Class: ECE 411 Spring 2018
Assignment: MP3 CP2 Report
Due: March 18, 2018
Members: Harsh Modhera (hmodhe2), Ben Li (cli91), Forrest Hornitschek (hornits2)

Checkpoint 2: Progress

For this checkpoint, we were tasked with providing functionality for all remaining LC3b instructions, as well as the integration of two L1 caches, one strictly for fetching instructions and another for reading/writing data. The toughest instruction to coordinate with the rest of the pipeline was the LDI/STI opcode, due to its need for multiple requests from memory. This required keeping a controller outside the dcache to detect such an instruction, and keep it in the MEM stage while it retrieves the address from memory to make the subsequent load or store. This also introduces the need for a stall of the pipeline for all previous stages whenever there is a dcache miss. In the case of a TRAP, once the memory completes its request, a reset of the previous pipeline registers occurs to replace currently executing instructions with NOPs, which will harmlessly pass through the pipeline as branches not taken. From checkpoint 1, both the instructions and data were connect to their own instances of magic memory. We progressed through this checkpoint by first connecting data to its own physical memory, then inserting a dcache before memory, then instructions as an icache with its own physical memory, and finally connecting both to a single instantiation of memory as determined by the arbiter. The arbiter uses a state machine to control which cache is making its request to memory at a given time. Because both may wish to read from memory at once, as a temporary design choice, we decided to give the dcache priority over the icache. In such a scenario, the icache will ignore an ACK signal that arrives specifically to declare the data request as complete, and wait for its turn to read from memory. To avoid starvation for repeated data requests, the icache must finish before dcache is serviced. Because there are now two caches for a single memory, there may be instantaneous requests immediately following an ACK response, which cannot be supported by physical memory, since it requires STB & CYC to be low for at least one cycle following a read or write. So, a one cycle wait state is enforced between consecutive requests. The L2 cache will be included in the next checkpoint between the arbiter and physical memory.

Checkpoint 2: Bugs

Because we inserted a cache design implemented individually from the previous MP, there needed to be slight fixes in its communication with the CPU for selecting the appropriate word from the block. The module sel_bytes.sv was included in the CPU to determine the SEL signal based on the offset of the memory address and the current byte_enable. Before making this addition, the cache was retrieving the correct 128-bit block but the incorrect word.

We noticed in the waveform that while the stall signal was high, meaning nothing within the pipeline registers should change, the branch would not successfully jump in the program once the stall was lifted and the branch was free to execute. Within the stall window and after just one cycle, the control word would change to all zeroes. We knew it must be the reset signal, meant to clear the registers after a branch is taken, that was overwriting these values. However, it should have been waiting until the stall signal went low to execute its instruction and change to PC. So, to fix this error we included the condition that no stall may occur for a reset to be valid.

When attempting to have the arbiter control input to memory, a problem would arise when the icache makes a request followed by a dcache request while icache is being serviced. Because the dcache was assigned an ACK with no limitations, due to its priority over icache, it would falsely believe that the ACK meant for the instruction request also meant its own read or write was completed. The solution was to add the condition to dcache's received ACK that the arbiter must be in either the single data request or dual data/instruction request state.

Checkpoint 2: Testing

For testing this checkpoint, we ran ModelSim on shorter test programs and compared register values with the results of the LC3b simulator by stepping through its execution. Because we now have a cache included as a component of the processor, we wrote a test case that reads and writes to memory that includes addresses with the same index. We could then monitor the data arrays to ensure the correct way and line were being written to and/or evicted from. For larger programs, such as the CP2 B test, we ran the test-o-matic executable that provided a memory list and an array of all changes to the register file. This made it easier to spot where the program first produces an error when run on our pipelined processor. We also wrote our own test code to sufficiently test instructions such as TRAP, JSR, LEA, and LDI/STI. For all test cases we ran, the register file matched the correct final register values and would properly stop execution for an infinite branch.

Checkpoint 2: Member Contribution

Harsh and Ben began the checkpoint by adding the control necessary for stalling and resetting the pipeline registers. This includes the proper behavior for LDI/STI and TRAP instructions in the MEM stage of the pipeline. Ben tested and verified each LC3b instruction to ensure the correctness of the pipeline when connected to magic memory. Forrest inserted the split L1 caches into icache and dcache, then made necessary tweaks in control logic to verify execution still behaves correctly when the delay from physical memory is introduced. Harsh and Forrest then created the arbiter unit that allowed us to switch to a single physical memory block that both caches make requests to. Harsh modified our paper design to include an initial plan for data forwarding in the next checkpoint, while Forrest wrote up the progress report for the current checkpoint.

Checkpoint 3: Roadmap

For checkpoint 3, we read up on data and control hazards in the textbook starting from section 4.7. Although the examples and details provided were for the MIPS architecture, we tried our best to map it to our LC3b version. We used a paper copy of our datapath and added logic for the forwarding in order to overcome data hazards. For control hazards, there were two small methods mentioned. One was to assume branch is not taken and squash the pipelines when it is determined otherwise. The second was to determine the branch earlier in the pipeline. When we did checkpoint 0, we actually accounted for this, and implemented the branch taken logic in the execute stage and the reset logic to squash registers on a taken branch. For this checkpoint, we will work heavily on the forwarding aspect and get our design working without the need for nops. The L2 addition should be an easy addition to this task and we have split L1 caches working with an arbiter connected to one physical memory instance currently.

Checkpoint 3: Proposed Member Contribution

Ben: Begin roadmap and design plan for branch prediction and advanced features in the upcoming checkpoint

Harsh: Create forwarding unit module and connect additional muxes and signals to pipeline for proper data forwarding and hazard detection

Forrest: Set up connections for L2 cache and integrate into cache hierarchy using control from arbiter module.

All: Test and debug any errors, verify functionality of new implementation

*Subject to change