Contents

Instruction Scheduling

- Scheduling: act of finding independent instructions
  - "Static" done at compile time by the compiler (software)
  - "Dynamic" done at runtime by the processor (hardware)
- Why schedule code?
  - Scalar pipelines: fill in load-to-use delay slots to improve CPI
  - Superscalar: place independent instructions together
    * As above, load-to-use delay slots
    * Allow multiple-issue decode logic to let them execute at the same time

Dynamic Instruction Scheduling

- Also called "out-of-order" processors
- Hardware re-schedules instructions within a sliding window of instructions
- As with pipelining and superscalar, ISA unchanged
  - Same hardware/software interface, appearance of in-order
- Increases scheduling scope
  - Does loop unrolling transparently
  - Uses branch prediction to "unroll" branches
- "Dynamic scheduling" done by hardware

- Still 2-wide superscalar, but now out-of-order too
  - Allows instructions to issue when dependencies are ready
- Longer pipeline
  - In-order front end: Fetch, "Dispatch"
  - Out-of-order execution core: "Issue", "RegisterRead", Execute, Memory, Writeback
  - In-order retirement: "Commit"

Dependence Types

- **RAW** (read after write): true dependence
- **WAW** (write after write): output dependence
- **WAR** (write after read): anti-dependence

WAW & WAR are false; can be totally eliminated by renaming

Register Renaming

- To eliminate register conflicts/hazards
- Architectural vs Physical registers - level of indirection
- Renaming - conceptually write each register once
  - Removes false dependencies
  - Leaves true dependencies intact
- When to reuse a physical register?
  - After overwriting instructions is complete

Simplified

- Two key data structures:
  - `maptable[architectural_reg] -> physical_reg`
  - Free list: allocate (new) and free registers (implemented as a queue)
    * Ignore freeing of registers for now
- Algorithm: at decode stage for each instruction
  - Rewrites instruction with physical registers (rather than architectural registers)

Dynamic Instruction Scheduling Mechanisms

Dispatch

- Put renamed instructions into out-of-order structures
- Re-order buffer
  - Holds instructions from Fetch through Commit
- Issue Queue
  - Central piece of scheduling logic
  - Holds instructions from Dispatch through Issue
  - Tracks ready inputs
    * Physical register names + ready bit
    * "AND" the bits to tell is ready

Out-of-order Pipeline

- Execution (out-of-order) stages
- **Select** ready instructions
  - Send for execution
- **Wakeup** dependents

Content Addressable Memory

- A content addressable memory (CAM) is indexed by the **content** of each location, not by the address
  - Sometimes known as associative memory
- It compares an input key against a table of keys, and returns the location of the key in the table
  - In software this might be implemented with a hash table
  - Hardware hash table is also possible, but potentially slow
- To search all locations in a single cycle
  - You need to be able to compare the search key to all keys in the table simultaneously
    * This requires a lot of hardware
    * Fast CAMs are very hardware expensive
  - If you need to be able to do multiple searches in the same cycle, the hardware requirements are even greater

Out-of-order Benefits and Challenges

Benefits

- Allows speculative re-ordering
  - Loads/stores
  - Branch prediction to look past branches
- Done by hardware
  - Compiler may want different schedule for different hardware configs
  - Hardware has only its own configuration to deal with
- Schedule can change due to cache misses
- Memory-level parallelism
  - Executes "around" cache misses to find independent instructions
  - Finds and initiates independent misses, reducing memory latency

Challenges

- Design complexity
  - More complicated than in-order? Certainly!
  - But we have managed to overcome the design complexity
- Block frequency
  - Can we build a "high ILP" machine at a high clock frequency?
  - Yep with some additional pipe stages, clever design
- Limits to (efficiently) scaling the window and ILP
  - Large physical register file
  - Fast register renaming/wakeup/select/load queue/store queue
    - Active areas of micro-architectual research
  - Branch & memory depend prediction (limits effective window size)
    - 95% branch misprediction: 1 in 20 branches, or 1 in 100 insn
  - Plus all the issues of building wide in-order superscalar
- Power efficiency
  - Today, even mobile phone chips are out-of-order cores