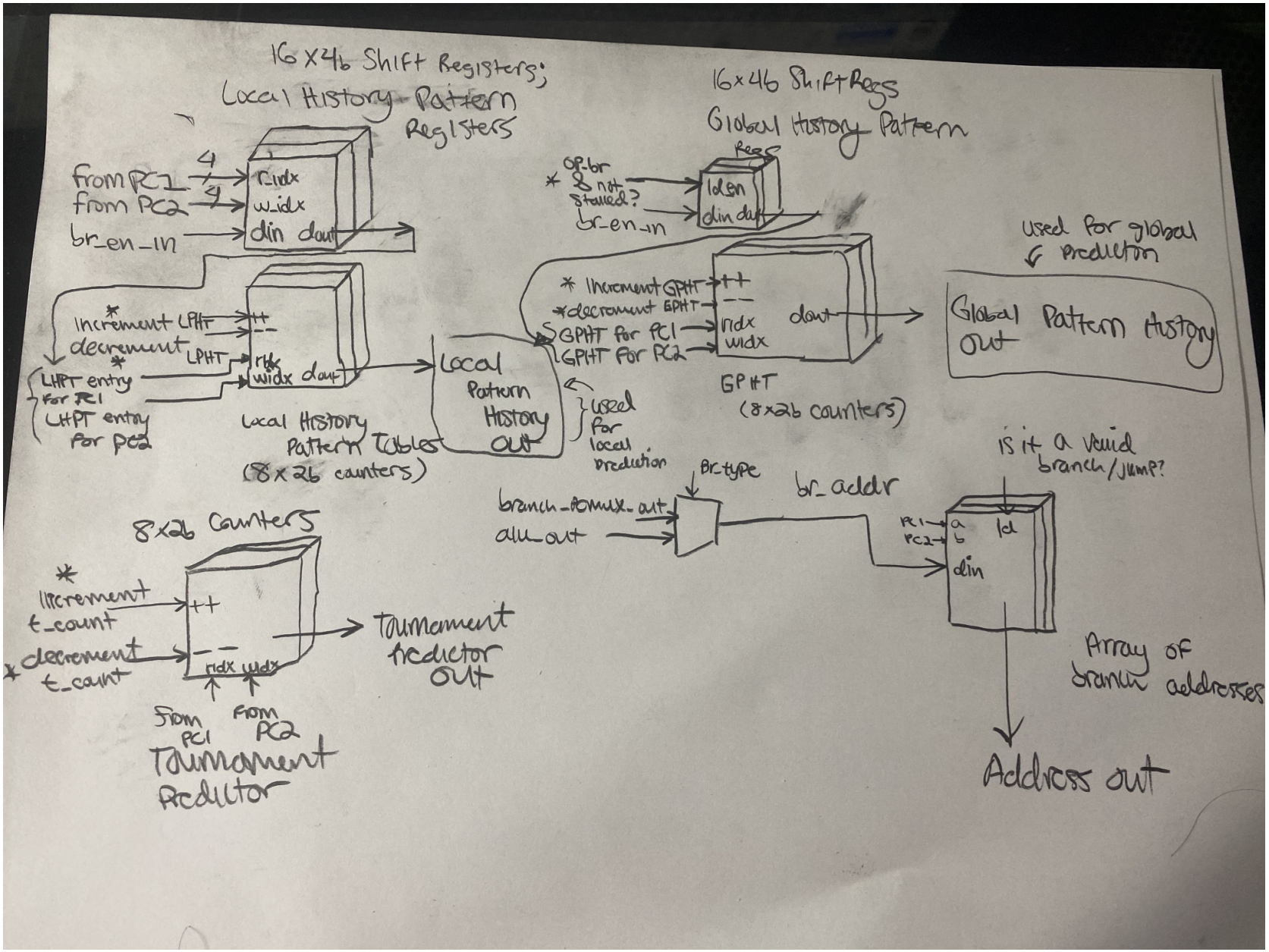
Tournament Branch Predictor:

Inputs: PC_1 (from IF/ID reg), // PC value from IF
PC_2 (from EX/MEM reg), // PC value from EX_MEM reg
alu_out (from EX/MEM reg), // calculated offset for new address (jal/jalr)
branch_pcmux_out (from reg), // next address for branches
br_type (from EX/MEM reg), // whether branch is direct (br)/indirect (jal/jalr)
is_stalled, // state of pipeline
br_en_in // evaluated br_en from EX/MEM pipeline reg
br_local_prediction, // T/NT from local predictor
br_global_prediction, // T/NT from global predictor
/* might be more efficient to pass in the entire pipeline register struct */
if_id_reg // struct
ex_mem_reg // struct

Outputs:
br_out, // T/NT from tournament branch prediction logic
bad_prediction, // nether (L | G) predictor was correct
table_update, // which predictor (L/G) was correct
br_addr_out, // branch address out

/* Logic needed to control counters/shift registers for local predictor and global predictor */
if !is_stalled and ex_mem_reg.opcode is op_br:
load global pattern history registers and allow history pattern tables (counters) to be incremented or decremented
if (br_type == BR)
use branch_pcmux_out value from datapath as target address
else if (br_type == JMP)
use alu_out from EX/MEM pipeline register with computed offset
if (br_en_in from EX/MEM pipeline reg matches br_global_prediction):
increment tournament predictor counters
else if (br_en_in from EX/MEM pipeline reg matches br_local_prediction):
decrement tournament predictor counter
else if ex_mem_reg.opcode is a branch and br_en_in differs from prediction:
if br_local_predictor was right: decrement tournament counter
else if br_global_predictor was right: increment tournament predictor counter
if if_id_reg.opcode is a branch and if_id_reg.pc_address is cached:
if global branch predictor was WT/T: update global predictor
if local branch predictor was WT/T: update local predictor
else if if_id_reg.opcode is jal/jalr and if_id_reg.pc_address is cached:
update both local and global predictors
if tournament branch predictor was WT/T:
br_out = br_global_prediction
update global predictor
else if tournament branch predictor was WNT/NT:
br_out = br_local_prediction
update local predictor



Local Branch P

Eviction write Buffer:

Cache: read_i, write_i, resp_output, rdata_output, addr_input, wdata_input
Physical memory: resp_input, rdata_i, addr_output, wdata_output, read_output, write_output, resp_input

Start:

Read_instruction:

if(hit)

Stay in idle, set resp_out to high and rdata_out to write_instruction

Else

Stay in idle, set resp_out to resp_in and set output to input

Write_instruction:

Go to Pause state and set resp_output to high

Write:

Sends buffer to memory and handles EWB hits

Go to idle state

stalling:

if(read)

Goes to read state

if(write_instruction)

state goes to write

Else

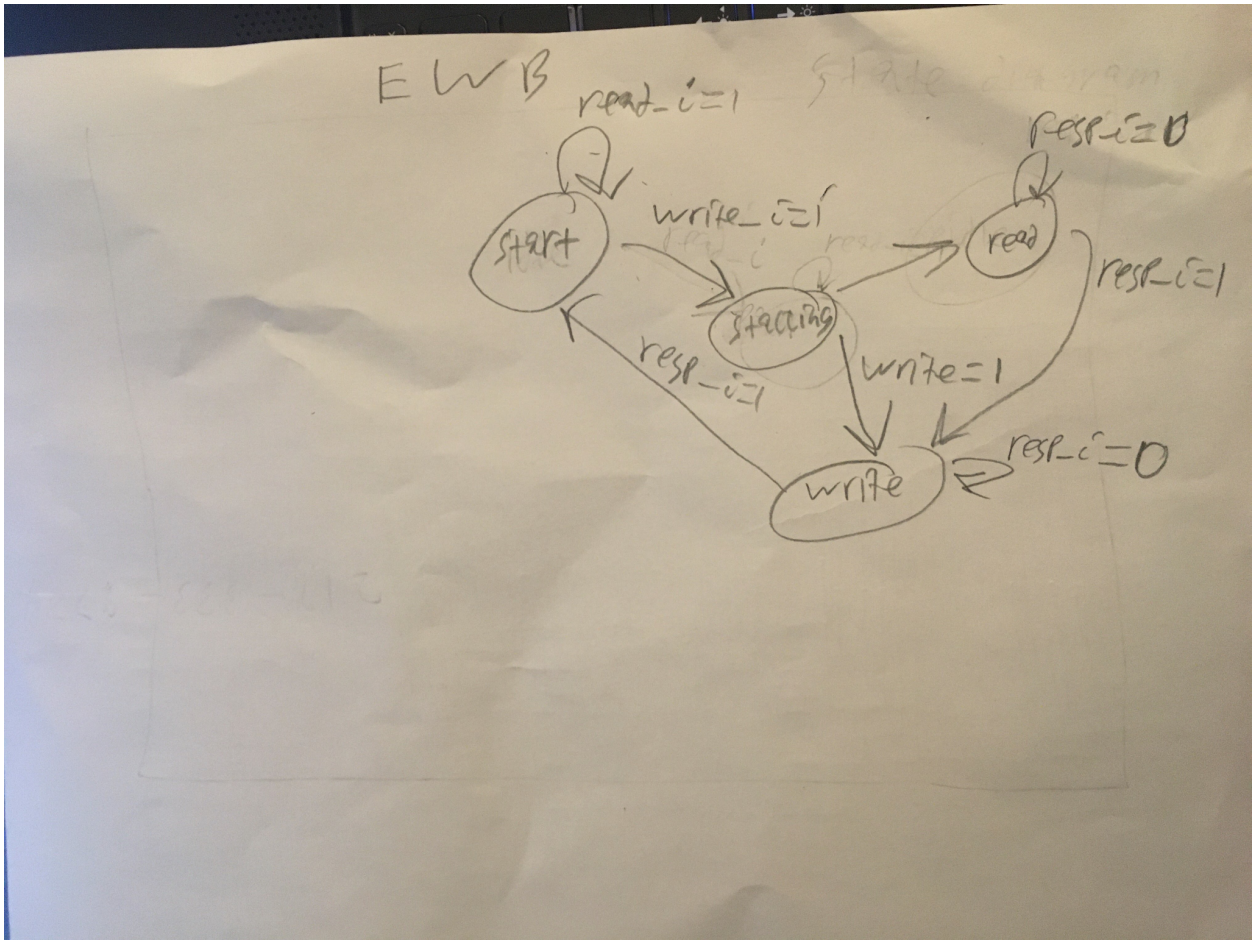
Stay in pause

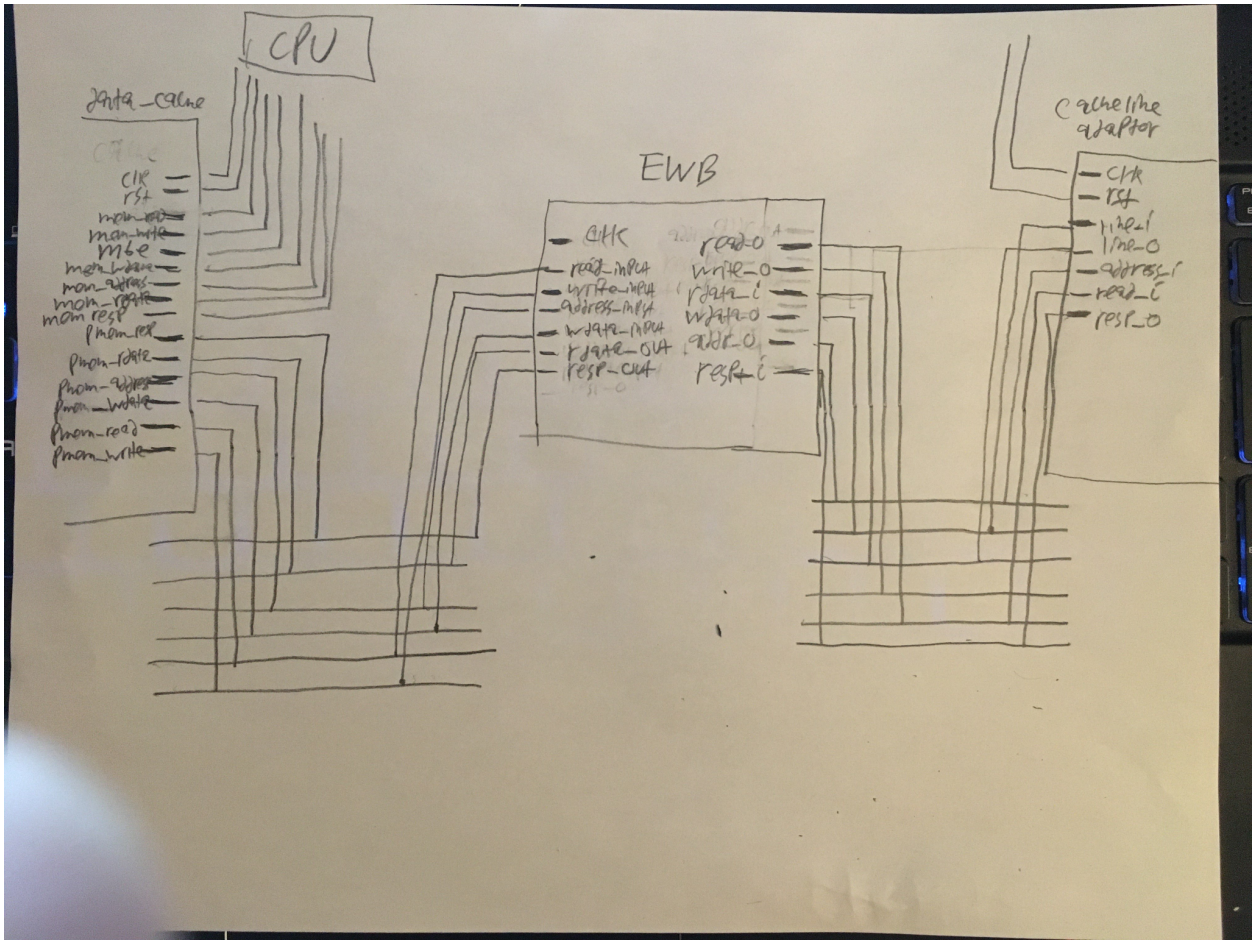
Waits for cache to finish reading or raises a write_input, write back happens when a read is finished.

Read:

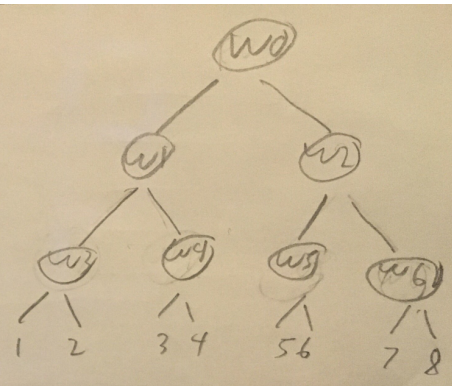
Go to read when resp_i is low otherwise go to write

Sets output to input


EWB
read_i=1
start
write_i=1
reading
read
resp_i=0
resp_i=1
resp_i=1
write=1
write
resp_i=0


CPU
data_cache
clk
rst
mem_read
mem_write
mem_wdata
mem_address
mem_rdata
mem_resp
pmem_resp
pmem_rdata
pmem_address
pmem_write
pmem_read
pmem_write
EWB
clk
read_inPort
write_inPort
address_inPort
wdata_inPort
rdata_outPort
resp_outPort
read_0
write_0
rdata_i
wdata_0
addr_0
resp_i
cacheline adaptor
clk
rst
line_i
line_o
address_i
read_i
resp_o

8-way PLRU



module LRU;

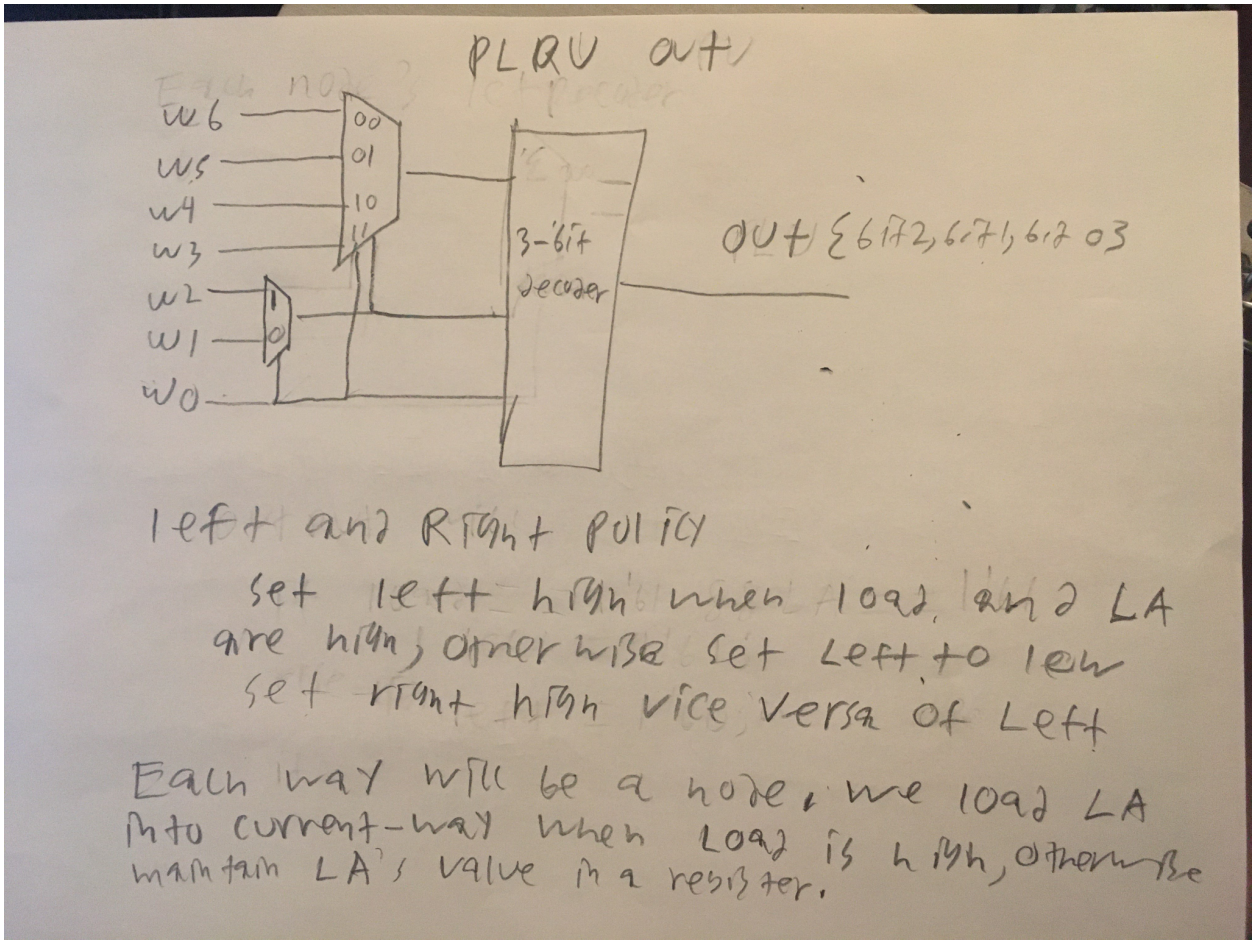
inputs: clk, rst, load, [2:0] LW

output: [2:0] OUT

module Node

input: clk, rst, load, LA, current_way

output: left, right



Bits of LW determines which node to select at each level. LA is the most recent way. W0-w7 are outputs from each node, which is the current way value.

Progress report:

For this checkpoint Max worked on developing the data cache and instruction cache then integrated the two with the arbiter. I worked on designing advanced features, and forwarding. Michael worked on forwarding, testing forwarding and static branch predictor. We all worked on debugging and developed a testbench arbiter for verification. We are still currently testing our checkpoint at this time. Some of the bugs that we found were interfacing with the RISC-V monitor, the arbiter wasn't latched correctly. Another bug when an instruction read and instruction write occurred at the same time, the solution was to reevaluate the state machine.

Roadmap:

For the next checkpoint we plan on implementing PLRU, eviction write back buffer, and tournament branch predictor, and pipelined caches. I will implement the PLRU, michael is going to implement the eviction write back buffer and pipeline caches, and max is going to implement the tournament branch predictor.