

15-740/18-740 Computer Architecture

Homework 4

Due Monday, October 17, at 12:00 PM

1. The Reorder Buffer

The reorder buffer (ROB) plays a key role in ensuring in-order execution semantics are exposed to the programmer despite the underlying hardware possibly executing instructions out of program order. In this problem, you will perform the tasks of the ROB.

(a) Say a program consists of the following instruction stream:

```
0x0040: LD  r0 ← [0xABCD]
0x0044: ADD r1 ← r2, r3
```

Assume:

- A LD takes 3 cycles to execute and an ADD takes 1 cycle to execute.
- The LD and ADD can write their results to the ROB on the same cycle they are ready (i.e., a bypass path for both tag and data values are used in this system).
- It takes one cycle for the ROB to write data ready to be committed to architectural state to the architectural register file.
- The initial contents of the architectural register file is $r0 = 5$, $r1 = 4$, $r2 = 3$, and $r3 = 2$.
- The value at address $0xABCD$ is 1.

Fill in the contents of the architectural register file and a ROB over the next six cycles for a processor which can issue 1 instruction per cycle. Here is the initial state to get you started:

Cycle 0:

Architectural register file

r0	5
r1	4
r2	3
r3	2

ROB

Valid	Done	InstructionAddr	ResultRegister	ResultValue
0	-	-	-	-
0	-	-	-	-

(b) Now assume that on a system without an ROB an exception occurs when the ADD is executed. What will the contents of the architectural register file be at the time of the exception (when it is exposed to the programmer)? Why is this bad?

2. Out-of-Order Execution

Out-of-order processors make efficient use of their functional units by executing instructions according to the flow of data between them.

(a) In class, we learned about a graphical way of showing the data dependencies (edges) of instructions (vertices) using a data flow graph. For the instruction stream below, draw the corresponding data flow graph.

```
ADD r3 ← r1, r2
MUL r4 ← r1, r3
ADD r5 ← r8, r9
DIV r6 ← r1, r3
ADD r7 ← r6, r5
```

- (b) Of course, while a data flow graph is a helpful way of visualizing what goes on in an out-of-order processor, real machines execute instructions using multiple pipeline stages. Take the code from above and find the number of cycles it would take to execute on a processor with *in-order fetch* and *out-of-order dispatch*. Assume the following:
- It takes one cycle to insert an instruction into a reservation station and another cycle to schedule the instruction from the reservation station to an execution unit and these stages can be pipelined.
 - Tags are broadcast one cycle before their corresponding operation finishes executing.
 - A single tag broadcast bus exists and if more than one reservation station entry to the same execution unit is ready to be dispatched, the older reservation station entry will use the bus and the younger ones will have to stall. Similarly, if more than one instruction contends for writing to the ROB or architectural register file during a single cycle, the oldest one receives access and the younger ones must stall.
 - ADDs take 3 cycles and are pipelined, MULs take 5 cycles and are pipelined, and DIVs take 7 cycles and are pipelined.
- (c) How many cycles will the program take to execute on an *in-order fetch* and *in-order dispatch* machine, under the same assumptions as above?