Eric Gratta, Zaeem Hussain

Rami Melhem

CS2410 Project #1 Report

10/21/15

**Architecture Implementation**

General structure:

Each unit in the architecture is represented by a Java class and all are instantiated before the simulation begins. Some helper classes – Emulator, Instruction, Scoreboard, and RegisterFile – were used to simplify the code of the structural classes:

The Emulator class parsed the memory locations and branch target labels from the input file as well as the list of instructions.

The constructor for the Instruction object contains the logic for decoding. All instructions are represented as an Instruction object and passed along from unit to unit, with extra information being added to properties along the way. References to Instruction objects and their properties supplanted the physical connections between units.

The Scoreboard class contained the register renaming information and handled the logic that determined whether a register was available or not.

The RegisterFile class was a generic ArrayList<T> wrapper that had write and read functions; one instance was created for integers and another for floats.

Generally, when a structural unit has physical connections to another unit, it is instantiated with a direct reference to that unit and calls methods on it directly. Otherwise, shared information may be stored in the Instruction object or via the Scoreboard. For example, the ROB communicates writebacks and commits directly to the reservation stations and commits directly to the register files, but it also communicates release of ROB slots to the Scoreboard which the reservation station receives indirectly through the Scoreboard. As another example, the original branch prediction generated at the Fetch Unit is stored as a property in the Instruction object to be read later by the Branch Unit when determining if the prediction was incorrect. If it was incorrect, a variety of flush methods are called from the main loop but the BranchUnit communicates the event to the FetchUnit directly. We did not attempt to mirror physical connections but sought the simplest implementation.

|  |  |
| --- | --- |
| **Structures (in order)** | **Auxiliary** |
| FetchUnit  DecodeUnit  IssueUnit  ReservationStations  IntUnits  MULTUnit  FPUnit  BranchUnit  LoadStoreUnit  ReorderBuffer | Emulator  Instruction  RegisterFile  Scoreboard |

Design decisions:

The main method for our project is in the Simulator class. First, we run the Emulator on the input file and instantiated all of the structural units. Then, the main program loop begins, which executes until no more instructions are being fetched and everything has exited the pipeline. The main program simply consists of calls to the cycle() method that every structural class implemented, representing the work that would be done in a single clock cycle.

To correctly simulate parallel operations with a sequential program, we reversed the order of execution of the architecture. That is, the cycle() method for the ROB was first, then the execution units, then the reservation stations, issue unit, decode unit, and fetch unit. This caused a few complications with writeback that we had to deal with. Since the execution unit cycle() would occur before the reservation stations cycle(), data would be made available to the reservation stations that was only supposed to be available on the next cycle. The same issue occurred with commits. Thus, we queued writes and commits in the ROB object for one cycle to achieve correctness.

Assumptions:

We enforced that loads and stores needed to execute in order because there was no mechanism for knowing the memory addresses for those instructions ahead of time.

We also assumed that NB and NC are on independent buses. That is, the writeback to the reservation stations which occurs when execution units write to the ROB did not take away bus bandwidth from commits. For example, 4 writebacks and 4 commits could all happen on the same cycle if NB=NC=4.

When instructions were marked for flushing due to an incorrect branch prediction, they were flagged in the ROB and allowed to complete execution and write back but not commit. This could create some unnecessary stalls in the ROB and the reservation stations from instructions that do not contribute to the final program results.

**Comparative Analysis**

See charts below for the data that these analyses reference.

**Setup 1:**

Limiting the issuing and commits of instructions has a bad but negligible effect on the cycle count, and accordingly the IPC tends to decrease by a small amount. The program cycle counts only increase by a few instructions, with program 5 actually having the same cycle count, compared with the default. This suggests that, rather than structural hazards preventing the execution of instructions, data hazards present in the programs are the more significant bottleneck causing stalls. Programs with fewer close data dependencies may have experienced greater penalties in performance.

Since the sizes of the reservation stations and the ROB did not change, but the throughput of instructions was reduced, we see that fewer stalls occurred from instructions trying to enter the reservation stations or the ROB. Despite this slight reduction in structural hazards, the cycle count still increased from the data dependencies discussed above.

**Setup 2:**

Overall, this setup exhibited a somewhat larger increase in the cycle count than that which occurred in Setup 1. Despite the fetching and decoding being limited, it seems that the largest bottleneck in this setup was the reduction of the ROB size from 16 to 4. We can see in the last chart that the number of stalls of instructions trying to enter the ROB spiked very significantly from the default setup, while the number of stalls caused by the reservations dropped to almost none. However, reducing the throughput of instructions entering the pipeline probably also contributed to the increase in number of cycles and the corresponding decrease in IPC. Program 5, strangely, has a slight reduction in cycle count, a reason for which is hypothesized in the analysis of Setup 3.

**Setup 3:**

Overall, this setup exhibited some increases in the cycle count, although program 5 is an outlier in having a greatly decreased cycle count even though the ROB size was halved. Program 5 reveals a complex situation where, when the ROB is large (like in the default scenario) and instructions can be freely assigned to it, the reservation station quickly becomes a bottleneck that stalls the in-order issue of instructions, and additionally the penalty for branching increases because more instructions have been issued that need to cycle through the execution units before being flushed (at least in our implementation, instructions that were issued complete even if they are marked for flushing in the ROB).

If a reservation station gets filled for an execution unit that takes many cycles or contains instructions waiting on data, it will stall the issue of instructions until one of the instructions commits. With a large ROB, instructions can be freely issued until a reservation station is full, which will happen quickly, but then the issue of instructions will also be quickly blocked. Something to do with the organization of instructions using the same execution unit caused the overall number of structural stalls to decrease slightly in this scenario for program 5, while in others the number of structural stalls increased.

**Cycles**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Benchmarks | 1 | 2 | 3 | 4 | 5 |
| Default | 53 | 111 | 78 | 73 | 139 |
| 1. NW=NB=NC=2 | 56 | 113 | 80 | 78 | 139 |
| 2. NF=ND=2,NQ=NI=4, NW=NR=NC=4 | 63 | 143 | 90 | 101 | 127 |
| 3. NW=NB=NC=NF= ND=4, NQ=NI=8, NR=8 | 54 | 116 | 78 | 88 | 106 |

**IPC**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Benchmarks | 1 | 2 | 3 | 4 | 5 |
| Default | 0.5094 | 0.7477 | 0.5 | 0.7534 | 0.3819 |
| 1. NW=NB=NC=2 | 0.4821 | 0.7345 | 0.4875 | 0.7051 | 0.3819 |
| 2. NF=ND=2,NQ=NI=4, NW=NR=NC=4 | 0.4285 | 0.5804 | 0.4333 | 0.5445 | 0.4173 |
| 3. NW=NB=NC=NF= ND=4, NQ=NI=8, NR=8 | 0.5 | 0.7155 | 0.5 | 0.625 | 0.5 |

**Stalls due to reservation stations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Benchmarks | 1 | 2 | 3 | 4 | 5 |
| Default | 26 | 74 | 39 | 16 | 85 |
| 1. NW=NB=NC=2 | 20 | 53 | 28 | 11 | 70 |
| 2. NF=ND=2,NQ=NI=4, NW=NR=NC=4 | 0 | 3 | 0 | 1 | 1 |
| 3. NW=NB=NC=NF= ND=4, NQ=NI=8, NR=8 | 0 | 5 | 0 | 2 | 3 |

**Stalls due to ROB**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Benchmarks | 1 | 2 | 3 | 4 | 5 |
| Default | 0 | 0 | 0 | 27 | 11 |
| 1. NW=NB=NC=2 | 0 | 0 | 0 | 17 | 8 |
| 2. NF=ND=2,NQ=NI=4, NW=NR=NC=4 | 49 | 126 | 68 | 85 | 104 |
| 3. NW=NB=NC=NF= ND=4, NQ=NI=8, NR=8 | 30 | 86 | 60 | 68 | 81 |