ECEn 528

Study Guide ­ Dynamic Scheduling and Speculation

● Read Sections C.7, 3.4­3.6 and 3.8 of H&P  Things to focus on

■ Do not worry too much about all of the details of how the scoreboard maintains instruction state in section C.7; instead, focus on the basic idea that the scoreboard must look for potential WAR,

WAW, and RAW hazards and prevent instructions from issuing when the hazards are there

■ Again, don't worry about all the details of Tomasulo's scheme in Figure 3.9. But do understand why it works: how the WAR/WAW hazards are removed and the essential dependences satisfied. The loop­based example is particularly important.

■ Working through Case Study 1 in chapter 3 as we learn the techniques could be very helpful

■ Don't worry about all the details in figure 3.14, but do understand how the reorder buffer actually works

■ Understand why there is a difference in timing between figures 3.19 and 3.20

■ If you're interested in history, here is Robert Tomasulo talking about the IBM 360/91 at University of Michigan: http://leccap.engin.umich.edu/leccap/view/tomasulo/5047

 Clarifications

■ Dynamic scheduling is also known as out­of­order execution (OoO)

■ Dynamic scheduling is an interesting variation of “clever” solutions to hazards: we still “stall” for hazards, but the stall of one instruction doesn't affect other instructions directly as it does in an in­order pipe.

■ The description of Tomasulo's scheme uses dynamic scheduling only for the floating­point units; the algorithm can be used for all instructions. Historically, though, when it was first used, it was only used for floating­point instructions.

■ The book says that placing the instruction in a reservation station is “issue”, and then starting its execution is “execute”. The alternate terminology they suggest (“dispatch” and “issue”) is also extremely common, and more descriptive. Even worse, some people mix the two and call the process “issue” then “dispatch”!

■ The description of exception handling on p. 174 does not provide precise exceptions!

■ The description of reservation stations assumes one “group” of reservation stations per functional unit; it is possible for functional units to share a reservation station, though the control logic is more complex.

■ A statement is made on page 175 that in a dynamically scheduled pipeline, there must be one extra cycle of latency between a producing instruction and a consuming instruction. This is not true in general, but does require more hardware to remove that cycle and the circuit timing is tricky. However, as we will see in a paper we read later on, removing that cycle is very important.

■ The subtle distinction made in the second paragraph of Section 3.6 is that their presentation of dynamic scheduling did not allow scheduling beyond a branch and didn't provide precise exceptions

■ Note that a “dataflow execution” where instructions execute whenever their operands are available quite naturally leads to the issue of multiple instructions per cycle.

■ The reorder buffer is like a future file, but is not organized like a register file, but rather a FIFO with associative lookup (on register numbers)

 Answer the following questions:

1. Why are WAR hazards possible in a dynamically scheduled pipeline?

The instructions can execute and finish out of order

1. Why is dynamic scheduling helpful? Think of a situation in which dynamic scheduling would perform better than static scheduling.

It allows code compiled for one pipeline to work on others; it allows handling of dependences that can’t be known at compile-time; it allows the processor to tolerate unpredictable delays, such as cache misses

1. Why do compilers still schedule code even if the hardware uses dynamic scheduling?

It decreases the amount of work the hardware has to do, and not all hardware is guaranteed to use dynamic scheduling(?)

1. Why does Tomasulo's scheme allow removal of hazards which arise from name dependences which a compiler cannot?

Tomasulo’s scheme allows renaming across branches, which the compiler cannot always determine

1. Why does the description of exception handling on p. 95­96 not provide precise exceptions?

There is a possibility of continuing to execute instructions that should not be executed

1. Why does dynamic scheduling require accurate branch prediction?

The processor must know the BP was correct before allowing an instruction after the branch to begin execution

1. How does the use of a reorder buffer differ from Tomasulo's scheme?

In Tomasulo’s, once an instruction writes its result, any subsequently issued instructions will find the result in the register file.

With speculation, the register file is not updated until the instruction commits, and the ROB supplies operands in the interval between the completion of instruction execution and instruction commit.

1. Why must commit be performed in order?

It prevents irrevocable actions (updating state/taking an exception) until instructions commit

1. How many instructions can be in flight when using a reorder buffer? How many when using register renaming?

Depth of ROB; number of registers available

1. Why might you want to handle a TLB miss at commit instead of when it is first detected?

It could have potentially resolved by then? Something about page faults?

1. How would exceptions or branch mispredictions be handled in a register renaming scheme?

???