

# 实验三（选题一）

---

2021年6月20日，计83李天勤2018080106

## 现代处理器内存消歧和存取优化模拟实验

---

Modern Processor Disambiguation and Access Optimization Simulation Experiment

### 实验目的

---

This experiment is split into two parts

1. realize microarchitecture component that allows data bypass, simulating the behavior of each component in transient execution to help us understand the composition and function of the memory subsystem of modern processors as well as understand the predictive optimization of modern processors for memory access data prediction.
2. use simulator to realize SpectreV4's POC, verifying the correctness of the first part

### 实验内容

---

The memory subsystem main includes `LSQ`, `TLB`, `PageTable`, `Cache`, `PhysicalMemory`. After the components of the memory subsystem are correctly implemented, prediction and optimization can be performed for different memory access sequences. Prediction optimization is implemented in the transient window, instruction is sent to `Loadstore queue`, but the result has not been submitted in the time window.

Split into four subtasks:

1. Task 1 implement TLB and page table access operations, as well as `Cache` and `PhysicalMemory` access operations
2. Task 2 implement `LSQ` scheduling that allows out of order execution of memory operations on the basis of task one, implement `Store-to-Load` forward in `out-of-order` execution, and perform a given memory access sequence test to evaluate the `TLB miss rate` and `Cache miss rate` of our simulator
3. Task 3 optimize these memory access
4. Task 4 give tests cases from a designer's perspective to verify the correctness the implementation

### 实验设计

---

The first task was to understand the overall components and composition of the architecture given to us, based on the definitions declared in the file `define.h`. Then we complete the cache related operations in the file `memory.cpp`. Specifically, the load and store instructions that accesses the cache. If the access is a hit, the cache will be updated and the corresponding data will be returned. At the same time, we have to update flags. Next, we have to complete the operations related to the TLB located in the file `tlb.cpp`, mainly the operations to be performed when querying the TLB and updating the TLB. We also have to complete the core logic located in the file `controller.h`, we have to implement the LSQ scheduling that allows for memory to be performed out of order. The overall function simulates a pipeline, with each clock cycle checked

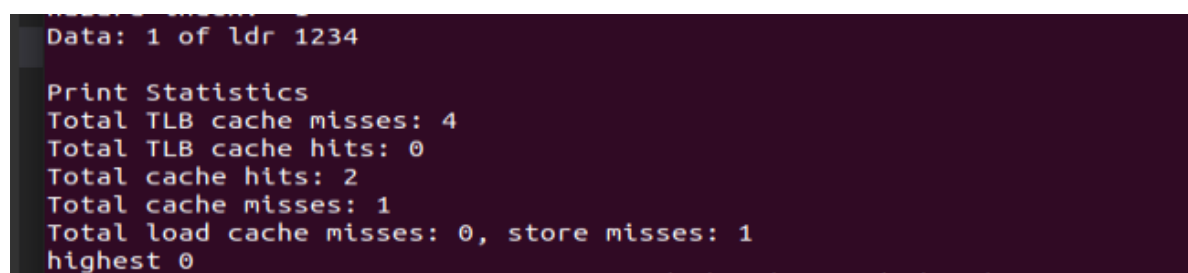
using the function `check_1sq()`. The basic logic is: in each stage, an instruction will enter the next stage after the execution of the stage is completed. The instruction processing process is divided into 6 stages, arrive, dispatch, translation, data, and retire, which correspond to the 5 clocks dictated by the `que_t` structure. We also have to slightly change `main.cpp` to output the correct statistics and to complete the reload function.

## 测试用例的说明

This is the test case that I used.

```
flush
str 0x0 0x1234
str 0x1 [0x1234]
ldr 0x1234
```

Running `./main ./trace/test0.out`



```
Data: 1 of ldr 1234
Print Statistics
Total TLB cache misses: 4
Total TLB cache hits: 0
Total cache hits: 2
Total cache misses: 1
Total load cache misses: 0, store misses: 1
highest 0
```

We can see that the `ldr` instruction is able to correctly read the data located in the second store instruction. It shows that because the first instruction is located in the `quick store`, thus the `load` instruction will bypass the first instruction, but perform flush, and rerun after the second instruction is completed, then giving us the correct result. This verifies that the data fetched by the load instruction is indeed correct and that the stored data is written to memory correctly.

## 问答题

请从以下几个问题中选择三个感兴趣的问题回答

1. 在本实验中仅仅针对RAW hazard进行了内存消歧，我们是否需要针对RAW，WAW，WAR进行消歧？如果需要请给出设计思路，并且使用简单样例来验证你的加速效果；否则给出不需要的理由。

【解答】 In this experiment, memory disambiguation was performed on RAW hazard. After each store instruction is committed, each load instruction that is connected to the bypass is checked. And if the connection is not correct, mark a rerun, so that it passes forward and flush-rerun. The mechanism ensures the correctness of the read-after-write instructions. However, the problems of WAW and WAR will not occur because the scheduling mechanism we have implemented requires that only the earliest instructions can enter the retire state. So there is no need for disambiguation.

2. 在瞬态窗口中的操作原则是不能影响宏观结构上的组件，避免瞬态数据的泄露。但是在实验二中我们依旧使用侧信道将数据提取出来了，请分析原因。

【解答】 The reason for that we use the side channel is that the load command calls `LoadFromCache` in the transient window to read data from the cache, and updates the cache when it misses, thereby using the side channel to obtain the data

3. 请结合你的模拟器的实现，分析要保证瞬态执行结果不被系统程序员获取应该做出怎样的限制。

【解答】 Restrict the load instruction to be the same as the store instruction, and only actually access the cache to read the data in the retire phase

4. 除了实验二中的SpectreV4，我们是否有其他的利用行为？可以结合通用寄存器，内存，外设等组件进行联想。
5. 给定一个平台（或处理器），应该如何挖掘平台上的瞬态执行漏洞？给出你的设计。
6. 给定一个平台（或处理器），应该如何验证平台上是否存在，以及存在何种瞬态执行漏洞？给出你的设计。
7. 如何从一个设计者的视角验证你的设计的正确性？给出你的验证点以及评价指标

## 对实验的意见与建议

---

This experiment was a nice summation to many topics that we have covered during this course. The framework given was relatively easy to compete and complete, and the guide was informative and easy to read.