EE312 Lab 6: Cache Lab

Prairsunan Chanpanich 20190747
Sorn Chottananurak 20200819

1. **Introduction**

   For the last assignment of the EE312 Computer Architecture course, we are requested to implement cache on the pipelined CPU model we have implemented before. To get a hands-on experience of the knowledge we obtained from the lectures, a direct mapped cache structure is used for the memory-related instructions, and cache miss cases are managed with different penalties for each cache miss type.

2. **Design**

   The program is based on our pipelined CPU program, with some modification. By adding cache register files to determine cache status and insert some conditions in the sequential logic for value update, cache read miss and write miss are handled. The policies used for cache write-hit and write-miss are write-through and write-allocate. The number of cycles used for cached hits, data memory access when cache misses, and for cache update must be implemented carefully.

3. **Implementation**

   The implementation of CPU with cache in this assignment differs from the pipelined CPU by the introduction of cache parameters. First, variables to contain cache states are declared and some are then initialized. The implementation of the pipelined CPU then starts as in the previous assignment.

   In the IF stage, the PC is only stalled when load hazards occur. The ALU result is then calculated for every type of instruction. In the ID stage, the stage is abandoned when one of the hazard conditions is matched, which is the same implementation from the pipelined CPU.

   Before going into the EX stage, the rest of the declaration of variables to handle caches are implemented here. The flags are set whether the situation is read hit/miss or write hit/miss or neither. In the EX stage, apart from the old implementation, the registers to contain cache miss states are also parsed to the next stage. The MEM stage registers are updated. Then, a combinational logic to handle the cache is implemented. It updates the cache state starting from, entering cache, waiting for a penalty, and the last stage where cache read or write is done. The forwarding unit is implemented next, before the main sequential logic to update values. Here, before proceeding to the same method we used in the pipelined CPU, it is checked whether a cache miss occurs or not. For each stage of the cache read miss or cache write miss. First, the penalty cycle is counted to wait for data retrieval. After the number of penalty cycles matches the condition, the cache stage is changed into its last stage for cache output read or write. The penalty cycles are 8 cycles for data memory access and one cycle for cache update, as well as for cache hit(no miss). Therefore, it follows the number of cycle policies stated in the lab direction. When all cache value updates are done, cache states are all reset to the initial value when there is no cache miss, and the updates of the other values proceed as in the pipelined CPU. Lastly, the terminal condition is implemented, followed by register file values and data memory update. The writeback is done as output of the program.

4. Evaluation

Unlike in previous assignments, we tested our program twice, first with cache implementation and second without cache implementation by changing the code in RISCV_TOP.v, so we can see and compare the number of cycles. We get all tests passed for both programs with the number of cycles as follows.

With cache: inst - 46 cycles, for loop - 222 cycles, sort - 28153 cycles
Without cache: inst - 43 cycles, for loop - 337 cycles, sort - 53985 cycles

It can be seen from the results that the number of cycles for sort and for loop tests decrease a lot with the implementation of cache microarchitexture. This is because cache decreases time to access the memory unit. However, the number of cycles for the inst test increases a bit for the cache implementation. The reason is that cache hit requires one more cycle. Overall, cache implementation still benefits CPU in terms of memory access time.

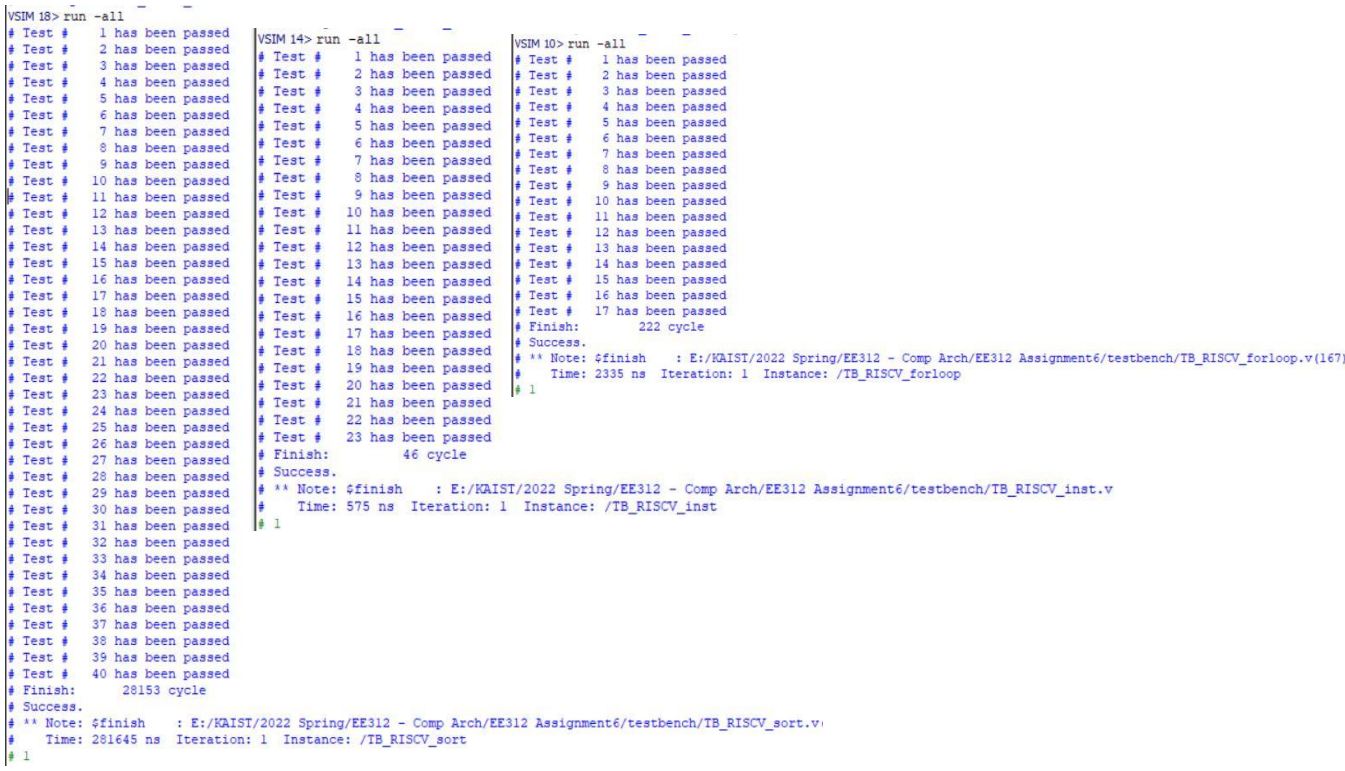

Fig. 1. Simulation results for CPU with cache of sort, inst, and for loop tests.

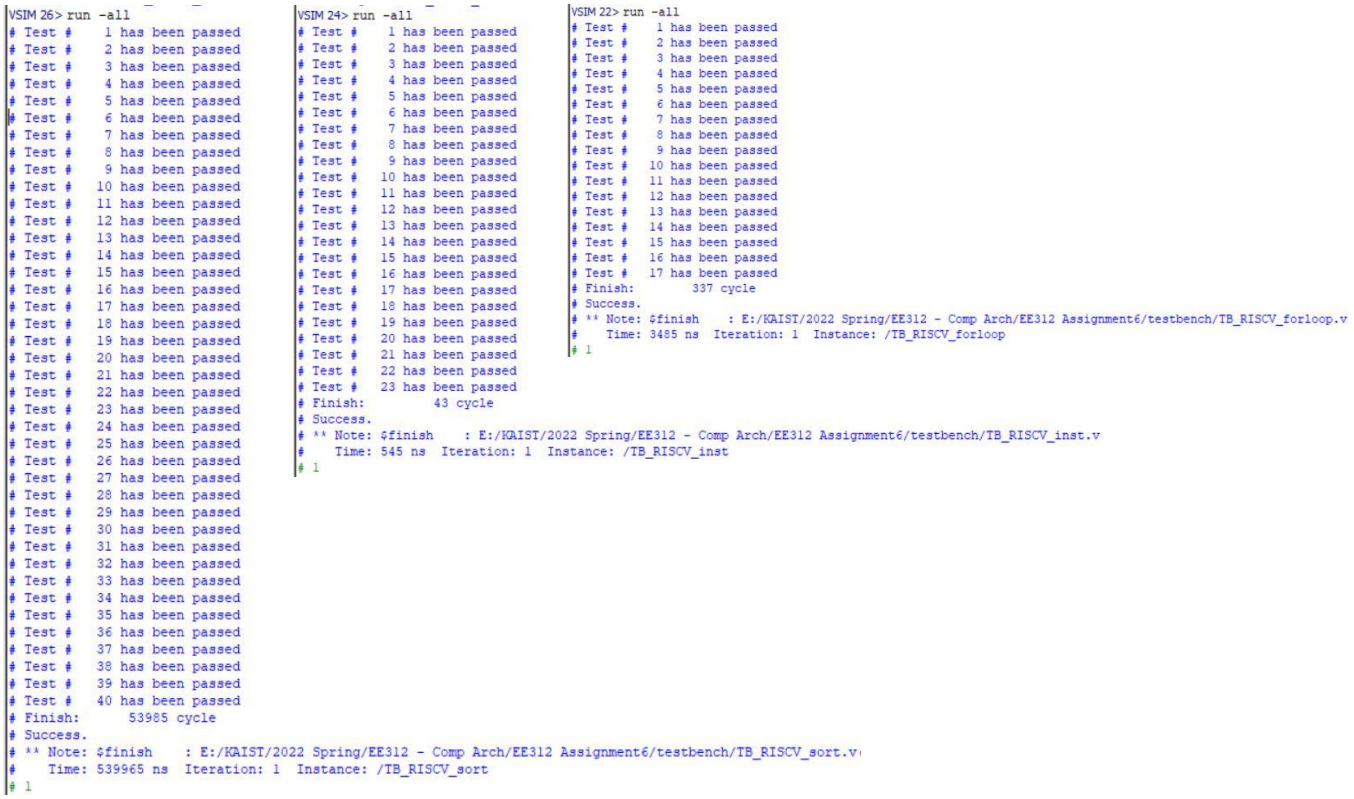

Fig. 2. Simulation results for CPU without cache of sort, inst, and for loop tests.

5. Discussion

The load of this task is appropriate to be the final task when students are having final exams. It is good that we don't have to implement the pipelined CPU as a final task.

Note that we submit both with cache and without cache versions of RISCV_TOP, with the names RISCV_TOP.v for cache and RISCV_TOP_nocache.v for no cache implementation.

6. Conclusion

Through the implementation of a pipelined CPU with cache microarchitecture, we have learned how a direct-mapped cache will be applied to a CPU with the write-through and write-allocate policy for cache write hit/miss. This task has improved our understanding of the process of cache and memory accesses.