Notes for 10/2/09

Data hazards—created when data in registers is modified that other instructions already in the pipeline need. These can be ameliorated by forwarding/bypassing data to instructions dependent on the modified data earlier in the pipeline.

Structural hazards—occurs when part of a processor's hardware, such as the ALU, is needed by more than one instruction at the same time.

Control hazards—what can we put in the pipeline if we have not evaluated a conditional branch yet. This is partially ameliorated by the branch delay slot—an instruction after the branch that always gets executed.

You can also use branch prediction to help ameliorate control hazards.

- Static branch condition: We can always assume the branch is taken or not taken (not taken is not a good prediction for loops).
- Dynamic branch condition: Relies on dynamic state of the processor to predict a branch, e.g. a single bit predictor that always predicts you will branch the same way that you branched last time. More sophisticated predictors include 2-bit and correlating and/or tournament predictors.

If your branch prediction predicts incorrectly, then you have to flush the pipeline.
A branch target buffer contains addresses in it to help predict where a branch will branch to.

Superscalar—try to get greater than one IPC

Dynamic vs. static multiple issue

Static multiple issue—consider the Itanium—EPIC (explicitly parallel instruction computer). It is the compiler's job to decide which instructions can be executed simultaneously and to insert bubbles to prevent hazards.

If you have already defined the ISA and if you have already set that the processor can only fetch one instruction at a time, then you can still use dynamic multiple issue.

Dynamic multiple issue—consider Tomasulo's algorithm—even if the semantics of the ISA are that only one instruction can be fetched at a time, then your hardware can still try to execute instructions simultaneously if you make sure that you do not introduce data hazards.

Tomasulo's algorithm:
Common data bus—used to broadcast data to reservation stations that may need it
Reservation stations—allows the processor to fetch and reuse data without needing to wait for it to be stored into a register
Reorder buffer—allows out-of-order execution by reordering instructions to be in order before the instructions are committed; this facilitates precise interrupts, i.e. so that, when an interrupt occurs, you can assume that all instructions before the faulting instruction have been committed and that all instructions after the faulting instruction have not been committed
Register renaming—used to get around false dependencies when you reuse the same register name but do not use the same value in it. With static multiple issue, the compiler does the renaming. With dynamic, the reservation stations handle this.