ECE 4750 Computer Architecture

Topic 9: Advanced Processors – Memory Disambiguation

http://www.csl.cornell.edu/courses/ece4750
School of Electrical and Computer Engineering
Cornell University

revision: 2024-12-03-16-12

List of Problems

Problem 1. Short Answer

Part 1.A Memory Disambiguation with In-Order Load/Store Issue and Unified Stores

Consider the complete *single-issue* IO2L microarchitecture with an in-order front-end and out-of-order issue/writeback with late commit (see Figure A.7 in Appendix A). Assume we add support for memory disambiguation implemented using in-order load/store issue with unified stores (so we don't need the FLB shown in Figure A.7 in Appendix A). Consider the following instruction sequence.

```
1 mul r1, r2, r3
2 mul r4, r1, r5
3 sw  r1, 0(r6)
4 sw  r7, 0(r8)
5 lw  r9, 0(r10)  # assume R[r10] == R[r6]
```

Draw a pipeline diagram illustrating how this instruction sequence would execute on this microarchitecture. As in lecture, you can denote the integer issue queue with *ii* and the memory issue queue with *im*. **Draw microarchitectural arrows showing RAW dependencies through registers, and also draw an arrow illustrating how data is transfered using store-to-load bypassing/forwarding out of the finished store buffer.** The arrow should start in the pipe stage that writes the finished store buffer and should end in the pipe stage that *searches* and then reads the data from the finished store buffer.

`mul r1, r2, r3`																		
`mul r4, r1, r5`																		
`sw  r1, 0(r6)`																		
`sw  r7, 0(r8)`																		
`lw  r9, 0(r10)`																		

Part 1.B Memory Disambiguation with Out-of-Order Load/Store Issue

Consider the complete *quad-issue* IO2L microarchitecture with an in-order front-end and out-of-order issue/writeback with late commit (see Figure A.7 in Appendix A). This microarchitecture includes pointer-based register renaming, memory disambiguation with out-of-order load/store issue and unified stores, branch prediction, and speculative execution. You should assume that the memory disambiguation scheme with out-of-order load/store issue is exactly as described in lecture.

Consider the following assembly instruction sequence.

```
1  mul  x1, x2, x3
2  sw   x1, 0(x4)
3  mul  x5, x1, x6
4  lw   x7, 0(x8)   # Assume R[x8]  != R[x4]
5  lw   x9, 0(x10)  # Assume R[x10] == R[x4]
6  addi x9, x9, 1
7  mul  x9, x9, x11
8  addi x9, x9, 2
```

Draw a pipeline diagram illustrating how this instruction sequence executes on the IO2L microarchitecture. Use arrows on the pipeline diagram to illustrate RAW hazards through memory and RAW dependencies through memory. Your arrows should start in the stage which writes memory at a specific address and end in the stage that reads memory at that same address.

— Remember that this microarchitecture is quad-issue! —

`mul  x1, x2, x3`																										
`sw   x1, 0(x4)`																										
`mul  x5, x1, x6`																										
`lw   x7, 0(x8)`																										
`lw   x9, 0(x10)`																										
`addi x9, x9, 1`																										
`mul  x9, x9, x11`																										
`addi x9, x9, 2`																										

Answer this multiple-choice question and justify your answer.

Combining out-of-order ld/st issue with split stores would improve the performance of this instruction seqeuence:	true	false

Appendix A: TinyRV1 Canonical Microarchitectures

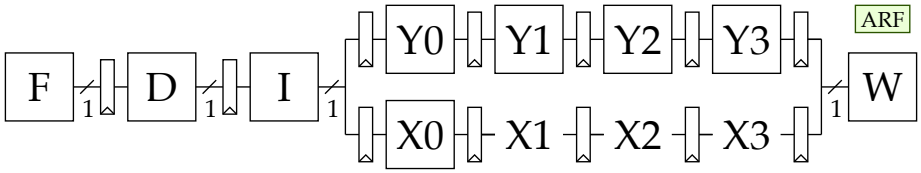

Figure A.1: I3L Microarchitecture for MUL, ADDU, ADDIU

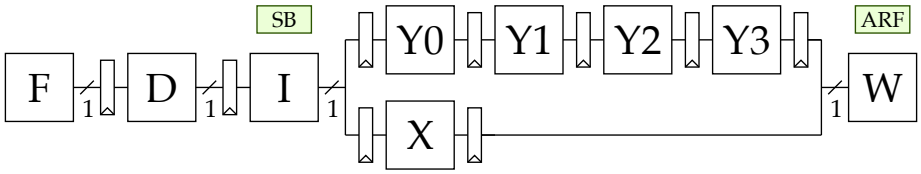

Figure A.2: I2OE Microarchitecture for MUL, ADDU, ADDIU

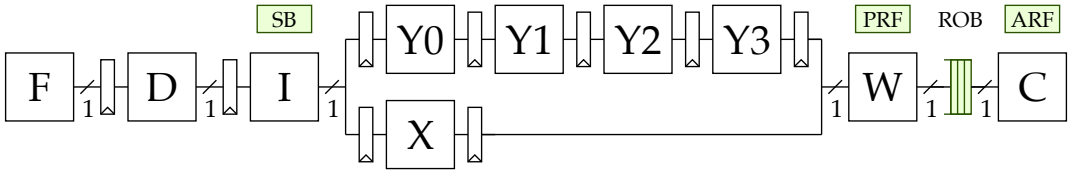

Figure A.3: I2OL Microarchitecture for MUL, ADDU, ADDIU

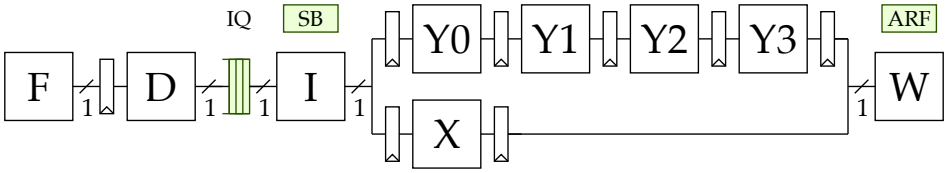

Figure A.4: IO2E Microarchitecture for MUL, ADDU, ADDIU

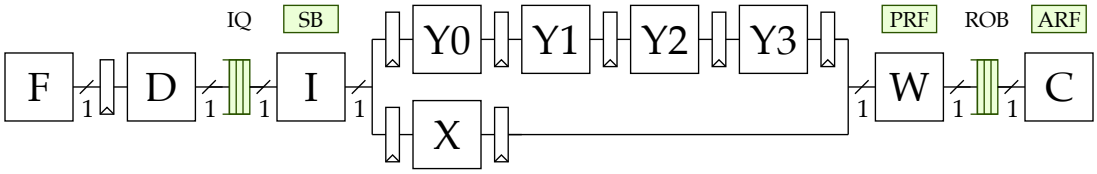

Figure A.5: IO2L Microarchitecture for MUL, ADDU, ADDIU

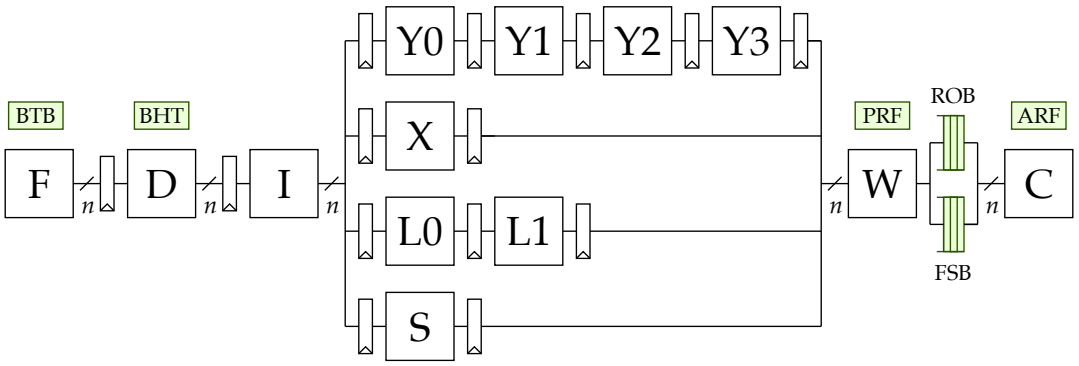

Figure A.6: Complete I2OL Microarchitecture (single issue: n = 1; quad issue: n = 4)

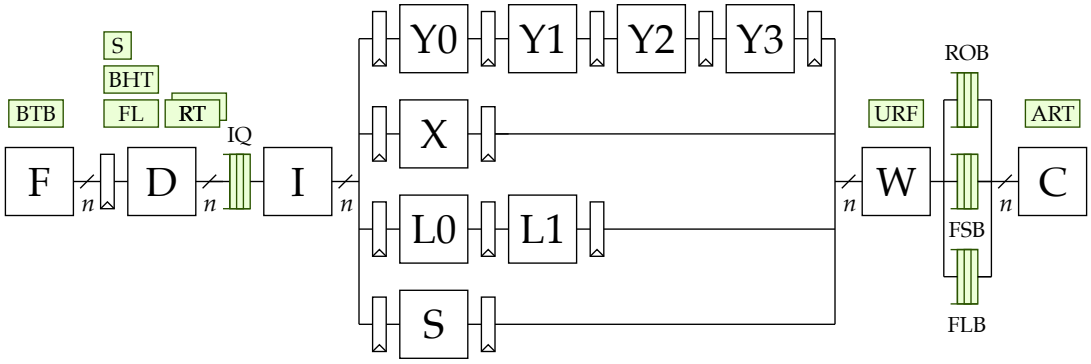

Figure A.7: Complete IO2L Microarchitecture (single issue: n = 1; quad issue: n = 4)