Extra credit description

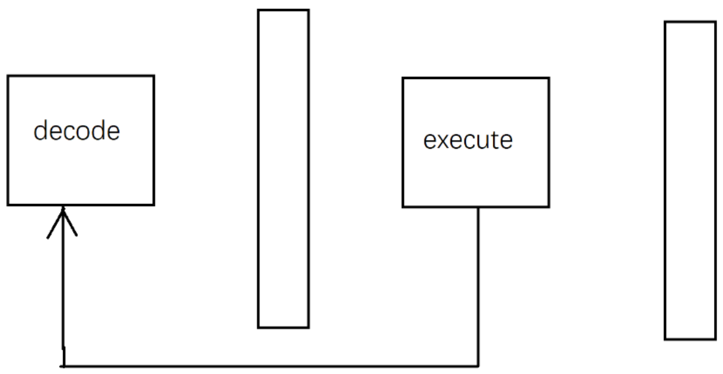
For the extra credit options, we did four of them:

- resolve branch in decode stage
- forwarding to ID stage and mem-to-mem forwarding
- using dynamic branch prediction.
- Use LRU for cache replacements

For resolving branch in decode stage, we make the decision in decode stage according to the signals we decoded from the instruction. Compare to resolve in execute, resolve in decode can save one cycle for making decision when encounter the branch instruction, but the area will increase because of the extra logic in decode stage.

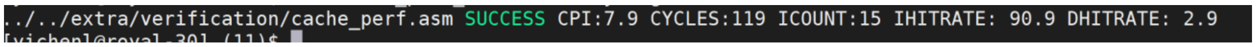
For mem-to-mem forwarding, we add the forwarding path from memory stage to the input of the pipeline registers in execute stage. It is useful when we have a store instruction after load instruction, where the load is writing the value of the load instruction. Using mem-to-mem forwarding can help to reduce one cycle use in this condition, but it will add extra forwarding logic and increase area (In our design it is an exception because we used 2-cycle read in our cache, so whenever we need mem-to-mem forwarding, we always consume one more cycle in cache read and do not need to stall in our original design, which makes the mem-to-mem forwarding not effective to our design).

For exe-to-branch forwarding, we add forwarding path from execution to decode stage. Previously in exe-to-exe, we forward execution data before getting into pipeline register to decode stage just before decode's pipeline register. It gives us opportunity to feed execution results into decode branch resolution with the same forwarding path.



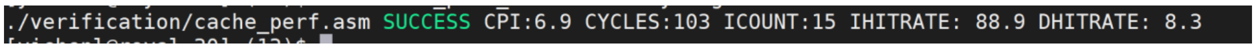
For using LRU for cache replacements, we use a reference bit storing the most recently used cache. Then when we want to do replacements, we will replace the other one (the least recently used). The benefit is that generally the least recently used is less likely to be accessed, so the general hit rate can increase. However, there are still overheads. If the sequence of accesses is to access the least recently used ones that we just replaced with new data, then we will continue to get misses.

Old results:



```
../../extra/verification/cache_perf.asm SUCCESS CPI:7.9 CYCLES:119 ICOUNT:15 IHITRATE: 90.9 DHITRATE: 2.9
```

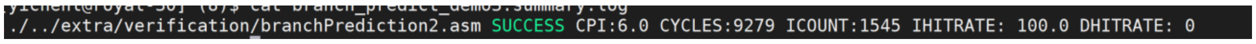
New results:



```
./verification/cache_perf.asm SUCCESS CPI:6.9 CYCLES:103 ICOUNT:15 IHITRATE: 88.9 DHITRATE: 8.3
```

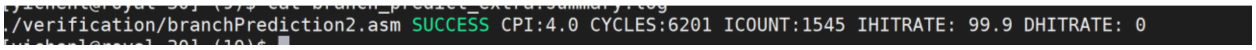
For dynamic branch prediction, we intended to increase the rate of branch predictions to increase CPI and improve performance. To achieve that goal, we store the previous action at that position of the branch, and use the record to predict the branch decision (if the previous action was to take the branch, then we will take it again this time). Compared to the original design, dynamic branch prediction can reduce the number of prediction failures when we meet a for-loop that need to execute many times. Using dynamic branch prediction, we have additional logic and will increase the total area.

Old results:



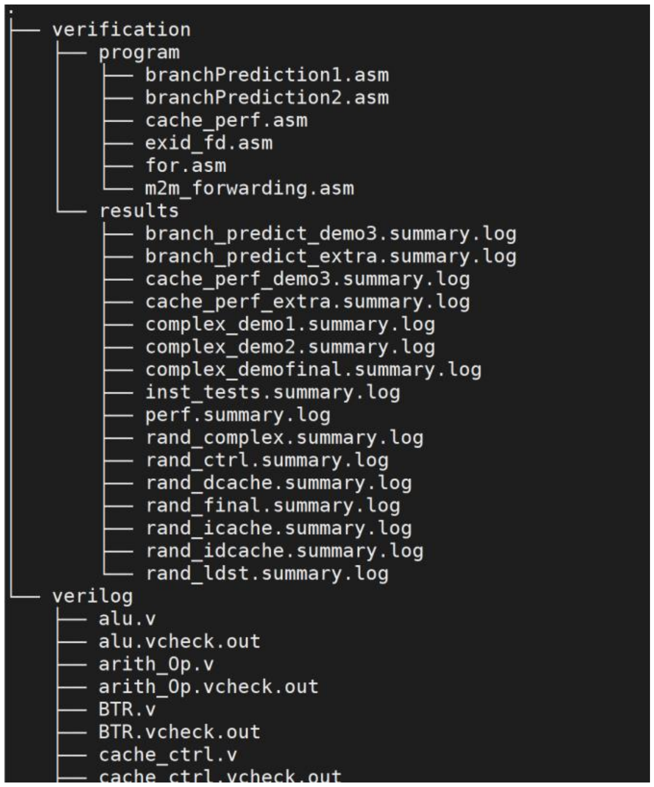
```
./../extra/verification/branchPrediction2.asm SUCCESS CPI:6.0 CYCLES:9279 ICOUNT:1545 IHITRATE: 100.0 DHITRATE: 0
```

New results:



```
./verification/branchPrediction2.asm SUCCESS CPI:4.0 CYCLES:6201 ICOUNT:1545 IHITRATE: 99.9 DHITRATE: 0
```

The structure of our upload files is:



```
.
├── verification
│   ├── program
│   │   ├── branchPrediction1.asm
│   │   ├── branchPrediction2.asm
│   │   ├── cache_perf.asm
│   │   ├── exid_fd.asm
│   │   ├── for.asm
│   │   └── m2m_forwarding.asm
│   └── results
│       ├── branch_predict_demo3.summary.log
│       ├── branch_predict_extra.summary.log
│       ├── cache_perf_demo3.summary.log
│       ├── cache_perf_extra.summary.log
│       ├── complex_demo1.summary.log
│       ├── complex_demo2.summary.log
│       ├── complex_demofinal.summary.log
│       ├── inst_tests.summary.log
│       ├── perf.summary.log
│       ├── rand_complex.summary.log
│       ├── rand_ctrl.summary.log
│       ├── rand_dcache.summary.log
│       ├── rand_final.summary.log
│       ├── rand_icache.summary.log
│       ├── rand_idcache.summary.log
│       └── rand_ldst.summary.log
└── verilog
    ├── alu.v
    ├── alu.vcheck.out
    ├── arith_Op.v
    ├── arith_Op.vcheck.out
    ├── BTR.v
    ├── BTR.vcheck.out
    ├── cache_ctrl.v
    ├── cache_ctrl.vcheck.out
```