30]
a[0]
b[0]
OR_res[0]
XOR
a[31]
b[31]
XOR_res[31]
a[30]
b[30]
XOR_res[30]
a[0]
b[0]
XOR_res[0]


SLL
a[31] a[30] a[29] ... a[0]
shamt
b[4:0] if b<32
32'd0
32'd1
32'd2
32'd31
SLL_tmp[31]
SLL_tmp[30]
SLL_tmp[0]
in case of shift more than current bit #.
SLL_tmp
32'b0
1
0
SLL_res
b<32

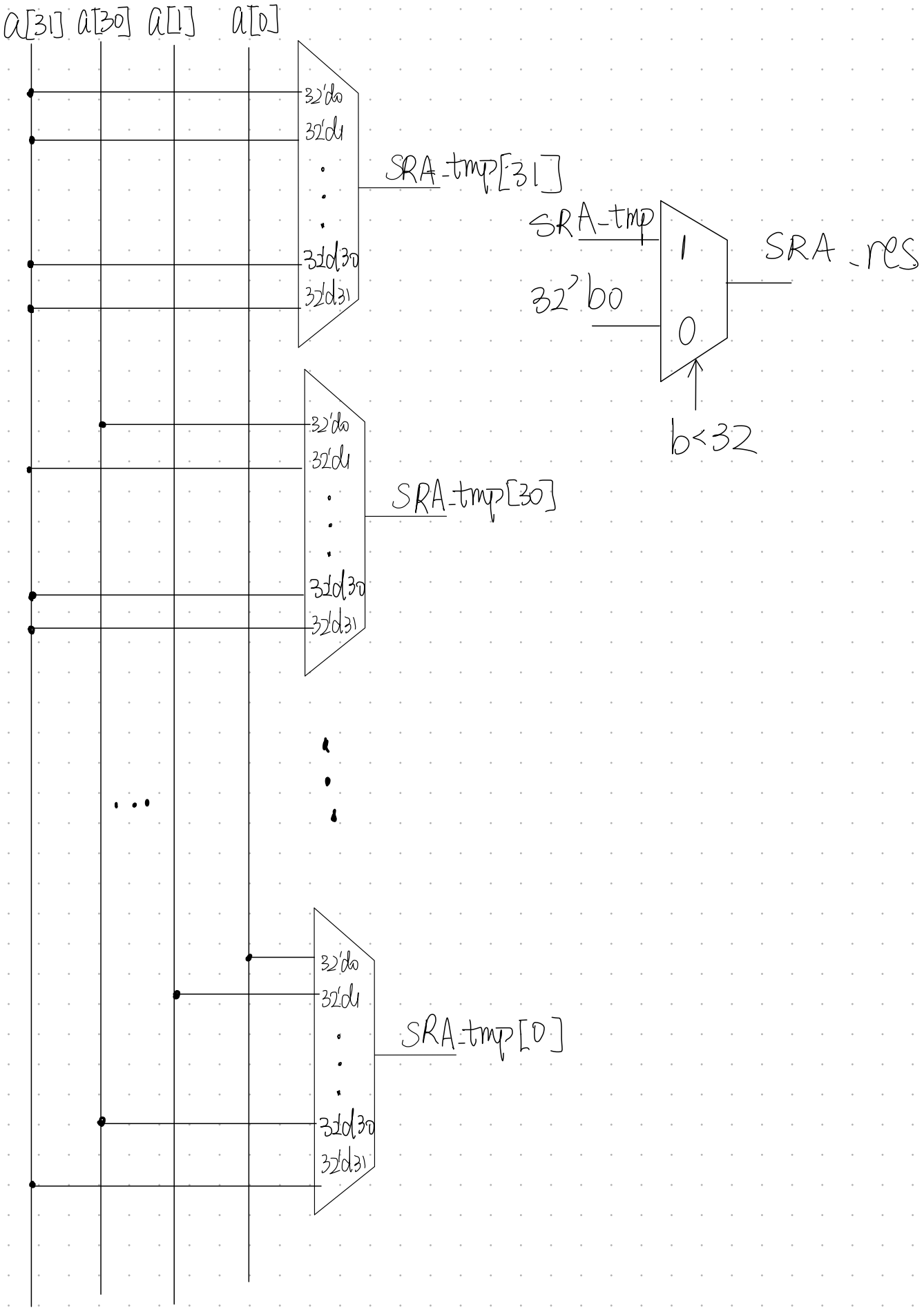
SRL



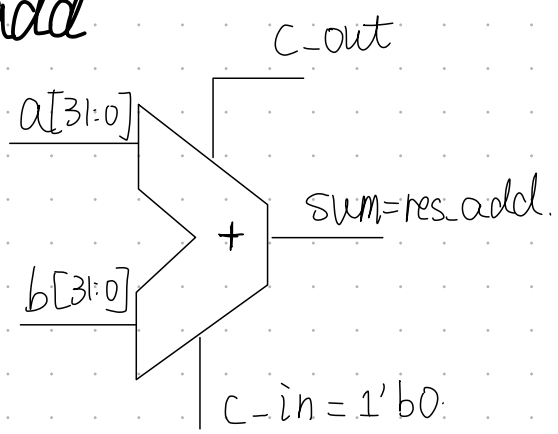
in case of shift more than current bit #.



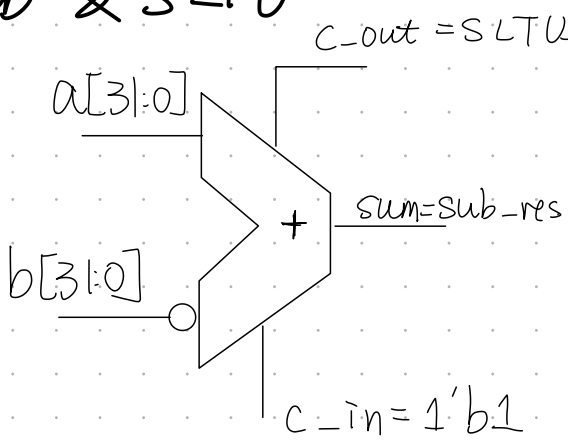
SRA


a[31] a[30] a[1] a[0]
32'd0
32'd1
32'd30
32'd31
SRA_tmp[31]
SRA_tmp
32'b0
1
0
SRA_res
b<32
SRA_tmp[30]
SRA_tmp[0]

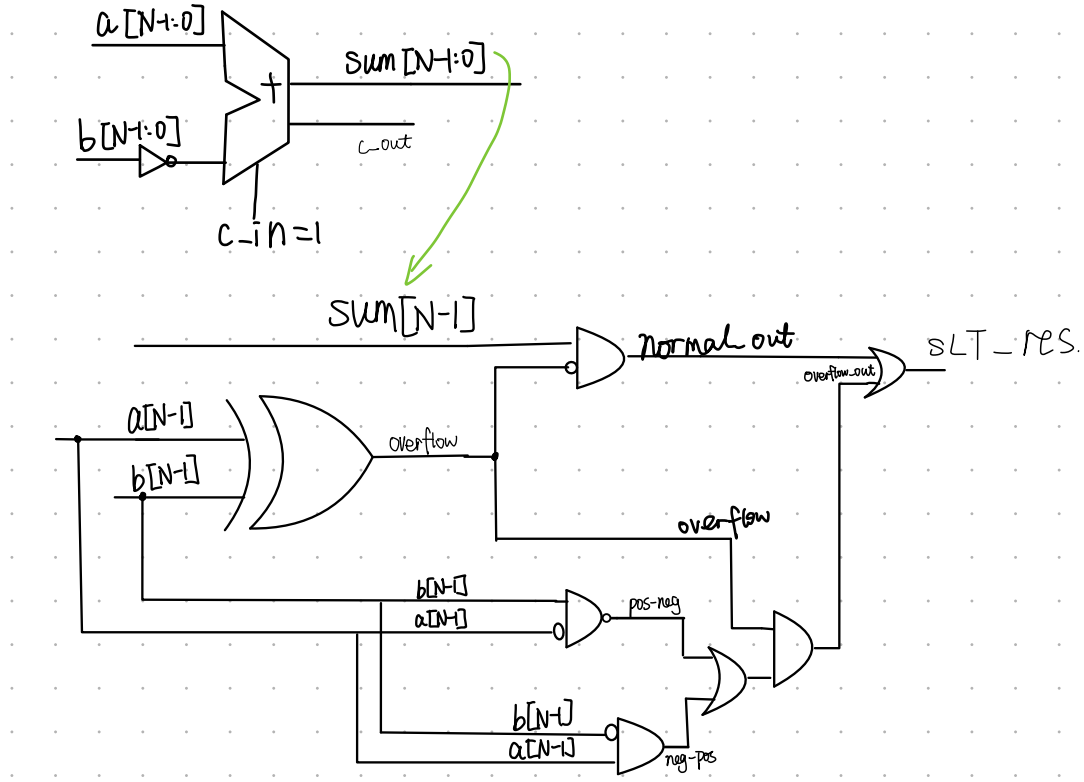
add



sub & SLTU

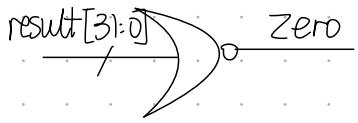


SLT

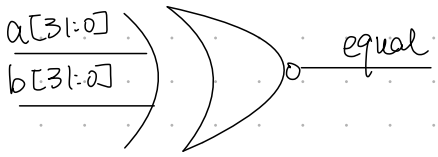


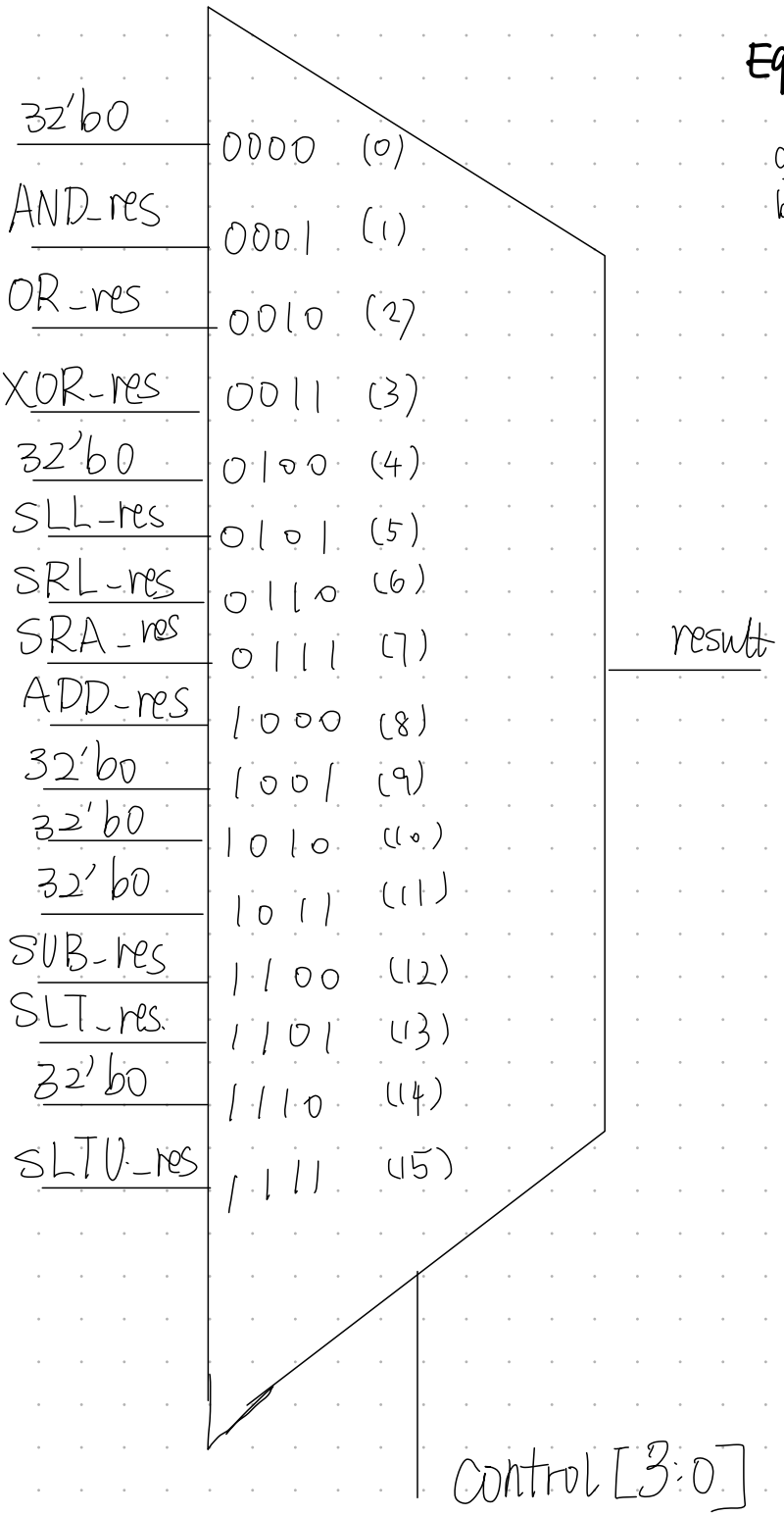
Result.

Zero

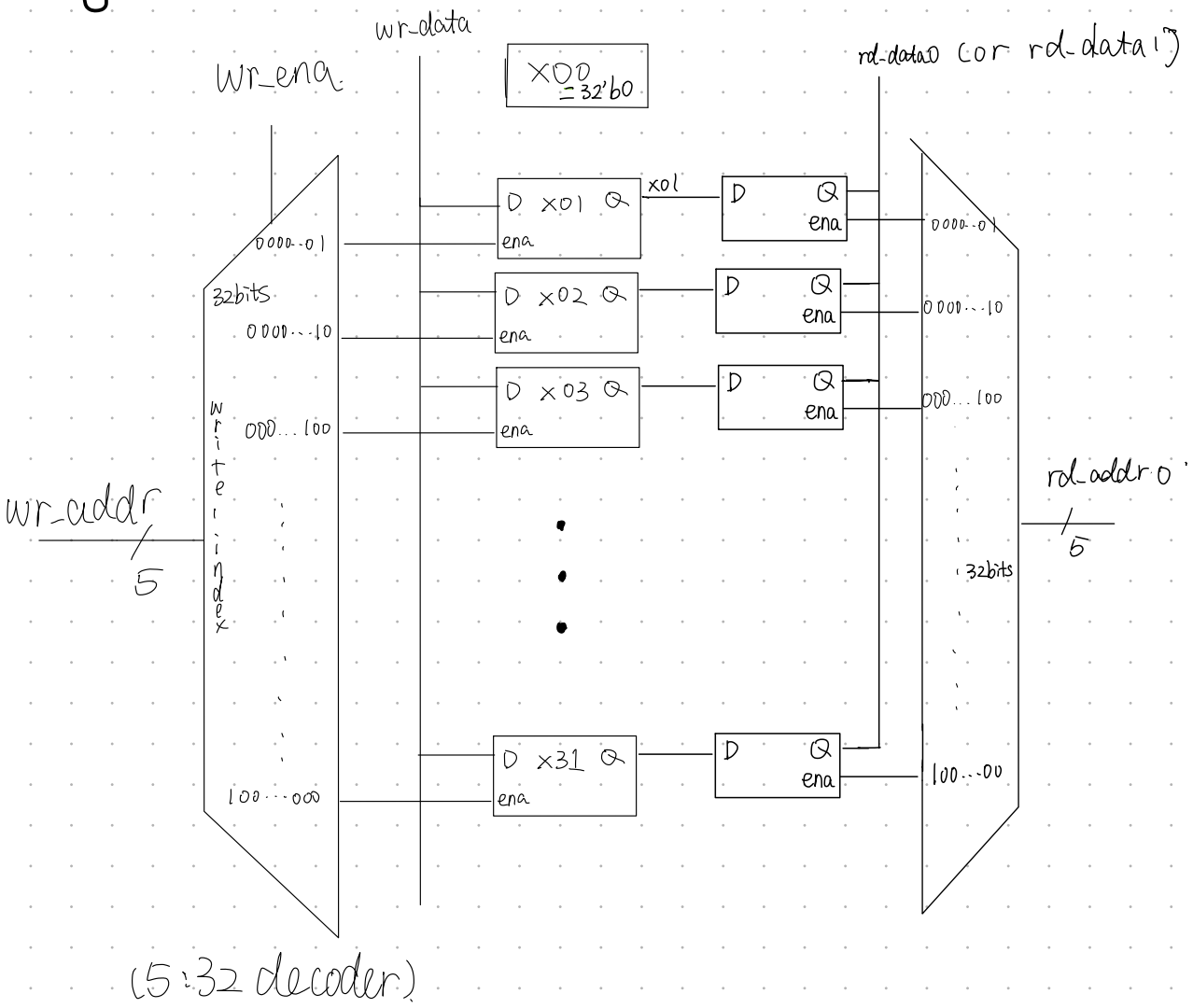


Equal





Register


wr_ena
wr_data
x00 = 32'b0
rd_data0 (or rd_data1)
0000...01
32bits
0000...10
000...100
write_index
wr_addr
5
100...000
D x01 Q
ena
x01
D x02 Q
ena
D x03 Q
ena
D x31 Q
ena
D Q
ena
0000...01
0000...10
000...100
100...00
rd_addr0
5
32bits
(5:32 decoder)