Lab 4: Cache Simulator Report

0511097 黃柏豪 0516329 黃昱彰

1. Basic Problems:
2. **Verilog:**

We use .v codes of lab 3 to implement the whole CPU.

* **Changes:** 
  + **In “Instruction\_Memory.v”:** Change the size of instruction\_file from [0:31] to [0:64], and read input data from lab4\_test\_data.txt.
  + **In “TestBench.v”:** Change the whole testbench.v code into the testbench.v code that in supplied files.
* **Outcome:** 
  + After simulating the CPU, it writes ICACHE data and DCACHE data into correspond .txt file.

1. **cpp codes:**
2. **direct\_mapped\_cache.cpp**

* **In function “simulate”:** We add two double type number hit and total to calculate the hits when access data. Therefore, we can know the hit rate and miss rate(1-hit\_rate)
* If the cache hit, we add 1 to variable “hit”.

And we add 1 to variable “total” when each time read the input data.

* hit rate = ( hit / total )\*100;
* miss rate = (( total – hit ) / total ) \*100;
* **In main function:** We use two for loop to run the call function of simulate.
* int i, j;

for(i = 1 ; i < 5 ; i++){

for(j = 0 ; j < 5 ; j++)

simulate(pow(2, i\*2) \* K, pow(2, j+4));

cout << endl;

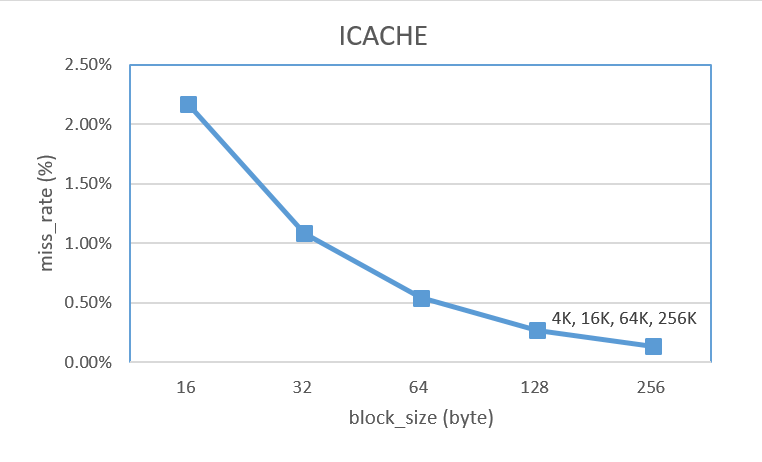
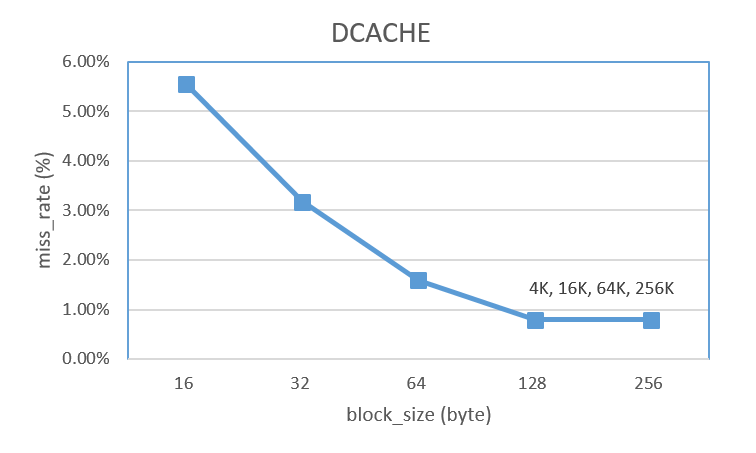
}

Because it has to run cache\_size in 4K, 16K, 64K, 256K

* They are all 2 to the power of i\*2
* 2^(1\*2) K, 2^(2\*2) K, 2^(3\*2) K, 2^(4\*2) K

And the block\_size are 16 B, 32 B, 64 B, 128 B, 256 B

* They are all 2 to the power of j+4
* 2^(0+4) B, 2^(1+4) B, 2^(2+4) B, 2^(3+4) B, 2^(4+4) B



1. **Direct\_mapped\_cache\_LRU.cpp**

* **In struct cache\_content:** We add one more integer array called “data” which has size of 16 to save data.
* **In function “simulate”:** We need to add one more parameter when calling the function which is “ways”, so that we can know how many ways does it have now.
* **In main function:** We use two for loop to run the call function of simulate.
* int i, j;

for(i = 0 ; i < 6 ; i++){

for(j = 0 ; j < 4 ; j++)

simulate(pow(2, j), pow(2, i) \* K, 64);

cout << endl;

}

Because it has to run 1-way, 2-way, 4-way, 8-way

* They are all 2 to the power of j
* 2^0-way, 2^1-way, 2^2-way, 2^3-way

And cache\_size are in 1K, 2K, 4K, 8K, 16K, 32K

* They are all 2 to the power of i
* 2^0 K, 2^1 K, 2^2 K, 2^3 K, 2^4 K, 2^5 K

※If you run the code, it will automatically print out its cache line, cache\_size, block\_size, miss\_rate, and n-way associative.