Advanced Computer Architecture: Lab 3 Report

Due on Dec 1st at 11:59am

Name: **Zhenghong Yu**
Mailbox: **yuzhh1@shanghaitech.edu.cn**
Student ID: 2020533156

Contents

0.1 Preknowledge

You should read following carefully, since they are very important to Lab3 implementation
The Lab2 is implemented in the branch **Lab3**, both tasks(Task1, Task2) are all realized in this branch.
To specify two cores, you may follow:

- For all task input, the format should be like: **./Simulator -c0 example1.riscv -c1 example2.riscv** as showed in Lab3 pdf. You must specify two programs of two core(e.t. you can't let one core to be idle). Also, I do not support other format like **-b** or something else. The branch predict policy is NT in default.
- The **CMakeLists.txt** is rewrite and do not support complier **CacheSim** and **CacheOptimized** anymore, please do not change the **CMakeLists.txt**

To gain more readable ouput, I change **fprintf()** to **sprintf()** to some special char pointer variables(e.t. **core0_out, core1_out**), and after two threads finished, I print these variables out(e.t. **putsline()**). However, the running comments of shared memory operations is still using **fprintf()** since they are strictly synchronized, they should be quite readable and beautiful.
The l1, l2, l3, memory cache line size should all be same and fixed as 32, do not change it.
Due to some strange reason, it rarely (I meet in about only 1 times and I cant repreduction it) will crashed and result in read invalid byte, however the address should be valid. If you run into this situation, just ignore it and rerun program.

1 Task1: Two threads Simulator

1.1 Requirement

- Two cores. Different program different core. Own registers, PC, piplines, two-level private caches. Shared L3 cache, memory.
- Avoid conflict between two programs' address.

1.2 Implementation

1.2.1 How to begin run two threads?

It is common to use **pthread.h** lib to help me do multi thread single program work. I take core 0 to thread 0 and take core 1 to thread 1. The two thread almost own private class and variable except memory, l3cache, some assist global variables. Here is some global definition in **MainCPU.cpp**

```
char *elfFile0 = nullptr; /* For core0 program text */
char *elfFile1 = nullptr; /* For core1 program text */
uint32_t stackBaseAddr0 = 0x80000000; /* core0 stack */
uint32_t stackBaseAddr1 = 0x7fc00000; /* core1 stack */
uint32_t stackSize = 0x400000;  /* Stack size */
uint32_t base0 = 0x10000000;    /* B&B for core 0 */
uint32_t base1 = 0x20000000;    /* B&B for core 1 */
MemoryManager memory;           /* Shared Main memory */
Cache *l3Cache;                 /* Shared l3 cache */
Cache *core0l1Cache, *core0l2Cache; /* core0 private cache */
Cache *core1l1Cache, *core1l2Cache; /* core1 private cache */
Cache::Policy l1Policy, l2Policy, l3Policy;
BranchPredictor::Strategy strategy0 = BranchPredictor::Strategy::NT;
```

```
BranchPredictor::Strategy strategy1 = BranchPredictor::Strategy::NT;
BranchPredictor branchPredictor0;
BranchPredictor branchPredictor1;
Simulator simulator0(&memory, &branchPredictor0);
Simulator simulator1(&memory, &branchPredictor1);

int32_t core0_id; /* core0(thread0) tid */
int32_t core1_id; /* core1(thread1) tid */

char core0_out[100005]; /* core0 format output */
char core1_out[100005]; /* core1 format output */
```

Here is some functions in **MainCPU.cpp** used in initilization
bool parseParameters(int argc, char **argv): modified to support multicore input.
void loadElfToMemory(ELFIO::elfio *reader, MemoryManager *memory, uint32_t base): modified with different basebound(introduce later).
void thread_initilization(bool core): New function for different thread running different core's preparing work(e.t. initialize private cache, read in program text, initialize stack).
void *thread_begin(void *arg): New function to start thread(core).
As we can see, the parallel running cores based on two parallel running simulator on different threads.
To start, the **main()** will call **thread_initilization()** to let prepare two threads' variables and classes sequentially.
After this, read in two programs and set them to the corresponding memory area. Here is an import idea on how to distinguish same address of different threads. I use **Base and Bound**(e.t. B&B) thinking to realize. Simplified, this means all one thread's text, data, heap(but not stack) will all add a unique base number, which will make sure that the two threads will not have actually same physical address space. In this code, thread 0(core 0) will add $0x10000000$ and thread 1(core 1) will add $0x20000000$.
Then, we use function **thread_begin()** to **pthread_create()** to start run two different threads(cores). In the following statement, as I use the same program, to distinguish different threads(cores), I'll use **pthread_self()** to get current thread tid and compare to do this(the implementation is simple and I'll not show all code here).

1.2.2 How to deal with shared space while two threads is running?

As we can see, thread 0(core 0) and thread 1(core 1) only share memory and l3cache, which means I may only concern about memory access part in simulator.
Following is some code for memory access to base bound address space in **Simulator.cpp** before memory read and memory write

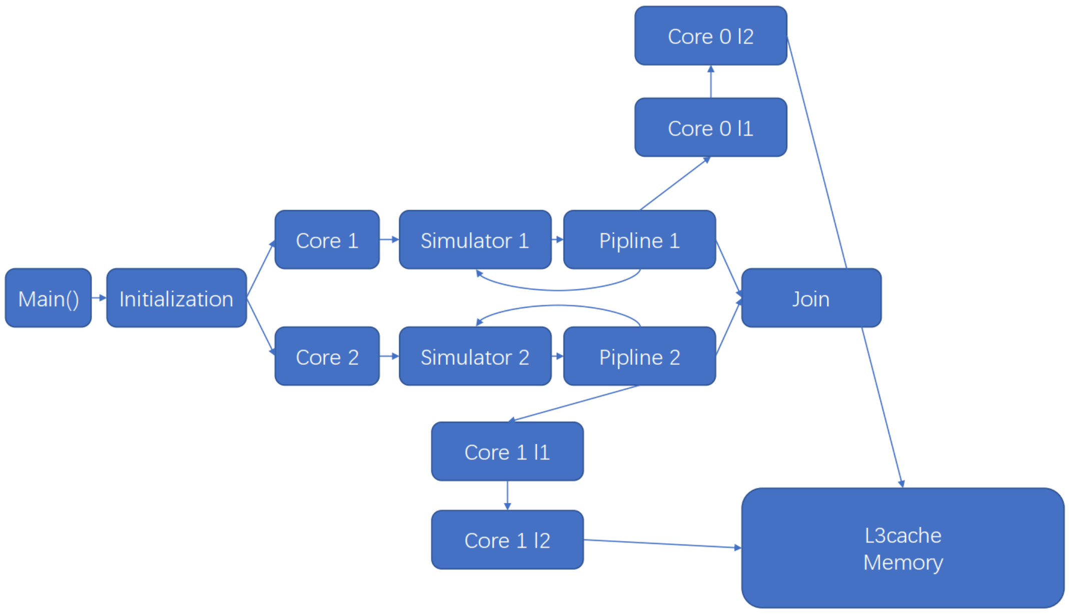
```
if (this->core)
{/* this->core is store to distinguish between two threads */
  if (out <= 0x7fc00000 && out > 0x7f800000)
  {
    /* Stack space do nothing, out is the previous address */
  }
  else
  {
    out += base1;
  }
}
```

```
else
{
    if (out <= 0x80000000 && out > 0x7fc00000)
    {
        /* Stack space do nothing */
    }
    else
    {
        out += base0;
    }
}
```

You should notice that the same code will also occur in systemcall handler.
I also modified some functions input in class **MemoryManager** but just for thread information passing, it is simple and easy to understand, I'll not list here.

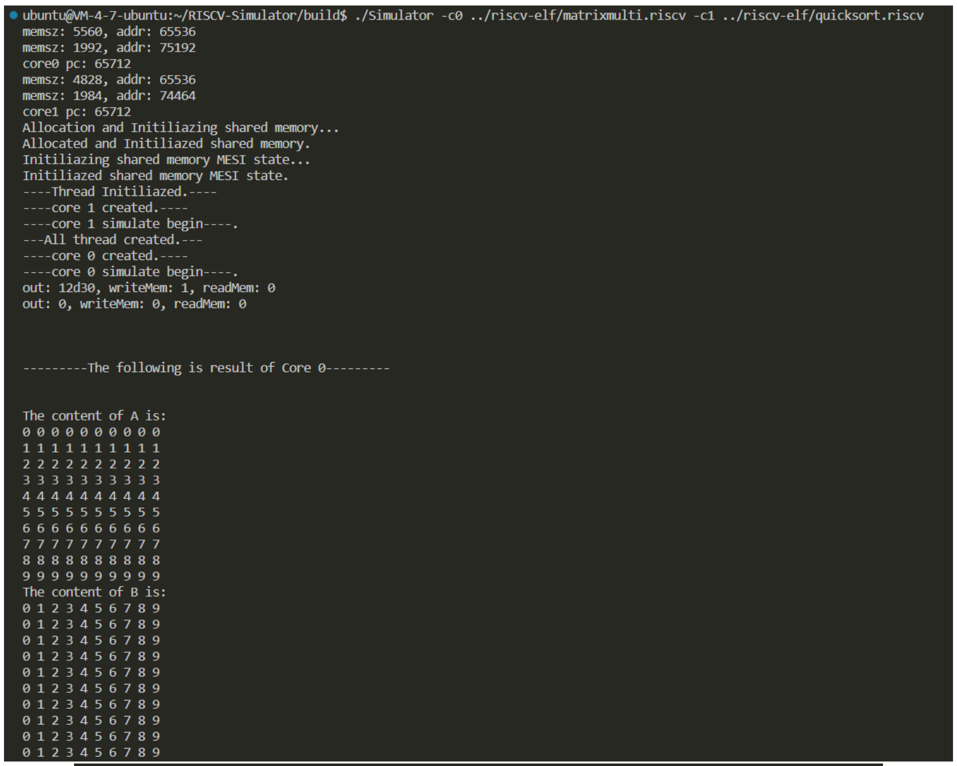
1.2.3 How to end two threads?

As we can see, the two threads(cores) run two different simulators, and when they finished(call the system-exit-call), it will run function (pthread_exit()). And in **main()**, two join fuction will wait for two threads' ending. As I mentioned in preknowledge before, the **main()** will then print two format string out and free all resources.

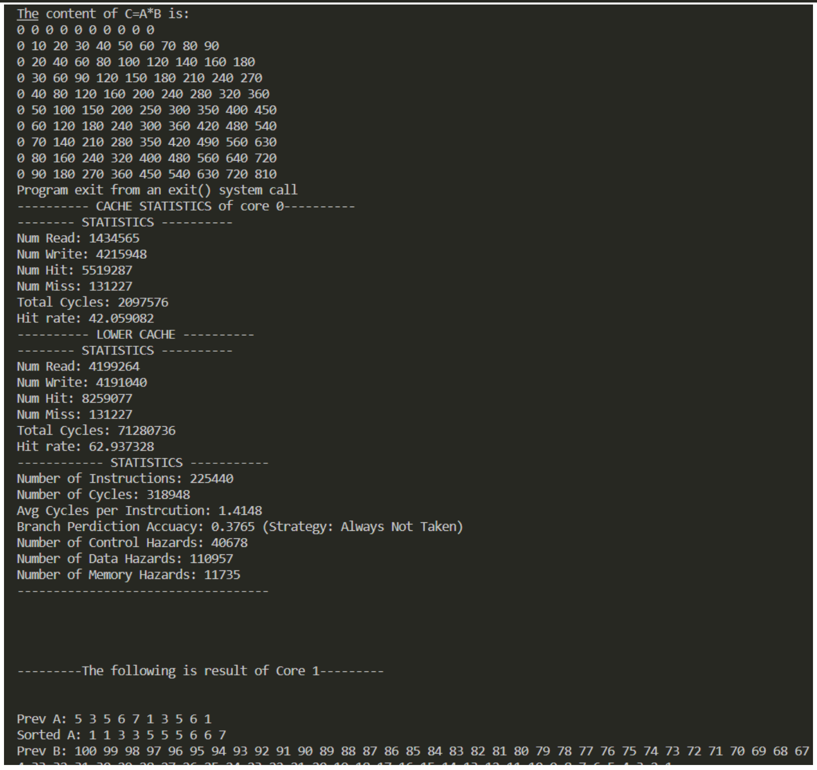


1.3 Test and Correctness

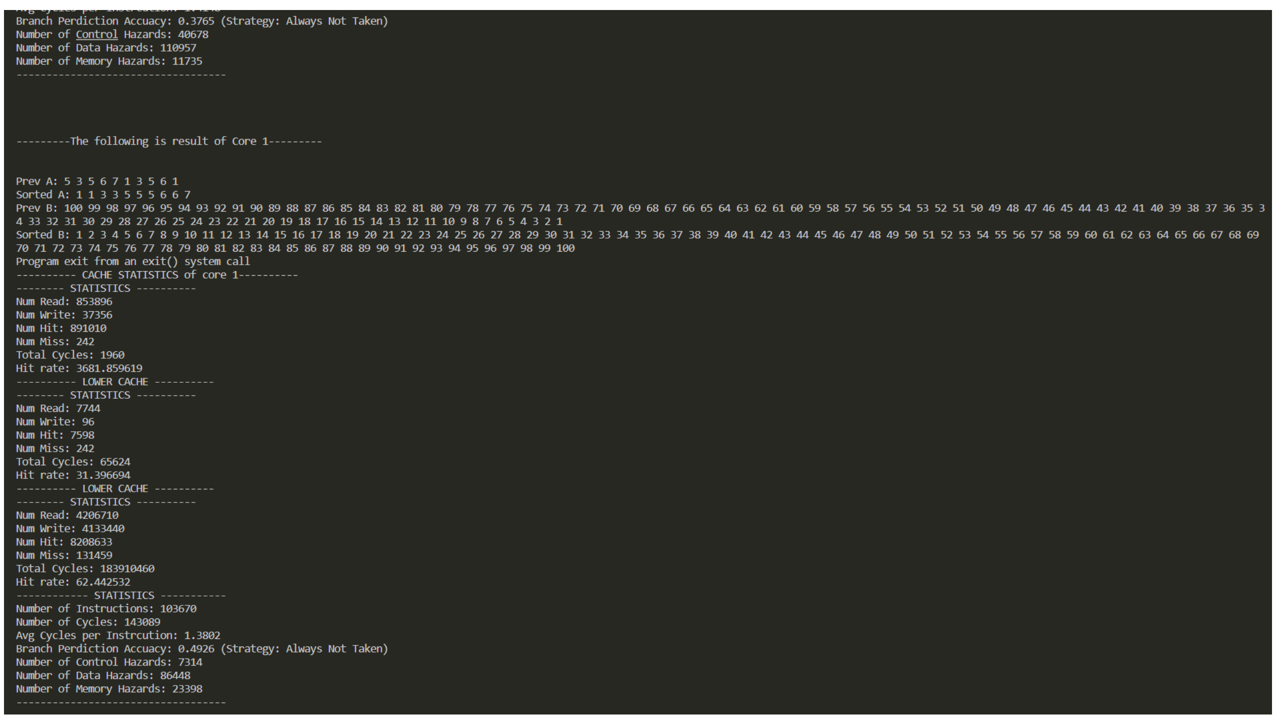
To test task1, you should input format string at ternimal which I mentioned in preknowledge before. It is simply to test whether it is correct since the platform has already offer some .riscv file. And I'll put several results as pictures bellow, you can also choose your test file.
Notice, since the output is too long, I part it into several pictures.
This is test for **matrixmulti.riscv** and **quicksort.riscv**:



```
ubuntu@VM-4-7-ubuntu:~/RISCV-Simulator/build$ ./Simulator -c0 ../riscv-elf/matrixmulti.riscv -c1 ../riscv-elf/quicksort.riscv
memsz: 5560, addr: 65536
memsz: 1992, addr: 75192
core0 pc: 65712
memsz: 4828, addr: 65536
memsz: 1984, addr: 74464
core1 pc: 65712
Allocation and Initiliazing shared memory...
Allocated and Initiliazed shared memory.
Initiliazing shared memory MESI state...
Initiliazed shared memory MESI state.
----Thread Initiliazed.----
----core 1 created.----
----core 1 simulate begin----.
---All thread created.---
----core 0 created.----
----core 0 simulate begin----.
out: 12d30, writeMem: 1, readMem: 0
out: 0, writeMem: 0, readMem: 0



---------The following is result of Core 0---------


The content of A is:
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9
The content of B is:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

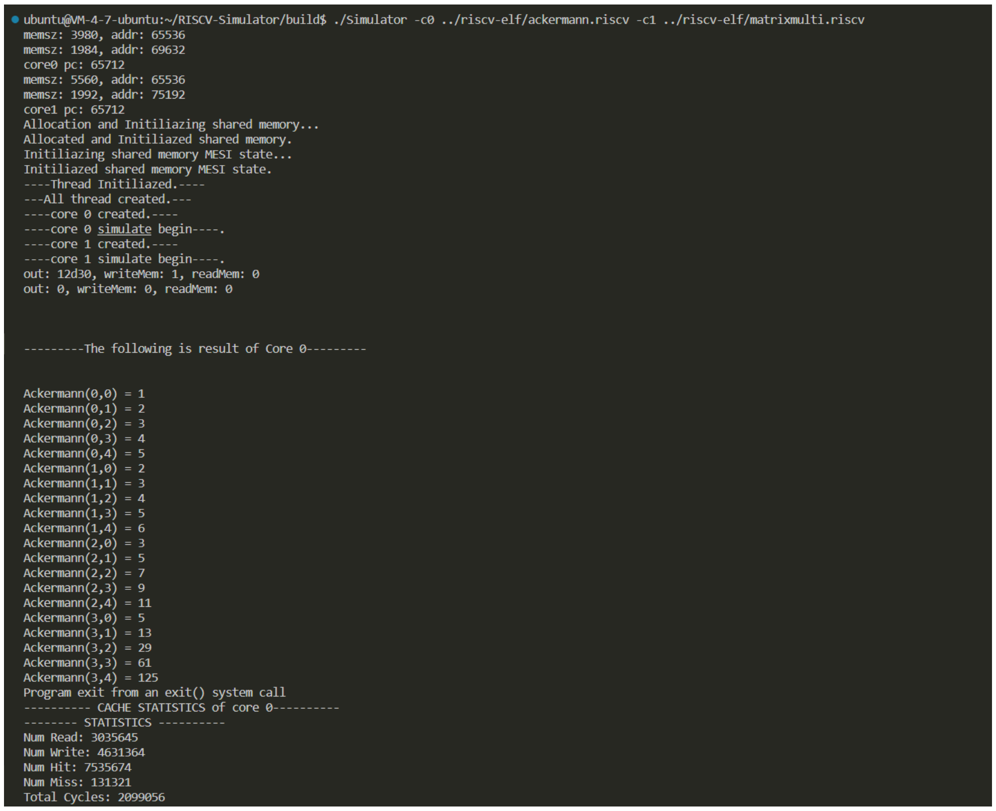


```
The content of C=A*B is:
0 0 0 0 0 0 0 0 0 0
0 10 20 30 40 50 60 70 80 90
0 20 40 60 80 100 120 140 160 180
0 30 60 90 120 150 180 210 240 270
0 40 80 120 160 200 240 280 320 360
0 50 100 150 200 250 300 350 400 450
0 60 120 180 240 300 360 420 480 540
0 70 140 210 280 350 420 490 560 630
0 80 160 240 320 400 480 560 640 720
0 90 180 270 360 450 540 630 720 810
Program exit from an exit() system call
---------- CACHE STATISTICS of core 0----------
-------- STATISTICS ----------
Num Read: 1434565
Num Write: 4215948
Num Hit: 5519287
Num Miss: 131227
Total Cycles: 2097576
Hit rate: 42.059082
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4199264
Num Write: 4191040
Num Hit: 8259077
Num Miss: 131227
Total Cycles: 71280736
Hit rate: 62.937328
------------ STATISTICS -----------
Number of Instructions: 225440
Number of Cycles: 318948
Avg Cycles per Instrcution: 1.4148
Branch Perdiction Accuacy: 0.3765 (Strategy: Always Not Taken)
Number of Control Hazards: 40678
Number of Data Hazards: 110957
Number of Memory Hazards: 11735
-----------------------------------




---------The following is result of Core 1---------


Prev A: 5 3 5 6 7 1 3 5 6 1
Sorted A: 1 1 3 3 5 5 5 6 6 7
Prev B: 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67
```

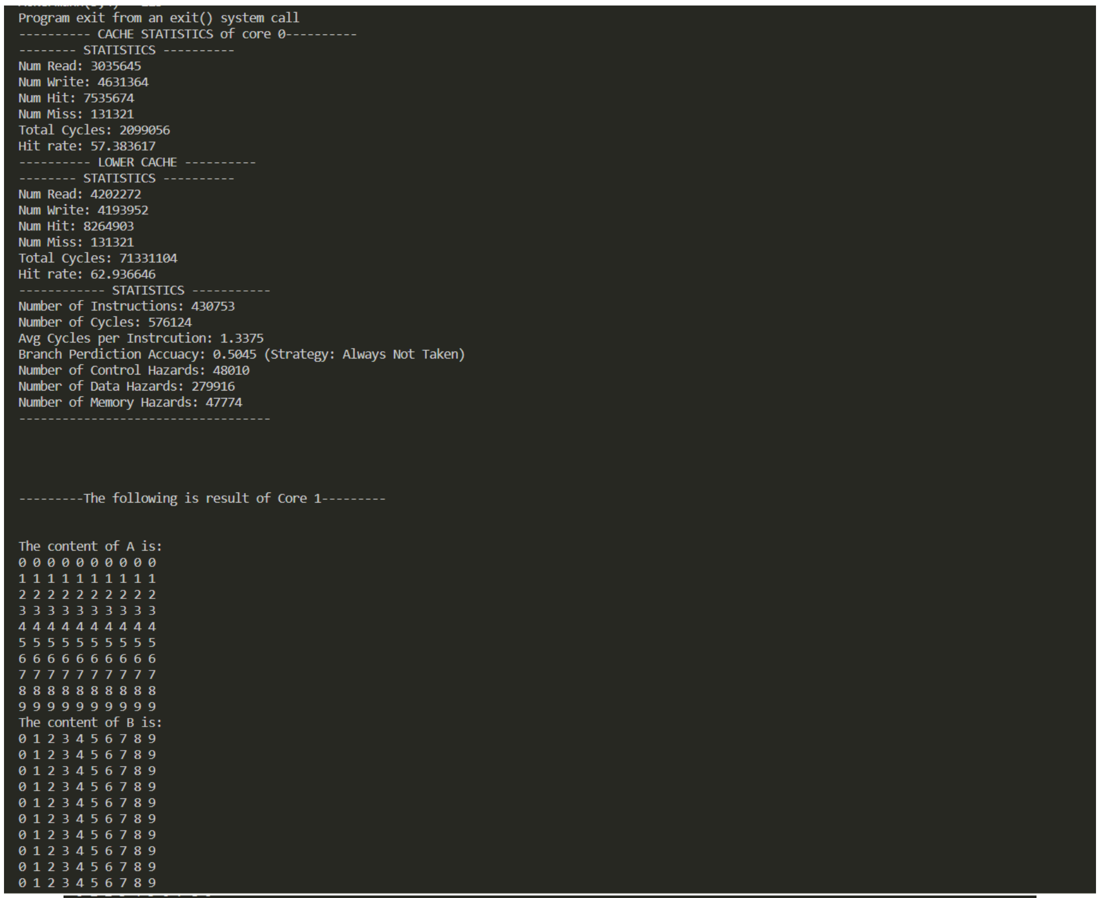


```
Branch Perdiction Accuacy: 0.3765 (Strategy: Always Not Taken)
Number of Control Hazards: 40678
Number of Data Hazards: 110957
Number of Memory Hazards: 11735
----------------------------------




---------The following is result of Core 1---------


Prev A: 5 3 5 6 7 1 3 5 6 1
Sorted A: 1 1 3 3 5 5 5 6 6 7
Prev B: 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 3
4 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Sorted B: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
Program exit from an exit() system call
---------- CACHE STATISTICS of core 1----------
-------- STATISTICS ----------
Num Read: 853896
Num Write: 37356
Num Hit: 891010
Num Miss: 242
Total Cycles: 1960
Hit rate: 3681.859619
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 7744
Num Write: 96
Num Hit: 7598
Num Miss: 242
Total Cycles: 65624
Hit rate: 31.396694
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4206710
Num Write: 4133440
Num Hit: 8208633
Num Miss: 131459
Total Cycles: 183910460
Hit rate: 62.442532
------------ STATISTICS -----------
Number of Instructions: 103670
Number of Cycles: 143089
Avg Cycles per Instrcution: 1.3802
Branch Perdiction Accuacy: 0.4926 (Strategy: Always Not Taken)
Number of Control Hazards: 7314
Number of Data Hazards: 86448
Number of Memory Hazards: 23398
----------------------------------
```

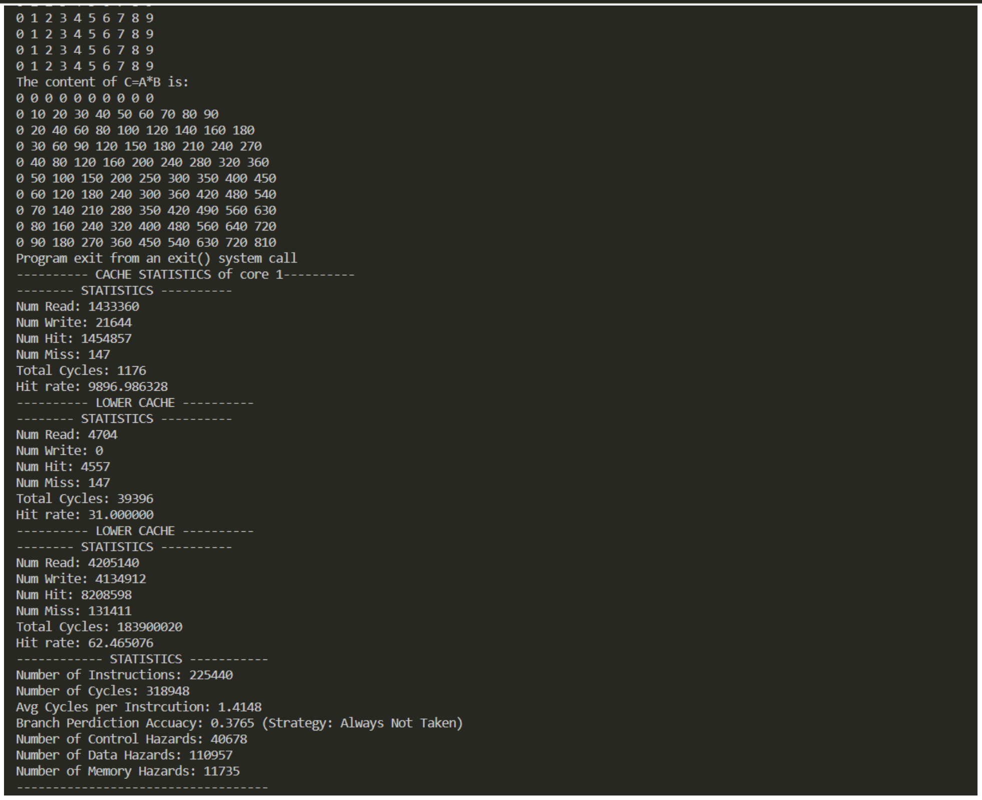
This is test for **ackermann.riscv** and **matrixmulti.riscv**:



```
ubuntu@VM-4-7-ubuntu:~/RISCV-Simulator/build$ ./Simulator -c0 ../riscv-elf/ackermann.riscv -c1 ../riscv-elf/matrixmulti.riscv
memsz: 3980, addr: 65536
memsz: 1984, addr: 69632
core0 pc: 65712
memsz: 5560, addr: 65536
memsz: 1992, addr: 75192
core1 pc: 65712
Allocation and Initiliazing shared memory...
Allocated and Initiliazed shared memory.
Initiliazing shared memory MESI state...
Initiliazed shared memory MESI state.
----Thread Initiliazed.----
---All thread created.---
----core 0 created.----
----core 0 simulate begin----.
----core 1 created.----
----core 1 simulate begin----.
out: 12d30, writeMem: 1, readMem: 0
out: 0, writeMem: 0, readMem: 0



---------The following is result of Core 0---------


Ackermann(0,0) = 1
Ackermann(0,1) = 2
Ackermann(0,2) = 3
Ackermann(0,3) = 4
Ackermann(0,4) = 5
Ackermann(1,0) = 2
Ackermann(1,1) = 3
Ackermann(1,2) = 4
Ackermann(1,3) = 5
Ackermann(1,4) = 6
Ackermann(2,0) = 3
Ackermann(2,1) = 5
Ackermann(2,2) = 7
Ackermann(2,3) = 9
Ackermann(2,4) = 11
Ackermann(3,0) = 5
Ackermann(3,1) = 13
Ackermann(3,2) = 29
Ackermann(3,3) = 61
Ackermann(3,4) = 125
Program exit from an exit() system call
---------- CACHE STATISTICS of core 0----------
-------- STATISTICS ----------
Num Read: 3035645
Num Write: 4631364
Num Hit: 7535674
Num Miss: 131321
Total Cycles: 2099056
```



```
Program exit from an exit() system call
---------- CACHE STATISTICS of core 0----------
-------- STATISTICS ----------
Num Read: 3035645
Num Write: 4631364
Num Hit: 7535674
Num Miss: 131321
Total Cycles: 2099056
Hit rate: 57.383617
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4202272
Num Write: 4193952
Num Hit: 8264903
Num Miss: 131321
Total Cycles: 71331104
Hit rate: 62.936646
------------ STATISTICS -----------
Number of Instructions: 430753
Number of Cycles: 576124
Avg Cycles per Instrcution: 1.3375
Branch Perdiction Accuacy: 0.5045 (Strategy: Always Not Taken)
Number of Control Hazards: 48010
Number of Data Hazards: 279916
Number of Memory Hazards: 47774
-----------------------------------



---------The following is result of Core 1---------


The content of A is:
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9
The content of B is:
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```



```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
The content of C=A*B is:
0 0 0 0 0 0 0 0 0 0
0 10 20 30 40 50 60 70 80 90
0 20 40 60 80 100 120 140 160 180
0 30 60 90 120 150 180 210 240 270
0 40 80 120 160 200 240 280 320 360
0 50 100 150 200 250 300 350 400 450
0 60 120 180 240 300 360 420 480 540
0 70 140 210 280 350 420 490 560 630
0 80 160 240 320 400 480 560 640 720
0 90 180 270 360 450 540 630 720 810
Program exit from an exit() system call
---------- CACHE STATISTICS of core 1----------
-------- STATISTICS ----------
Num Read: 1433360
Num Write: 21644
Num Hit: 1454857
Num Miss: 147
Total Cycles: 1176
Hit rate: 9896.986328
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4704
Num Write: 0
Num Hit: 4557
Num Miss: 147
Total Cycles: 39396
Hit rate: 31.000000
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4205140
Num Write: 4134912
Num Hit: 8208598
Num Miss: 131411
Total Cycles: 183900020
Hit rate: 62.465076
------------ STATISTICS -----------
Number of Instructions: 225440
Number of Cycles: 318948
Avg Cycles per Instrcution: 1.4148
Branch Perdiction Accuacy: 0.3765 (Strategy: Always Not Taken)
Number of Control Hazards: 40678
Number of Data Hazards: 110957
Number of Memory Hazards: 11735
-----------------------------------
```

As I mentioned before, the reason why the output is so ”beautiful” is because I hold the output of different

thread into specified string buffer, it will look very ugly without this method multi thread ouput :(.
We can see that the program run currently and print current result.

2 Task2

2.1 Requirement

- Realize MESI based on directories.
- Running two official test programs, add false sharing tests, verify implementation
- Bonus for same cache line same cycle memory operation.

2.2 Implementation

2.2.1 Shared Memory

Since the test programs ask for 4MB shared memory from $0x100000$ to $0x500000$, I choose this space as default shared memory space and allocate it at the begining of this simulator, the address in this area will not influenced by B&B policy and will handled by MESI directory. All other memory space will keep the same as no MESI policy.
Each time when we try to operate on shared memory, we will first try to get a lock **l3cache_mainmemory_lock** which declared in **MainCPU.cpp** and externed in **Cache.cpp** to avoid operate on the same cache, though it is inefficient , can fullfill complex multithread running environment well.

2.2.2 MESI

We first affirm that it is private l1, l2 cache, shared l3 cache. The policy is write back and write allocate.
Normally, directory based on MESI are like this, cache line has 4 status: **C-invalid**, **C-shared**, **C-modified**, **C-transient**, memory line has 4 status: **R(dir)**, **W(id)**, **TR(dir)**, **TW(id)**. However, in software implementation, memory operation should finished all together in one instruction, and it should have **E** state. To simplify, each level of each core cache will have a bit to store it's state, including shared l3 cache, also memory will have a bit to store its state. So I finally set my MESI directory as follows

```
/* MESI cacheline state */
enum cache_state
{
  CM = 0, /* Modified, should be the only one among all cache, memory */
  CE = 1, /* Exclusive, only owned by one core */
  CS = 2, /* Shared, have multiple copy among different core, memory */
  CI = 3, /* Invalid */
  CN = 4, /* Initialize State, for debug, it should be same as CI */
};

/* MESI memory line state */
enum memory_state
{
  MS = 0, /* Shared, valid data */
  MI = 1, /* Invalid */

/* MESI directory */
cache_state share_memory_cache[SHARE_CACHELINE_SIZE][2][3];
memory_state share_memory_memory[SHARE_CACHELINE_SIZE];
};
```

For initilization, all shared memory line are set to **MS**, all shared cache line are set to **CN**.
For shared memory space read:(suppose current is core 0)

- If core 0 cache hit, then all directory state keep the same.
- If core 0 cache miss, enter into memory, see for directory.
  - If memory data state is MS(shared), read data back.
    - If the data is only shared in memory, then core 0 l1, l2, l3 are all CE.
    - If the data has other valid copy in core 1, then core 0 l1, l2, l3 are all CS.
  - If memory data state is MI(in another core), find the data in another core(it is sure that the first hit in another core is the right copy), read back to memory, set another core data state to be CS, set memory state to MS. Read back to core 0, and set state to CS.

For shared memory space write:(suppose current is core 0), since it is write allocate, we first make sure that the right copy should appear in core 0 l1 cache.

- If core 0 l1 cache has valid copy, write into it, and see other state
  - If previous core 0 l1 cache state is CE, set it to CM, invalid any valid copy in core 0 space(core 0 l2, shared l3, memory) if they have, set their cache line valid bit to invalid, and state to MI
  - If previous core 0 l1 cache state is CS, set it to CM, invalid any valid copy in other space(core 0 l2, core 1 l1, l2, shared l3, memory) if they have, set their cache line valid bit to invalid, and state to MI
  - If previous core 0 l1 cache state is CM, do nothing.
- If core 0 l1 cache do not have valid copy, first read it, make sure that it has a valid copy in l1 cache, and see other state
  - If previous core 0 l1 cache state is CE, set it to CM, invalid any valid copy in core 0 space(core 0 l2, shared l3, memory) if they have, set their cache line valid bit to invalid, and state to MI
  - If previous core 0 l1 cache state is CS, set it to CM, invalid any valid copy in other space(core 0 l2, core 1 l1, l2, shared l3, memory) if they have, set their cache line valid bit to invalid, and state to MI
  - If previous core 0 l1 cache state is CM, do nothing.

For evicted policy, we know that a shared data need to be additional concerned if and only if its state is CM(since unmodified data copy wont write back)

- If evit from shared l3 cache to memory, delete the state in l3 cache, write data back to memory, set memory state to MS.
- If evit from private cache, delete the state in the previous level cache, update in the next level cache.

The real implementation is quite complex and it is impossible to list here, I nearly add 2000 lines of code to write and test this. The comments is readable and please directly read the code.

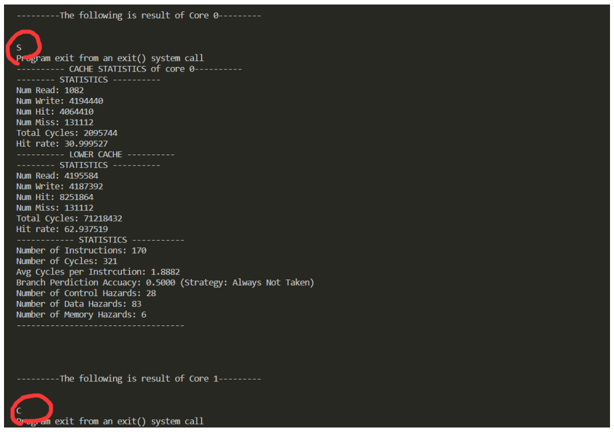
2.2.3 Additional Consider

First there is the same situation with Lab1, the platform previous load or store data per byte(e.t. read a int will cause 4 read of bytes), more seriously, this will cause if one byte of data is write or read, it will change the cache/memory MESI state and then remain bytes of data will get a different state situation. To solve this, I support **CM** state data update all block directly back to memory. Thus, when the read int request to invalid memory is received, the first byte will update memory with the right core copy, and then the remain byte will get a valid memory copy, which do not harm to data correctness.
What's more, a write through cache will be more simple and easier to implemented, it actually used to be my choice but more inefficiency. So I finally to choose write allocate.

2.3 Test and Correctness

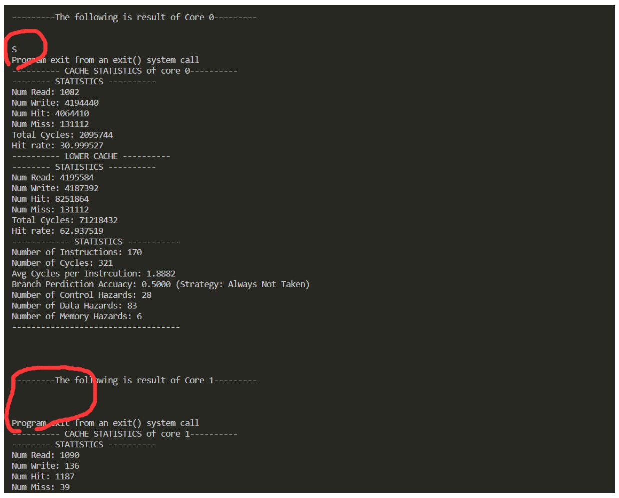
2.3.1 True Sharing

I directly run the official code, since the cache line size is 32. First we test the correctness, I run several times with following command **./Simulator -c0 ../riscv-elf/lab3-core0.riscv -c1 ../riscv-elf/lab3-core1.riscv**, it have and should have three kinds of output:
core 0: S core 1: C



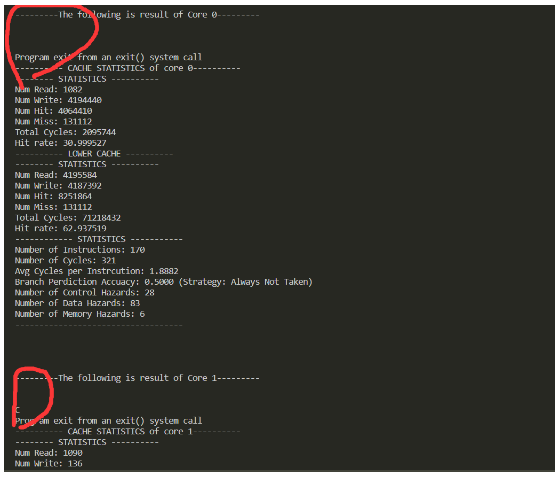
```
---------The following is result of Core 0---------

S
Program exit from an exit() system call
---------- CACHE STATISTICS of core 0----------
-------- STATISTICS ----------
Num Read: 1082
Num Write: 4194440
Num Hit: 4064410
Num Miss: 131112
Total Cycles: 2095744
Hit rate: 30.999527
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4195584
Num Write: 4187392
Num Hit: 8251864
Num Miss: 131112
Total Cycles: 71218432
Hit rate: 62.937519
------------ STATISTICS -----------
Number of Instructions: 170
Number of Cycles: 321
Avg Cycles per Instrcution: 1.8882
Branch Perdiction Accuacy: 0.5000 (Strategy: Always Not Taken)
Number of Control Hazards: 28
Number of Data Hazards: 83
Number of Memory Hazards: 6
-----------------------------------



---------The following is result of Core 1---------

C
Program exit from an exit() system call
```

core 0: S core 1:(nothing)



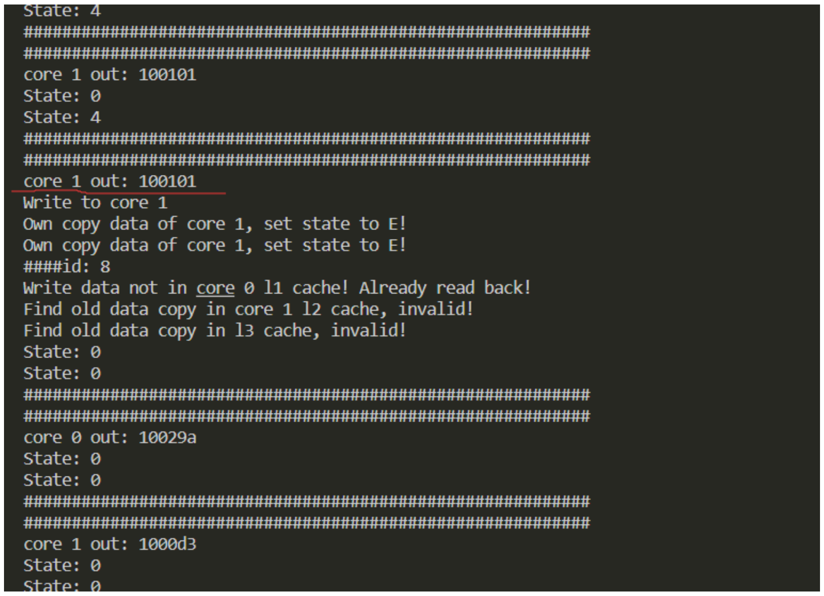
```
---------The following is result of Core 0---------

S
Program exit from an exit() system call
---------- CACHE STATISTICS of core 0----------
-------- STATISTICS ----------
Num Read: 1082
Num Write: 4194440
Num Hit: 4064410
Num Miss: 131112
Total Cycles: 2095744
Hit rate: 30.999527
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4195584
Num Write: 4187392
Num Hit: 8251864
Num Miss: 131112
Total Cycles: 71218432
Hit rate: 62.937519
------------ STATISTICS -----------
Number of Instructions: 170
Number of Cycles: 321
Avg Cycles per Instrcution: 1.8882
Branch Perdiction Accuacy: 0.5000 (Strategy: Always Not Taken)
Number of Control Hazards: 28
Number of Data Hazards: 83
Number of Memory Hazards: 6
-----------------------------------



---------The following is result of Core 1---------


Program exit from an exit() system call
---------- CACHE STATISTICS of core 1----------
-------- STATISTICS ----------
Num Read: 1090
Num Write: 136
Num Hit: 1187
Num Miss: 39
```

core 0:(nothing) core 1:C



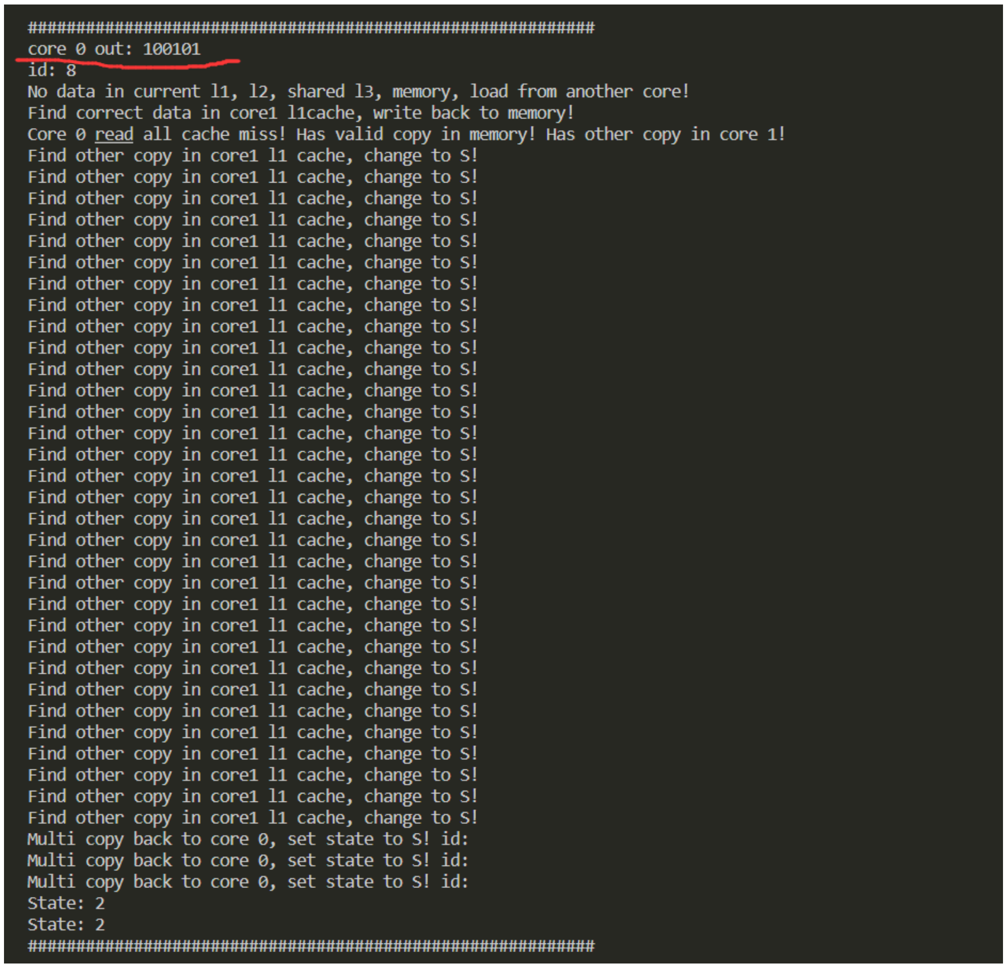
```
---------The following is result of Core 0---------



Program exit from an exit() system call
---------- CACHE STATISTICS of core 0----------
-------- STATISTICS ----------
Num Read: 1082
Num Write: 4194440
Num Hit: 4064410
Num Miss: 131112
Total Cycles: 2095744
Hit rate: 30.999527
---------- LOWER CACHE ----------
-------- STATISTICS ----------
Num Read: 4195584
Num Write: 4187392
Num Hit: 8251864
Num Miss: 131112
Total Cycles: 71218432
Hit rate: 62.937519
------------ STATISTICS -----------
Number of Instructions: 170
Number of Cycles: 321
Avg Cycles per Instrcution: 1.8882
Branch Perdiction Accuacy: 0.5000 (Strategy: Always Not Taken)
Number of Control Hazards: 28
Number of Data Hazards: 83
Number of Memory Hazards: 6
----------------------------------




---------The following is result of Core 1---------


C
Program exit from an exit() system call
---------- CACHE STATISTICS of core 1----------
-------- STATISTICS ----------
Num Read: 1090
Num Write: 136
```

Next, we use the debug ouput to make sure that it is true sharing. For example, suppose two core all write data before each other read, **lab3-core1.c** modified a[257], thus when **lab3-core0.c** read a[257], it will find invalid in memory and copy data back from core 1.
Here is what **lab3-core1.c** write a[257] happen:



```
State: 4
##########################################################
##########################################################
core 1 out: 100101
State: 0
State: 4
##########################################################
##########################################################
core 1 out: 100101
Write to core 1
Own copy data of core 1, set state to E!
Own copy data of core 1, set state to E!
####id: 8
Write data not in core 0 l1 cache! Already read back!
Find old data copy in core 1 l2 cache, invalid!
Find old data copy in l3 cache, invalid!
State: 0
State: 0
##########################################################
##########################################################
core 0 out: 10029a
State: 0
State: 0
##########################################################
##########################################################
core 1 out: 1000d3
State: 0
State: 0
```

We can see that when core 1 write, core 1 has valid copy in core 1 l1 cache, and the copy in core 1 l2 cache, shared l3 cache, memory are all invalid. Then, when core 0 try to read it:

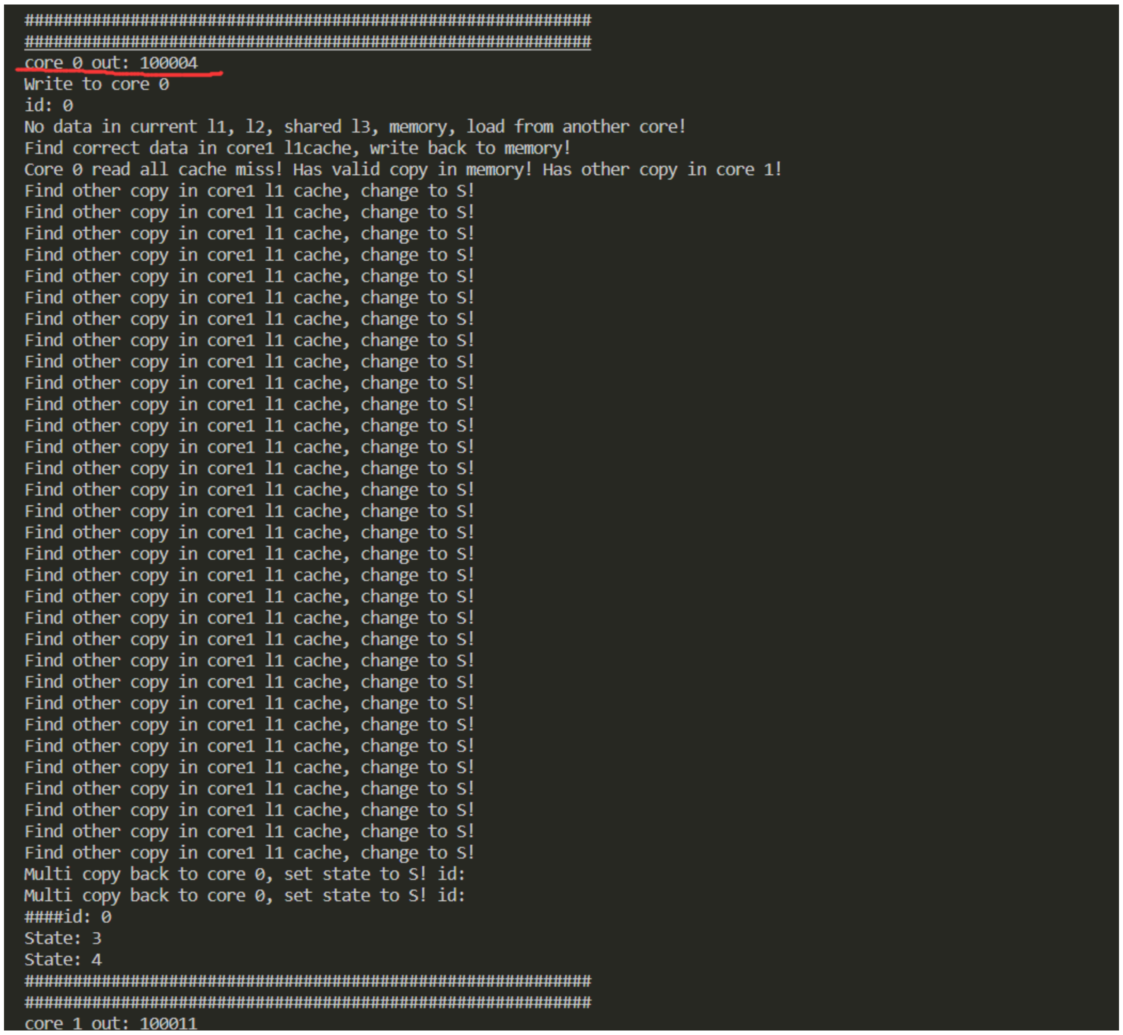


```
############################################################
core 0 out: 100101
id: 8
No data in current l1, l2, shared l3, memory, load from another core!
Find correct data in core1 l1cache, write back to memory!
Core 0 read all cache miss! Has valid copy in memory! Has other copy in core 1!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Multi copy back to core 0, set state to S! id:
Multi copy back to core 0, set state to S! id:
Multi copy back to core 0, set state to S! id:
State: 2
State: 2
############################################################
```

We can see that when core 0 read, it find that the copy in memory is invalid, the request let core 1 write data back to memory. Then core 0 read it back and set state to S. Thus, we proved that the true sharing situation is correct.

2.3.2 Fasle Sharing

I run two programs write by myself. **lab3-core0-test1.c** multiple write to array **a** from index 0 to index 9, and **lab3-core1-test1.c** multiple write to array **a** from index 10 to index 19(this two programs and corresponding .riscv can be found in ./test and ./riscv-elf). We know that they do not write each other's memory space, after the first write of each other, they should not occur any write miss. Since the multi thread might execute dynamic code in different oder, so here is just a example show that they actually suffer from write miss,



```
###########################################################
###########################################################
core 0 out: 100004
Write to core 0
id: 0
No data in current l1, l2, shared l3, memory, load from another core!
Find correct data in core1 l1cache, write back to memory!
Core 0 read all cache miss! Has valid copy in memory! Has other copy in core 1!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Find other copy in core1 l1 cache, change to S!
Multi copy back to core 0, set state to S! id:
Multi copy back to core 0, set state to S! id:
####id: 0
State: 3
State: 4
###########################################################
###########################################################
core 1 out: 100011
```

As show in the picture, when core 0 try to write a[4], it suffer a write miss which caused by core 1 right, since all the write are in a same cache line, which is called a false sharing. Thus, proved that false sharing is right.

2.4 How to deal with writing to same cache line conflict?

It is quite hard to implemented a totally parallel program that run on two cores, the cache structure is quite urgly. As you can see before, for code which not used shared memory, I allow them to execute paralleled, for code which used shared memory, I'll check and let them execute in sequential.

3 Something Want to Tell

Dear TA,
Hello,
This is a note that when I finished all labs I want to leave to you, this RISCV-Simulator is actually not a very good simulator platform.
There are some reasons,
First, the cache hit rate calculation is wrong, since the author implement cache as once a time load a byte no matter it is a int request or a long int request, which results the first byte read the whole cache line into

cache, and the remaining bytes are all cache hit! Thus, for example, when I once read a int data, I got only one cache miss and three cache hit! Is this real in life to calulate a cache hit rate?
Second, also about the cache, the author use highly resursion structure to implement multi level cache. It violate engineering code specification, since its compatibility is quite low and easy to broken down when try to modified it. Also, it is quite strange about its program flow design, the class MemoryManager both handle cache and memory(but actually author design to let MemoryManager just be main memory). That means student have to pay a lot time to understand how the author implement simple concept taught in class and try to obey its strange design. This semester I also choose Operating System, the project is writing a naive system called PintOS, I feel there is indeed a significant gap in engineering quality between the two source codes.
Third, actually when I talked with the author of this platfrom(I pose a PR to fix a bug of the Simulator before), it is just a curriculum design and there might exist several bugs or even design discount. I'm also wondering if there is a better Simulator you can use in the next few years.
I understand the lab is aim to practise what we learned in class, but I just feel that I spend a little more time on something not related with this target. There might not exist a better one, or I guess Prof. Wang are aim to training our ability of reading and modifying "Shit Mountain". Anyway, it just something I want to tell you, and I hope that the Simulator will be improved in the next few years. This course is a nice course, and teach me a lesson that Prof. Wang has said many times, "Simple, but not easy."