CS 152, Spring 2012
Section 8

Christopher Celio
Agenda

– More Out-of-Order
Intel Core 2 Duo (Penryn) Vs. NVidia GTX 280

- Intel Core 2 Duo (Penryn)
  - dual-core
  - 2007+
  - 45nm
  - 410 million transistors
  - ~2GHz
  - 3 or 6MB of cache
  - 10-35 Watts
  - 107mm²
    - each core is 22mm²
    - L2 SRAM is 6mm²/MB

- NVidia GTX 280
  - 10 core(?) (240 “stream” processors)
  - 2008
  - 65nm
  - 1.4 Billion transistors
  - 576mm²
  - 602 MHz(core clock)
  - 236 Watts !!!


Friday, March 16, 2012
Quiz 2

• Will be returned this Tuesday
Out-of-Order Control Complexity: MIPS R10000

[ SGI/MIPS Technologies Inc., 1995 ]
Out of Order Processors

Figure 2. R10000 block diagram (a) and pipeline timing diagram (b). The block diagram shows pipeline stages left to right to correspond to pipeline timing.

Out of Order Processors

Figure 6. Integer instruction queue, showing only one issue port. The queue issues two instructions in parallel.
BOOM: A Single Issue Slot

Control Signals

Physical Destination Register

Physical Source Registers

Issued to the Register Read stage

(UOP Code, BrMask, Ctrl...) → Resolve or Kill → Br Logic → Resolve or Kill

(From the register file's two write ports)

(WDest0, WDest1) → = → = → = → = → Ready

RS1, p1 → Ready

RS2, p2 → Ready

Issued to the Register Read stage
each instruction gets a br mask... allows us to kill instructions

(From the register file's two write ports)

the register file has two write-ports, so watch both ports' write addresses

each slot asserts request when ready to fire

one slot gets the “issue”

(note: I show a bus implementation, but it's actually implemented with a bunch of muxes)
OOO Styles
On dispatch into ROB, ready sources can be in regfile or in ROB dest (copied into src1/src2 if ready before dispatch)
On completion, write to dest field and broadcast to src fields.
On issue, read from ROB src fields
Unified Physical Register File
(MIPS R10K, Alpha 21264, Intel Pentium 4 & Sandy Bridge)

- Rename all architectural registers into a single *physical* register file during decode, no register values read
- Functional units read and write from single unified register file holding committed and temporary registers in execute
- Commit only updates mapping of architectural register to physical register, no data movement
• As mentioned earlier, 21264 uses explicit renaming, as opposed to data-in-ROB design
• What does ROB hold?
DEC Alpha 21264

- 1996/1997
- single-core
  - 4-way
  - out-of-order
  - highly speculative
  - 7-stage
  - up to 80 instructions in flight
  - tournament branch predictor
- 15.2M transistors
  - 6M for logic
  - rest is caching, history tables
- 350 nm
- 600 MHz
- 64KB I$, 64KB D$ (on-chip)
  - 1 to 16MB L2$ (off-chip)
- 314mm$^2$ die (fairly large)
DEC Alpha 21264
21264 Register Renaming

- Registers are renamed, then instructions are inserted into the issue queue (window)
- Map table backed up on every in-flight insn
• What hazards does renaming obviate?

• In what situations is renaming useful?

• If you had to choose between branch prediction and renaming, which would you pick?
21264 Register Renaming

• What hazards does renaming obviate?
  – WAR, WAW

• In what situations is renaming useful?

• If you had to choose between branch prediction and renaming, which would you pick?
• What hazards does renaming obviate?
  – WAR, WAW

• In what situations is renaming useful?
  – Code with ILP and name dependencies: loops

• If you had to choose between branch prediction and renaming, which would you pick?
21264 Register Renaming

• What hazards does renaming obviate?
  – WAR, WAW

• In what situations is renaming useful?
  – Code with ILP and name dependencies: loops

• If you had to choose between branch prediction and renaming, which would you pick?
  – Not much ILP within a basic block, so renaming isn’t too useful without branch prediction
21264 Superscalar Execution

- 21264 couldn’t fit full bypassing into one clock cycle
- Instead, they fully bypass within each of two clusters; inter-cluster bypass takes another cycle
Question: Stores

• When are stores sent to memory?
  – at commit time

• Why are stores saved in a store buffer before commit time?
  – so they can be forwarded to dependent loads
only showing comparison logic for one Load

load is ready to fire
load can be bypassed out of SDQ
location in SDQ to get ld data from

Data Mem

Friday, March 16, 2012
- single issue
- 6-stage
- full branch speculation (BHT)
- magic, 1-cycle memory (no caches)
- no bypasses
- no floating point
- no exceptions
• To execute the critical instruction path quickly, want to execute loads ASAP
• Initially, loads speculatively bypass stores
• On a misspeculation, set a “wait” bit for that load’s PC, so it will behave conservatively from then on
• Clear wait bits periodically
Speculation in the 21264

• What does the 21264 speculate on?
  – Next I$ line/way
  – Branches, indirect jumps
  – Exceptions
  – Load/Store ordering
  – Load hit/miss
    • Shortens hit time by a cycle
  – Anything else?
Pentium processor

http://www.cs.clemson.edu/~mark/330/p6.html

Friday, March 16, 2012