CS211 Lab4 Report

Name: Feng Xiao
ID Number: 2023234296

2023-12-27

1 Lab content

In Lab1, we implemented the scoreboard algorithm to enable our simulator to execute the program out of order. However, because its instructions are issued sequentially, it cannot maximize the program running performance. After completing lab1, I thought, it is a pity not to implement a simulator for out-of-sequence emission. Maybe my idea was heard by the gods, and Lab4 appeared.

1.1 Tomasulo Algorithm

Tomasulo is a computer hardware architecture algorithm for dynamically scheduling instructions, allowing out-of-order execution and more efficient use of multiple execution units. It was proposed by IBM in 1967, and its first application was on IBM system 360. The biggest feature of Tomasulo is that it eliminates the risk of false data by using the idea of renaming, thereby improving the out-of-order performance of the machine.

1.2 Re-Order Buffer

The reorder buffer (ROB) holds the result of an instruction between the time when the operation associated with the instruction completes and the time when the instruction commits. The ROB therefore is a source of operands for instructions. However, the register file is not updated until the instruction commits (and we know definitively that the instruction should execute); thus, the ROB supplies operands in the interval between completion of instruction execution and instruction commit.

2 Implications of Tomasulo algorithm with reorder buffer

2.1 Data Structures of Tomasulo

For the selection of functional units, I use an INTUNIT to complete memory reading and writing, an ADDUNIT to process addition instructions, subtraction instructions, branch instructions and jump instructions, a MULUNITs to calculate integer multiplication, and a DIVUNIT to calculate integer division.For each of the above devices, there are 4 reservation stations, the size of the reorder buffer is 16, and each functional unit is designed as a pipeline.

```
1  const uint32_t REORDERBUFFERSIZE=16;
2  const uint32_t ALURESERVATIONSTATIONSIZE=4;
3  const uint32_t MULRESERVATIONSTATIONSIZE=4;
4  const uint32_t MEMRESERVATIONSTATIONSIZE=4;
5  const uint32_t DIVRESERVATIONSTATIONSIZE=4;
6  class tomasulo{
7  public:
8      struct reorderBufferEntry
9      {
10         uint32_t entry;
11         bool ready;
12         instCompleteSchedule period;
13         uint32_t executeCycle;
```

```
14          uint32_t busy;
15          Inst instruction;
16          RegId destination;
17          uint64_t value;
18          uint64_t address;
19      };
20      struct registerStatusEntry
21      {
22          uint32_t entry;
23          bool busy;
24      };
25      std::vector<registerStatusEntry> registerStatus;
26
27      struct reservationStationEntry
28      {
29          uint64_t PC;
30          uint32_t entry;
31          Inst instruction;
32          // bool busy;
33          Op instructionOp;
34          int64_t vj;
35          int64_t vk;
36          uint32_t qj;
37          uint32_t qk;
38          RegId destination;
39          uint64_t address;
40      };
41
42      std::vector<reorderBufferEntry> reorderBuffer;
43      std::vector<reservationStationEntry> aluReservationStation;
44      std::vector<reservationStationEntry> mulReservationStation;
45      std::vector<reservationStationEntry> memReservationStation;
46      std::vector<reservationStationEntry> divReservationStation;
47  };
```

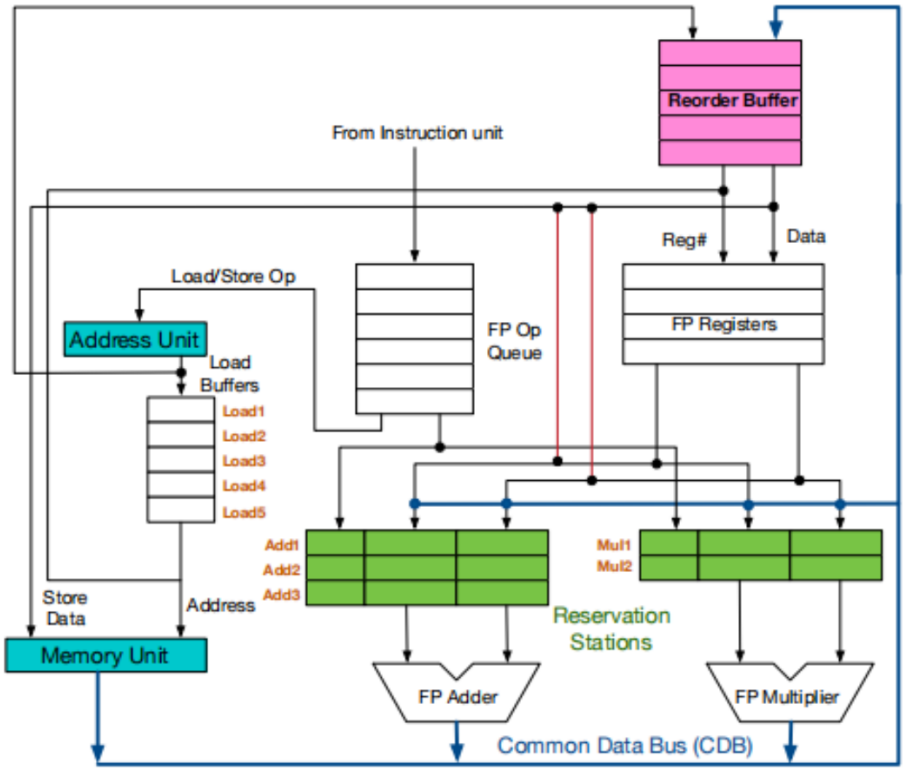
2.2 Instruction latency

For the settings of different instruction execution delays, I referred to the conclusions given in the Lab0 code.

Instruction Type	Unit	Latency
ALU	ALUUNIT	1
memCalc	INTUNIT	2
dataMem	INTUNIT	Changes
branchALU	ALUUNIT	1
iMul	MULUNIT	2
iDiv	DIVUNIT	6

2.3 Implementation of pipeline

In order to implement the Tomasulo algorithm with reorder buffer, I transformed the previous five-stage pipeline (fetch, decode, execute, memory access, write back) into a four-stage pipeline (issue, execute, writeback, commit).



Figuur 1: Tomasulo structure combined with ROB

2.3.1 Issue

The Tomasulo algorithm is issued sequentially, that is, instructions are issued to the reservation station one after another in the order in the program. The only criterion for judging whether it can be transmitted is whether the reserved station of the corresponding path of the command has a vacant position. As long as the reserved station has a vacant position, the command can be transmitted to the reserved station. At the end of the cycle, the reservation station and register result status table will be updated. If the instruction has data that can be read, it will be copied to the reservation station immediately; the register result status table always contains the latest value, that is, if the subsequent instruction If the destination register coincides with the destination register of the previous instruction, then only the write information of the subsequent instruction is retained.

2.3.2 Execute

The instruction obtains the source data by copying the data and listening to the CDB, and then starts execution. The execution may be multi-cycle, and the processor state is not changed during execution. The original Tomasulo cannot clear the reservation station when the instruction is executed, because the instruction needs to use the reservation station number to mark the instruction output and destination register. If the reservation station is allowed to be cleared when the instruction is executed, the reservation station number may be reused by subsequent instructions. Once the numbers are reused, the behavior of subsequent instructions listening to the CDB bus becomes confusing, and the results of the preceding instructions may be incorrectly written to the logic registers.

2.3.3 Write back

In the write-back phase, the instruction passes the data through the CDB bus to the register file and each reservation station.

2.4 Commit

When an instruction becomes the oldest instruction in the ROB, that is, when the head pointer of the ROB points to the instruction, the instruction can be submitted. At the end of the submission cycle, the logical register will be updated, and the corresponding Busy bit of the ROB will be set to invalid. The pointer will point to the next instruction.

2.5 Other Details

- For load and store instructions, in order to ensure their correctness, corresponding checks must also be performed - for example, when a store instruction is followed by a load instruction, and the two instructions access the same address, then the two instructions cannot Out of order.

- Due to the support of the reordering buffer, our simulator can implement sequential submission, which is conducive to achieving accurate exceptions and is more friendly to the implementation of branch prediction. Although this implementation does not add branch prediction

- The simulator implemented this time sets the functional devices to be completely pipelined, which allows us to achieve good results with a small number of functional units.

3 Quantitative evaluation and analysis

For this comparison, we used the original five-stage pipeline simulator as our control group. Since we did not implement branch prediction, we also removed the branch prediction of the original simulator for the fairness of the comparison.Some instructions require multiple cycles to be executed. We also ensure that the two simulators perform comparative experiments at the same time setting.

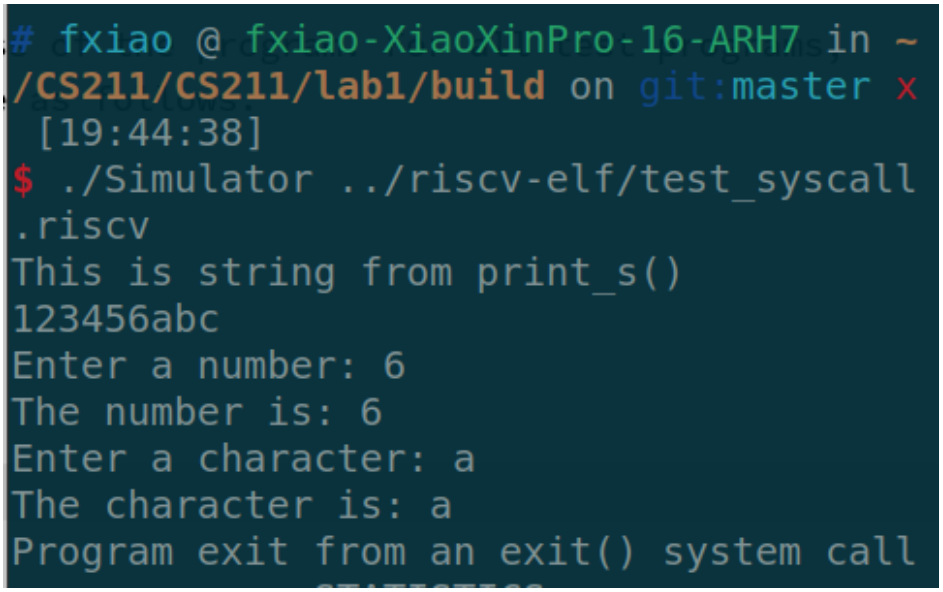
3.1 Correctness test

```
# fxiao @ fxiao-XiaoXinPro-16-ARH7 in
~/CS211/tem/CS211/lab4/build on git:ma
ster x [20:52:54]
$ ./Simulator ../riscv-elf/helloworld.
riscv
Hello, World!
Program exit from an exit() system cal
l
------------ STATISTICS -----------
Number of Instructions: 152
Number of Cycles: 241
Avg Cycles per Instrcution: 1.5855
Number of Control Hazards: 89
Number of Data Hazards: 816
Number of Memory Hazards: 0
---------------------------------
```

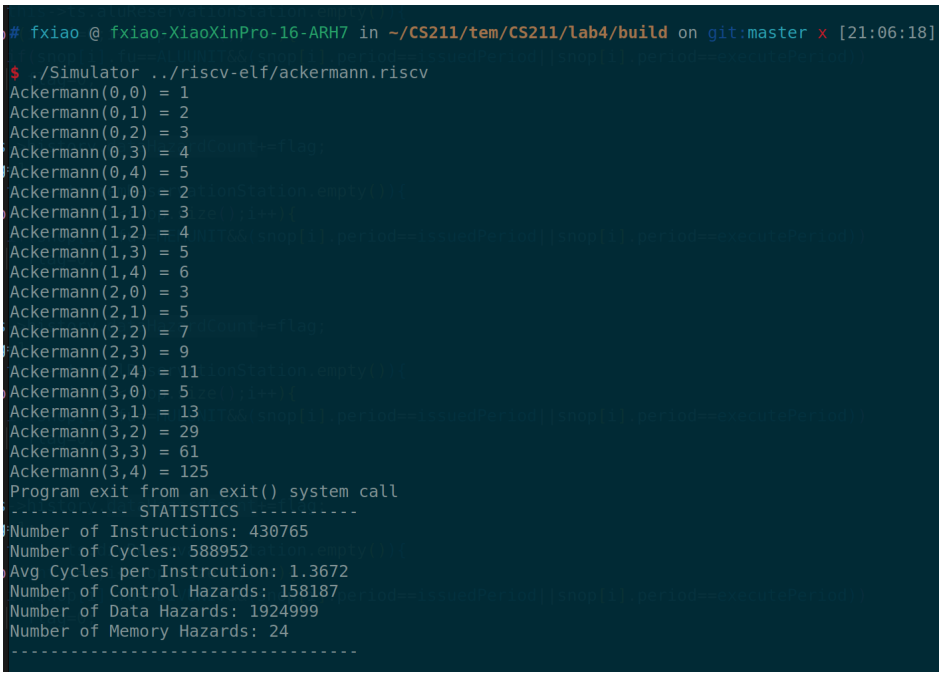
Figuur 2: Hello world test

```
# fxiao @ fxiao-XiaoXinPro-16-ARH7 in ~/CS211/tem/CS211/lab4/build
on git:master x [21:00:38]
$ ./Simulator ../riscv-elf/quicksort.riscv
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 8
1 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 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 1
4 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 4
5 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
------------ STATISTICS -----------
Number of Instructions: 103682
Number of Cycles: 135868
Avg Cycles per Instrcution: 1.3104
Number of Control Hazards: 32173
Number of Data Hazards: 423129
Number of Memory Hazards: 5292
```

Figuur 3: Quicksort Test



Figuur 4: Matrix multi Test



Figuur 5: Ackerman Test

3.2 Comparison test

As can be seen from the data in the table below, the Tomasulo algorithm we implemented comes with a simulator with a reordering buffer, which has an absolute advantage in performance, but there are more data risks. This is because we have multiple data processing units, which is more complicated. Data hazards are prone to occur, but in fact instructions will be executed every cycle.

Test Program	Tomasulo				Five stage pipeline			
	CPI	ControlH	DataH	MemoryH	CPI	ControlH	DataH	MemoryH
HelloWorld	1.5855	89	816	0	1.7961	34	78	1
Quicksort	1.3104	32173	423129	5292	1.5007	12835	93633	23398
Matrixmulti	1.5292	119316	1146434	1550	1.5783	57480	116620	11735
Ackerman	1.3672	158187	1924999	24	1.5564	61860	286492	47774

4 How can you further optimize this design

- Although it has been greatly improved compared to the previous simulator, there is still room for improvement. The first is the increase in branch predictors. We can clearly see that many cycles during program execution are caused by instruction fetching pauses caused by branch instructions.If branch prediction can be added, the pauses caused by ,control risks can be greatly reduced.
- Secondly, this simulator is relatively arbitrary in setting the size of the reservation station and reordering buffer. If its size can be carefully set, it can achieve the best performance while consuming the minimum resources.
- The third point is that when this simulator is designed, the reservation stations of each functional unit are independent. If they can be put together, the allocation of reservation stations can be adjusted more flexibly.
- Finally, if multiple read and write ports are set up during the writeback and commit phases, more instructions can be written back and committed at the same time.