ECE411 MP3 CP1 Report

Group: ZerotoOne

TA meeting:
We met our TA, Ahmed, twice during this checkpoint period. The following are some of the useful advice and design recommendations that we received:

1. The best way to quickly calculate the branching result and flushing the pipe: storing results from branch predictor into ID/EX regfile and route results back to PC from EX stage to avoid critical path.
2. How to implement a transparent regfile: tradeoff of using positive clock edge for writing and negative edge for reading v.s. data forwarding.
3. How to implement a faster cache: fewer logic(muxes) and more chips.

Progress Report:
By checkpoint 1, we successfully implemented a pipelined cpu that handles all required RISC-V instructions, where data hazards are not handled. Its functionality was thoroughly tested by our own test program in addition to the given tests. Beside the requirements specified on the mp3 manual, we completed some additional functionalities, listed below:

1. The elimination of control hazards and static branch prediction: the cpu have full support for all br and jump instructions without the need to insert nops. It executes the next instructions after branching by default, and flushes the pipe if a misprediction happens.
2. The elimination of data hazards when performing memory operations: all the pipes will stall if the MEM stage is waiting for the memory response signal, therefore eliminating the need to insert nops.

Beside the implemented functionalities, the design of the following parts are completed and ready to be implemented:

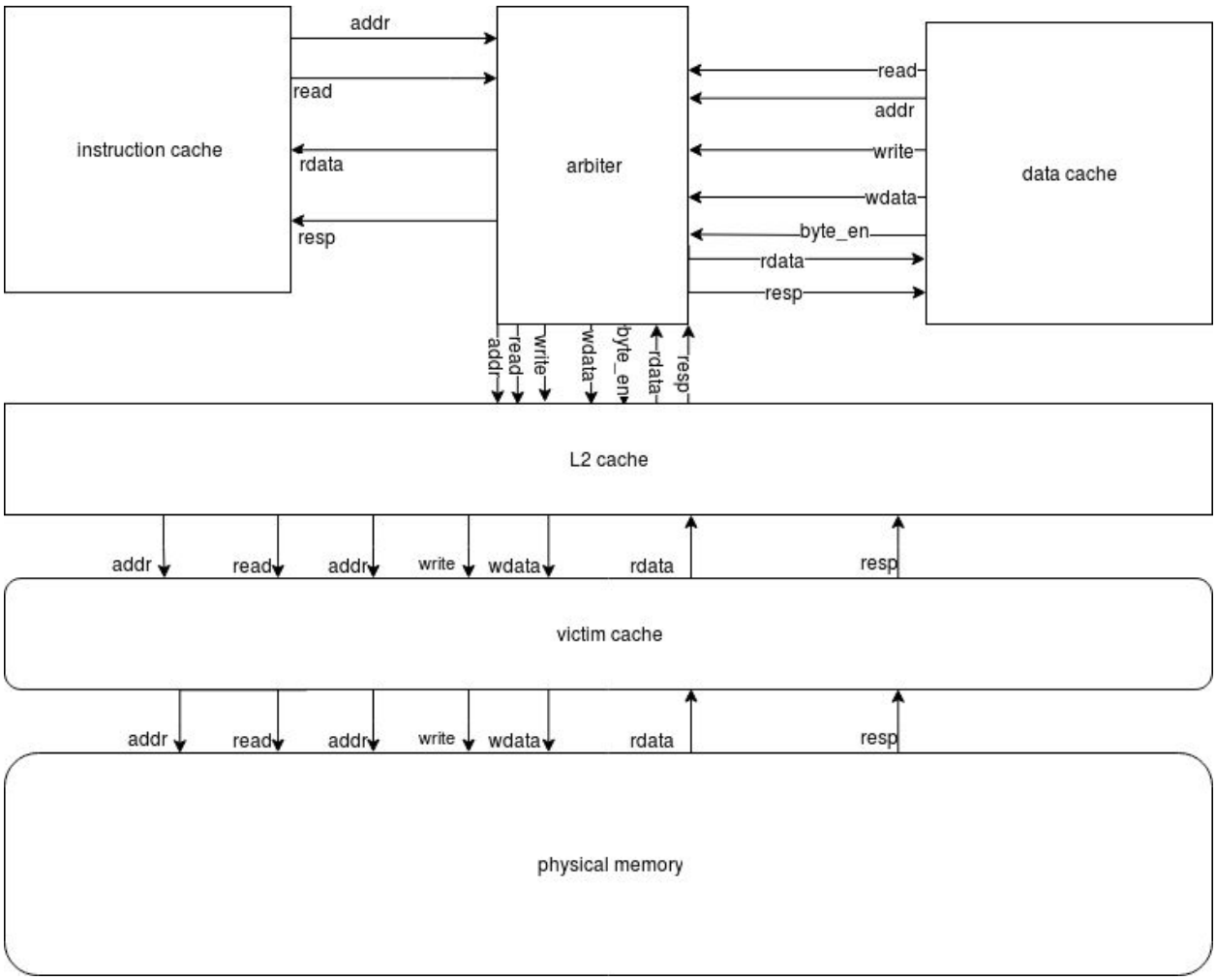
1. Data forwarding
2. L2 cache
3. Cache arbiter

Contribution:
Haichuan Xu: Implementation of pipelines, debugging, testing, forwarding design.
Haiyang Zhang: Implementation of pipelines, debugging, testing, design of the arbiter.
Yuan Ma: Implementation of pipelines, debugging, drawing L2 cache diagram.

Roadmap:
By checkpoint 2, we expect:

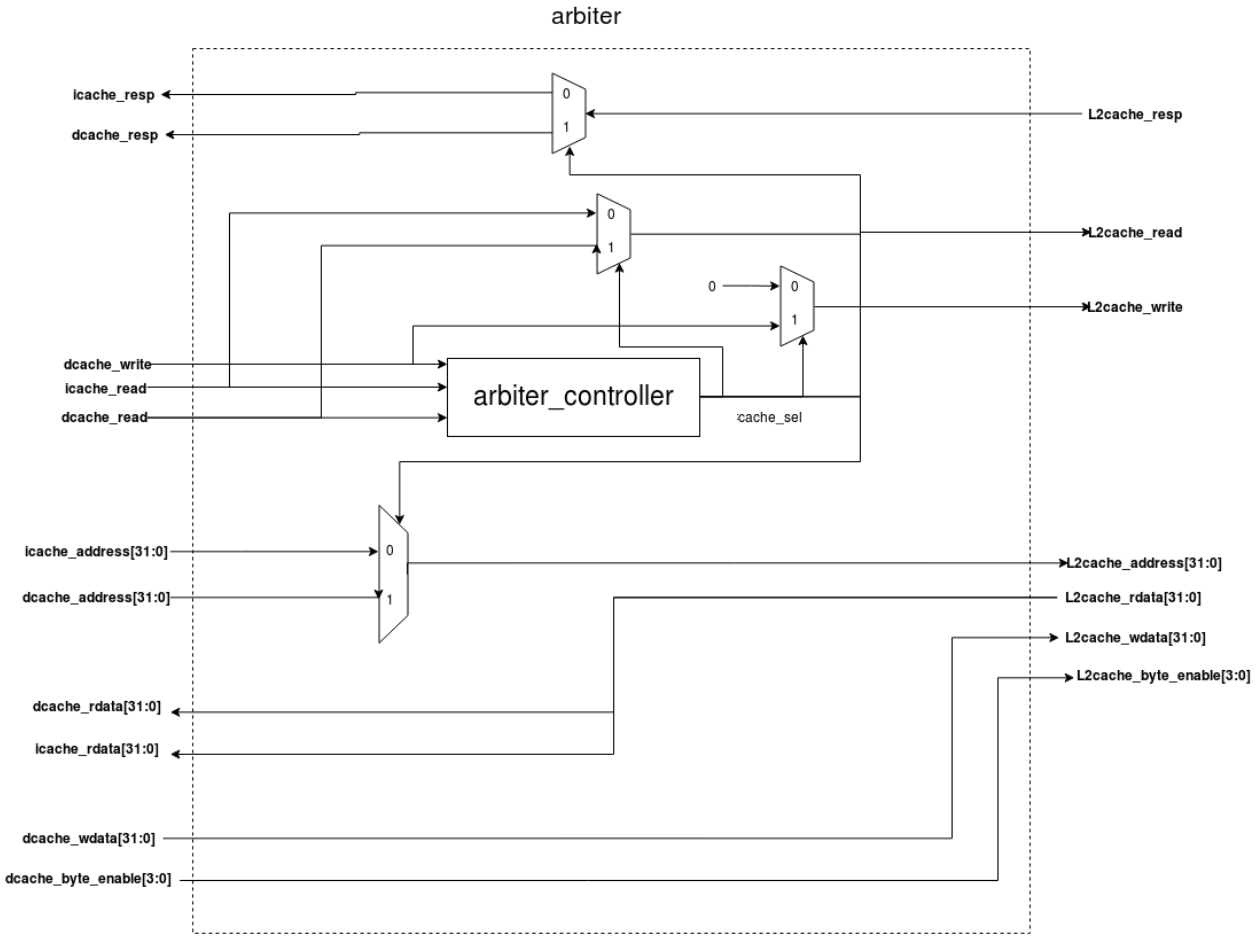
1. Finishing all the requirements on the manual: both the instruction and data L1 cache, as well as the arbiter.
2. Finishing the implementation of data forwarding: both MEM-EX, WB-EX, and regfile, thus eliminating all the data hazards.
3. Finishing the design of some advanced cache features: eviction buffer and victim cache

Memory Hierarchy

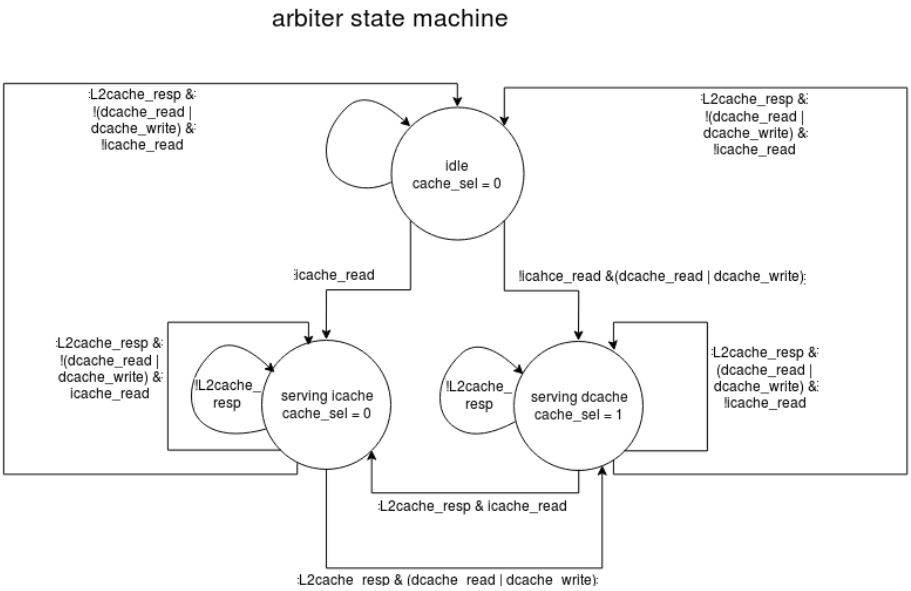


Design of the arbiter

Datapath



Control State machine

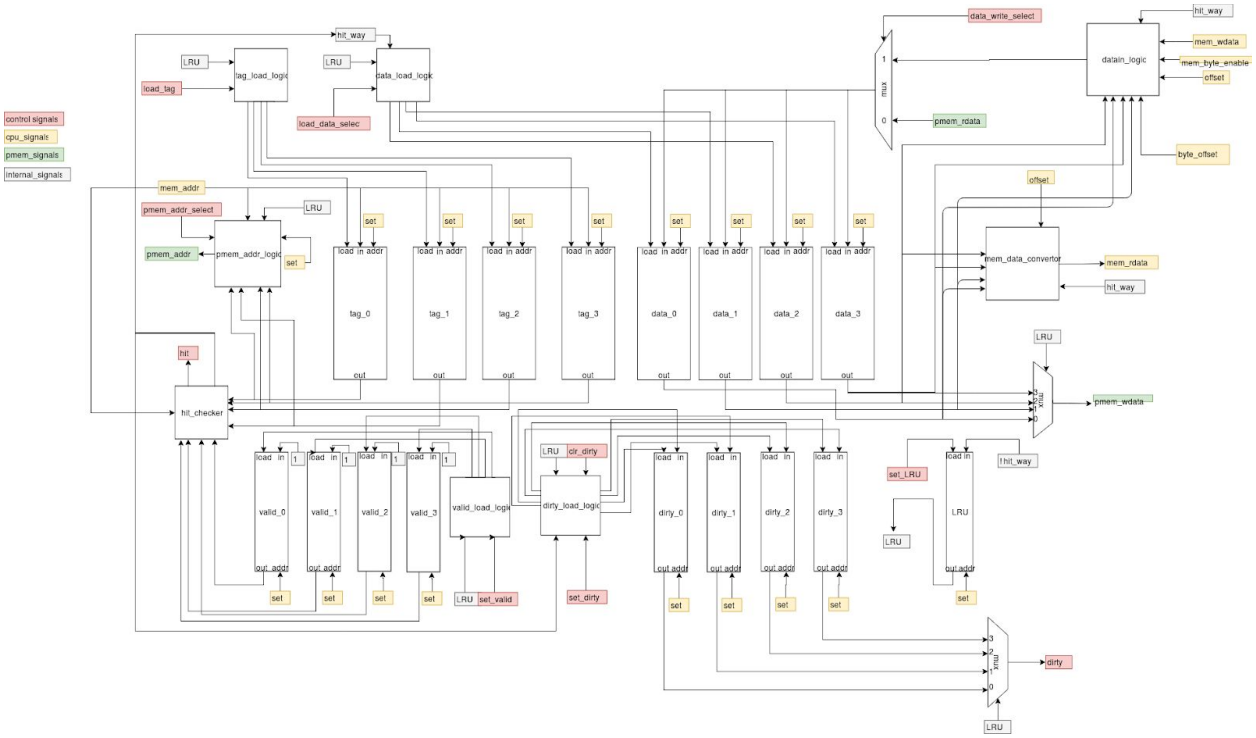


Description:

The arbiter works by choosing either sending all the signals from the instruction cache or all the signals from the data cache to the unified L2 cache. A state machine will output a signal, cache_sel, which tell the muxes to let the signals of one of the caches to go through. The state machine does cache selection in a round-robin fashion. Instruction cache will be prioritized if the arbiter is idle.

L2 Cache diagram

datapath



Details for the logic blocks in L2 Cache

tag_load_logic

This block is a decoder where if the input load signal is high, it output one of the tag_load signal to be high, according to LRU. Other signals will be low.

data_load_logic

This block is a decoder where if the first bit of the load_data_select is high, then depending the second bit, the block will raise the corresponding output signal according to hit_way or LRU.

valid_load_logic

This block is a decoder where if the input load signal is high, it outputs one of the tag_load signal to be high, according to LRU. Other signals will be low.

dirty_load_logic

This block is a decoder where if the input set_dirty signal is high, it outputs one of the tag_load signal to be high, according to hit_way. If the input clr_dirty signal is high, it outputs one of the tag_load signal to be high, according to LRU. Other signals will be low.

pmem_addr_logic

This block contains two MUX where if the pmem_addr_select is zero, the block outputs {mem_addr[31:5], 5'b0}. If the pmem_addr_select is one, the block outputs the {tag_X_out[23:0],set[2:0],5'b0}. tag_X_out depends on the LRU.

mem_data_convertor

This block is decoder that outputs the correct data according to the offset.

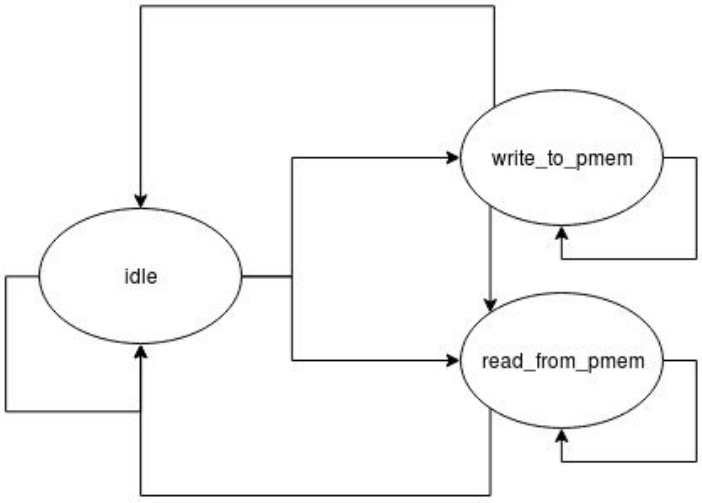
datain_logic

This block will replace the corresponding bits, according to the offset, in the data already in the data array with the input data from CPU.

hit_checker

This block compares the input address to all the tags in the given set in all ways and returns 0 if there is no hit or 1 and the hit way if there is a hit.

Finite state machine



FROM	TO	TRANSITION CONDITION
idle	idle	DEFAULT
idle	write_to_pmem	(mem_read|mem_write)==1 && hit==0 && dirty==1
idle	load_from_pmem	(mem_read|mem_write)==1 && hit==0 && dirty==0
write_to_pmem	write_to_pmem	pmem_resp==0
write_to_pmem	load_from_pmem	pmem_resp==1
load_from_pmem	load_from_pmem	pmem_resp==0

load_from_pmem	idle	pmem_resp==1

STATE	STATE DESCRIPTION

idle	mem_resp=(mem_read|mem_write)&hit
	load_LRU=(mem_read|mem_write)&hit;
	if (mem_write&hit) {
	load_data_select=2'b10;
	set_dirty=1;
	}

write_to_pmem	pmem_addr_select=1 (select {TAG in the LRU way ,set,5'b0}) clr_dirty=1 pmem_write=1
load_from_pmem	load_data_select={{pmem_resp},1'b1} (MSB: load enable; LSB:load the data to LRU way) data_write_select=1 (select pmem_rdata) pmem_addr_select=0 (select {addr[31:5], 5'b0}) pmem_read=1 load_tag= pmem_resp set_valid=1
DEFAULT	mem_resp=0 load_LTU=0 load_tag=0 set_valid=0 set_dirty=0 clr_dirty=0 load_data_select=2'00(MSB: disable data loading) data_write_select=0 pmem_addr_select=0 pmem_write=0 pmem_read=0