Modules

I only add dcache_controller and dcache_sram modules.

dcache_controller:

I only modified three parts of the module. The first part is turning 256-bit r_hit_data into 32-bit data the CPU requested. Since 256-bit is 8 words, and the requested word can't be across different words in a block, so I send back the following range of r_hit_data back to CPU, **[(cpu_offset>>2)*32, (cpu_offset>>2)*32 + 32)**.

The second part is the opposite, turning 32-bit write data to 256-bit block. I first assign read_hit_data to w_hit_data, and write the specific range similar to above formula to _hit_data.

The third part I modified is the state transition part. When STATE_MISS and dirty, I assign mem_enable = 1, mem_write = 1 and write_back = 1, therefore it will perform write back in the next cycle. For STATE_MISS but not dirty, I assign mem_enable = 1, mem_write = 0 and write_back = 0, since we only need to read from memory. For STATE_READMISS and mem_ack, I assign mem_enable = 0, cache_write = 1, since we are going to write the data from memory to cache. For STATE_READMISSOK, I simply assign cache_write back to 0. Finally, for STATE_WRITEBACK and mem_ack, I assign mem_write = 0, write_back = 0, because the writing action to memory is done.

dcache_sram:

There are two main parts for sram, reading and writing. I add a new variable reg [1:0] last[0:15] to track the last used block for each set. If the value is 00, it means both cache blocks are not used before, if 01, means the last used block is the 1st block, if 10, means the last used block is the 2nd block.

When writing, I first check if the input tag is equal to any tag of the cache set (tag[addr]). If found, I modified the found block and set the dirty bit on and set last[addr] to the corresponding value. If not found, it will replace a block based on the value of last[addr]. After replace, it sets the dirty bit to 0, valid bit to 1, and the tag to the input tag, and also modifies the value of last[addr]. By the way, if last[addr] is 00, that means both cache blocks are not used before, so I simply choose the 1st block to replace.

For reading, I first check if the input tag is equal to any tag of the cache set (tag[addr]). If found, I output the corresponding data, tag, and set hit_o to 1. If not found, I output the data and tag of the block which was last recently used by checking the last[addr] value, and also set hit_o to 0.

Others:

I connected the dcache module under CPU, and added stall signal input to all pipeline registers, and connected them with the output stall signal of dcache controller.

Difficulty

The main difficulty is that the given code is so hard to understand, there are so many variables, and I have to spend a lot of time trying to figure out what they mean.

And the debugging of this lab is also quite difficult too, we can not see the data in cache directly in gtkwave, so I have to print those variables out to debug. Also, my output number of cycles is different from the given one, and I don’t know why.

Development Environment:
macOS Big Sur 11.4