LSU EE 4720 **Homework 3** Solution **Due: 25 April 2005**

Problem 1: Do Problem 1 in the Spring 2004 EE 4720 final exam. Grade yourself using the solution, the grade should be out of five points. When grading yourself please explain what the mistakes were and what the correct answer should be, as a helpful grader would. Be polite in your explanations unless there was no serious attempt to solve the problem. In that case point out how final exam study time is being undermined by the need to catch up.

Problem 2: In Method 3 the commit register map is used to recover the state the ID register map was in just after the most recently committed instruction was decoded. In a system in which the ID register map is checkpointed for predicted CTIs, the commit register map won't be used very often.

(*a*) Describe how a system using Method 3 but without the commit register map could recover the ID map state before a faulting instruction. The ID map would be recovered using information in the ROB at and after the faulting instruction.

- Explain, with an example, what steps the processor takes to recover the information.

A ROB entry for an instruction includes, among other things, the architected destination register (call it rd), the new physical register that rd maps to (and the one the instruction will write), and the incumbent, that is, the physical register that rd mapped to before the instruction was decoded. When (and if) the instruction commits it is the incumbent that will be put back on the free list.

When recovery is necessary the incumbents can be used to return the ID map to the state it was before any in-flight instruction was decoded by writing the incumbents back into the register file starting from the tail of the ROB up to the faulting or mispredicted instruction.

The example below shows the state of a system with five instructions in flight. The numbers on the left show the architected destination, physical destination, and incumbent physical destination registers. Suppose the sw raises an exception. The ID map can be recovered by removing the tail entry from the ROB, and writing the incumbent back into the register map (using the architected destination as a key). This process is repeated until the faulting instruction is removed. In the example below ID shows normal activity. Squash recovery starts in cycle 10 and continues in 11 and 12; SQ shows a tail instruction being removed and the incumbent being written back, this happens for each of the instructions to be squashed.

If it is only possible to remove instructions from the head then recovery is a bit trickier since only the first incumbent (for each architected register) should be written to the ID register map. This can be done by keeping track of which architected registers have been written (using a bit vector) and not writing one twice. Apart from needing a bit vector, a disadvantage recovering from the head is that recovery can't start until after the faulting or mispredicted instruction reaches the head.

```
dst PR inc                  Normal Decode     Recovery----
r1  11 10   add r1, r2, r3   ID                               HEAD END
           sw 0(r0), r0
r4  16 15   sub r4, r2, r3        ID                   SQ
r1  4  11   xor r1, r2, r3           ID           SQ
r1  7  4    or  r1, r2, r3              ID     SQ             TAIL END
Cycle                       0 1  2  3  4  ... 10  11  12

ID Register Map
           r1   10           11     4  7      4   11
           r4   15                16                   15
```

(*b*) Show new connections to the ID register map to implement this. Try to do it without adding new read and write ports (that is, use existing ports).

Add a mux to the Addr and D In ports currently used to write the new physical register. On input to each mux is the existing connection to the port, the other is taken from either the head output (this part) or tail output (next problem).

(*c*) Describe the impact on performance when the technique is used for exceptions.

Since instructions must be squashed one at a time rather than all at once recovery will take longer. However, since exceptions are rare that won't have a big impact on performance.

(*d*) Describe the impact on performance when the technique is used for mispredicted branches.

If there are alot of instructions to squash then this will slow down the recovery process.

Problem 3: Consider the commit mapless system from the previous problem. Suppose it were possible to sequentially read the ROB from two locations, the head (as is currently done for committing instructions) and some other place, say at a mispredicted branch.

(*a*) How might this be used to recover the ID map faster than was done in the previous problem.

As explained in the previous problem, immediately start recovery by removing instructions from the tail (that's the other place) rather than waiting for the branch to reach the head.

(*b*) If this can be made to work for branches then there would be no need for checkpointing the register map. The impact on performance when using the mechanism for mispredicted branches depends on the following factors: how fast instructions are fetched, how many cycles it takes to resolve a branch (determine if the prediction was correct), how long it takes the fetch mechanism to bring correct-path instruction to ID, and how fast recovery can be done. Show a formula that will give the number of extra cycles needed to recover from a branch misprediction using this scheme (compared to checkpointing). For the formula, use the factors listed above and any other that is relevant.

Note that the amount of time to fetch the first correct-path instruction is a variable, it can be more than the one cycle shown in most other problems.

When a misprediction occurs two things have to happen: correct path instructions need to be fetched and the ID register map needs to be recovered. (Other things need to be done, but not for this problem.) The formula should indicate how much longer ID map recovery takes than getting the correct path instructions.

Let i_{f} denote the number of instructions that can be fetched per cycle and let i_{r} denote the number of instructions that can be squashed per cycle (as described in the previous problem). Let $t_{\mathrm{BR-ID-WB}}$ denote the average number of cycles a branch takes to resolve (move from ID to WB) and let $t_{\mathrm{BR-WB-ID}}$ denote the average number of cycles from when a mispredicted branch reaches writeback until the first correct path instruction is fetched and reaches ID.

When a misprediction occurs the number of instructions that need to be squashed is $i_{\mathrm{f}} \times t_{\mathrm{BR-ID-WB}}$; the ID map recovery will take $i_{\mathrm{f}} \times t_{\mathrm{BR-ID-WB}}/i_{\mathrm{r}}$ cycles. The amount of extra waiting time is then

$$\max\{0, \frac{i_{\mathrm{f}} \times t_{\mathrm{BR-ID-WB}}}{i_{\mathrm{r}}} - t_{\mathrm{BR-WB-ID}}\} \quad \text{cycles}$$

In an $n-$way superscalar processor $i_{\mathrm{f}} = n$ and it would be reasonable to expect i_{r} also to be n. Therefore there will be a performance penalty if average resolution time is longer than fetch time, which is only sometimes true. If it is not be too expensive to provide extra ports so that $i_{\mathrm{r}} > n$ then the penalty can be reduced further or eliminated.