

Pipelining Summary

Issues

- Time/Instruction vs. Throughput
- Control Hazards
 - conditional branching, ret instruction
- Data Hazards
 - Detection
 - Resolution
- Implementation Complications

Pipeline Performance

- Each instruction will take longer than with a non-pipelined system.
 - need to add *holding* registers
 - control is more complex (more stuff usually means *slower*).
- Throughput is improved
 - If the pipeline is always full, we finish one instruction every cycle.

Performance Metrics

- Clock Rate (Cycle Time)
- CPI: Cycles Per Instruction
 - an *average* over all instructions in some program being measured.
 - depends on system architecture
 - depends on the program being measured
 - does not depend on clock rate!

Seq vs Pipe

- CPI for SEQ Y86 is 1!
 - but we need a slow clock...
- At best, CPI for Pipelined Y86 is 1.
 - not attainable, branch control and data hazards will result in some stalls (bubbles in the pipeline).
 - Text shows derivation of CPI of 1.27
 - clock can run much faster than SEQ implementation
 - determined by the slowest stage in the pipeline.

SEQ vs. Pipelined Hypothetical Comparision

- Same program.
- CPI 1.0 vs. 1.27
- Seq Clock rate: 1MHz
- Pipelined Clock rate: 10MHz

Control Hazards

- Can't decide what to do when the decision needs to be made.
 - information required is not yet available.
- Examples
 - Conditional Branches (Jumps)
 - The ret instruction

Control Hazard Resolution

- Branch Prediction
 - assume branch is taken (or not). Must restart the pipeline later if the wrong decision was made (wasted cycles).
 - Some processors have complex branch prediction support (makes prediction based on past history).
- Stall the pipeline (necessary for ret instruction).

Data Hazards

- Data is not available when needed for some computation.
 - result of previous computation not yet stored in register.
 - result of read from memory not yet in register.
- Detection requires more complex control.

Data Hazard Resolution

- Can always stall the pipeline. (wasted cycles in some cases).
- If the data is available, but not yet in the right place: forwarding.
 - select ALU input from either result of decode state, result of previous ALU op, or last item read from memory.
- Not all data hazards can be resolved by forwarding!

Control Complexity

- Check section 4.5.8 for derivation of HCL expressions for pipelined control (that includes forwarding).
 - you won't be tested on pipelined control.
 - you should realize that although it's more complicated, it is feasible!

Implementation Complications

- Multicycle operations
 - integer division
 - floating point arithmetic.
 - Memory (cache miss)
- Execute stage can hold up the pipeline (everyone must wait for the ALU to finish).
 - It's not easy to resolve this, other than attempt to make a better ALU!
 - dynamic scheduling...