Lecture 10: ILP Innovations

• Today: handling memory dependences with the LSQ and innovations for each pipeline stage (Section 3.5, detailed notes)
The Alpha 21264 Out-of-Order Implementation

Branch prediction and instr fetch

Instr Fetch Queue

- R1 ← R1+R2
- R2 ← R1+R3
- BEQZ R2
- R3 ← R1+R2
- R1 ← R3+R2

Decoded & Rename

- Instr 1
- Instr 2
- Instr 3
- Instr 4
- Instr 5
- Instr 6

Reorder Buffer (ROB)

- Commited Reg Map
  - R1→P1
  - R2→P2

- P33 ← P1+P2
- P34 ← P33+P3
- BEQZ P34
- P35 ← P33+P34
- P36 ← P35+P34

Issue Queue (IQ)

- ALU
- ALU
- ALU

Speculative Reg Map
- R1→P36
- R2→P34

Register File
- P1→P64

Results written to regfile and tags broadcast to IQ
Out-of-Order Loads/Stores

What if the issue queue also had load/store instructions? Can we continue executing instructions out-of-order?
The issue queue checks for register dependences and executes instructions as soon as registers are ready.

Loads/stores access memory as well – must check for RAW, WAW, and WAR hazards for memory as well.

Hence, first check for register dependences to compute effective addresses; then check for memory dependences.
Memory Dependence Checking

- Load and store addresses are maintained in program order in the Load/Store Queue (LSQ)

- Loads can issue if they are guaranteed to not have true dependences with earlier stores

- Stores can issue only if we are ready to modify memory (can not recover if an earlier instr raises an exception)
Improving Performance

- Techniques to increase performance:
  - pipelining
    - improves clock speed
    - increases number of in-flight instructions
  - hazard/stall elimination
    - branch prediction
    - register renaming
    - efficient caching
    - out-of-order execution with large windows
    - memory disambiguation
    - bypassing
  - increased pipeline bandwidth
Deep Pipelining

- Increases the number of in-flight instructions
- Decreases the gap between successive independent instructions
- Increases the gap between dependent instructions
- Depending on the ILP in a program, there is an optimal pipeline depth
- Tough to pipeline some structures; increases the cost of bypassing
Increasing Width

- Difficult to find more than four independent instructions
- Difficult to fetch more than six instructions (else, must predict multiple branches)
- Increases the number of ports per structure
Reducing Stalls in Fetch

- Better branch prediction
  - novel ways to index/update and avoid aliasing
  - cascading branch predictors

- Trace cache
  - stores instructions in the common order of execution, not in sequential order
  - in Intel processors, the trace cache stores pre-decoded instructions
Reducing Stalls in Rename/Regfile

• Larger ROB/register file/issue queue

• Virtual physical registers: assign virtual register names to instructions, but assign a physical register only when the value is made available

• Runahead: while a long instruction waits, let a thread run ahead to prefetch (this thread can deallocate resources more aggressively than a processor supporting precise execution)

• Two-level register files: values being kept around in the register file for precise exceptions can be moved to 2nd level
Stalls in Issue Queue

- Two-level issue queues: 2nd level contains instructions that are less likely to be woken up in the near future

- Value prediction: tries to circumvent RAW hazards

- Memory dependence prediction: allows a load to execute even if there are prior stores with unresolved addresses

- Load hit prediction: instructions are scheduled early, assuming that the load will hit in cache
Functional Units

- Clustering: allows quick bypass among a small group of functional units; FUs can also be associated with a subset of the register file and issue queue
Title

• Bullet