CS 61C: Great Ideas in Computer Architecture (Machine Structures)
RISC in Retrospect, Misc Topics
(Fixed Point, Polling vs. Interrupts)

Instructor:
Michael Greenbaum

New-School Machine Structures

Today’s lecture
Hardware
Software
Warehousing & Achieve High Performance

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time

Smart Phone
Warehouse Scale
Computer

Core
Core
Core
Core

Input/Output
Memories
Functional Unit(s)

Logic Gates

Memory Controllers

Core Area Breakdown

In-Order Fetch

In-Order Decode and Register Renaming

In-Order Commit

Out-of-Order Execution

2 Threads per Core
Final Sync
Front-End Instruction Fetch & Decode

- 128 Entry TLB (4 way)
- 32 KB L1-cache (4 way)

Instruction Fetch Unit

- 18B Pre-Decoded, Fetch Buffer
- 18 Entry Instruction Queue

µOP (internal RISC-like) instruction, into which x86 instructions are translated

µOP is Intel name for internal RISC-like (MIPS) instruction bits

x86 bits

Loop Stream Detector (can run short loops out of the buffer)

x86 Decoding

- Translate up to 4 x86 instructions into µOPS (≈MIPS or RISC instructions) each cycle
- Only first x86 instruction in group can be complex (maps to 1-4 µOPS), rest must be simple (map to one µOP)
- Even more complex instructions, jump into microcode engine which spits out stream of µOPS

Branch Prediction

- Part of instruction fetch unit
- Several different types of branch predictor
  - Details not public
- Two-level Branch Table Buffer
- Loop count predictor
  - How many backwards taken branches before loop exit

Out-of-Order Execution Engine

- Renaming happens at uOP level (not original macro-x86 instructions)
- uOPs are scheduled based on dependency graph

Simultaneous multi-threading enhances performance and energy efficiency

Simultaneous Multi-Threading (SMT)
- Run 2 threads at the same time per core
- Take advantage of 4-wide execution engine
  - Keep it fed with multiple threads
  - Hide latency of a single thread
- Most power efficient performance feature
  - Very low die area cost
  - Can provide significant performance benefit depending on application
  - Much more efficient than adding an entire core
- Intel® Core™ microarchitecture (Nehalem) advantages
  - Larger caches
  - Massive memory BW
Nehalem Memory Hierarchy Overview

- **CPU Core**
  - 32KB L1 D$
  - 32KB L1 I$
  - 256KB L2$
- **L3 fully inclusive of higher levels (but L2 not inclusive of L1)**
- **8MB Shared L3$
- **QuickPath System Interconnect**
  - Each direction is 20b@6.4Gb/s
  - Each DRAM Channel is 64/72b wide at up to 1.33Gb/s
- **Private L1/L2 per core**
- **3 DDR3 DRAM Memory Controllers**
- **Each DRAM Channel is 64/72b wide at up to 1.33Gb/s**
- **QuickPath System Interconnect**
  - Each direction is 20b@6.4Gb/s

**Cache Hierarchy Latencies**

- **L1 I & D 32KB 8-way, latency 4 cycles, 64B blocks**
  - Note: 4KB Page (12 bits) + 8-way associativity (3 bits) means cache index doesn’t need to use virtual part of address, so L1 cache access and TLB lookup can occur in parallel, L1 cache uses physical address tags
  - Address Tag 6-bit Cache Index/6-bit Block Offset
  - 20-bit Virtual Page Num 12-bit Page Offset
- **L2 256 KB 8-way, latency <12 cycles**
- **L3 8 MB, 16-way, latency 30-40 cycles**
- **DRAM, latency ~180-200 cycles**

**Optimization: Concurrent Access to TLB & Phys. Addressed Cache**

- Index I is available without consulting the TLB ⇒ cache and TLB accesses can begin simultaneously
- Tag comparison is made after both accesses are completed
- Cases: $I + O = k$, $I + O < k$, $I + O > k$

**Non-Uniform Memory Access (NUMA)**

- FSB architecture
  - All memory in one location
- Starting with Intel® Core™ microarchitecture (Nehalem)
  - Memory located in multiple places
- Latency to memory dependent on location
- Local memory has highest BW, lowest latency
- Remote Memory still very fast

**Core’s Private Memory System**

- 36 Entry Reservation Station
- Load queue 48 entries
- Store queue 32 entries
- Divided statically between 2 threads
- Up to 16 outstanding misses in flight per core

**All Sockets can Access all Data**

- Local Memory Access ~60ns
- Remote Memory Access ~100ns

- Such systems called "NUMA" for Non Uniform Memory Access: some addresses are slower than others
What to do with So Many Features?

- Turbo Mode exposed as additional Enhanced Intel SpeedStep® Technology operating point
  - Operating system treats as any other P-state, requesting Turbo Mode when it needs more performance
  - Performance benefit comes from higher operating frequency – no need to enable or tune software
- Turbo Mode is transparent to system
  - Frequency transitions handled completely in hardware
  - PCU keeps silicon within existing operating limits
  - Systems designed to same specs, with or without Turbo Mode

Managing Active Power

- Operating system changes frequency as needed to meet performance needs, minimize power
  - Enhanced Intel SpeedStep® Technology
  - Referred to as processor P-States
- PCU tunes voltage for given frequency, operating conditions, and silicon characteristics

Power Control Unit

Turbo Mode Enabling

- "Introduction to Performance Analysis on Nehalem Based Processors", 72 pages

"Software optimization based on performance analysis of large existing applications, in most cases, reduces to optimizing the code generation by the compiler and optimizing the memory access. Optimizing the code generation by the compiler requires inspection of the assembler of the time consuming parts of the application and verifying that the compiler generated a reasonable code stream. Optimizing the memory access is a complex issue involving the bandwidth and latency capabilities of the platform, hardware and software prefetching efficiencies and the virtual address layout of the heavily accessed variables."
Administrivia

- HKN surveys at end of lecture today.
- Extra OH
  - Mine: 12-2pm today in the Soda Alcoves
  - Come into the SD lab tomorrow wherever there's a blank sign-up time.
- Project 3 Face-to-Face grading tomorrow, 8/10 in 200 SD lab. Don't forget to show up!
- Final Exam - Thursday, 8/11, 9am - 12pm 2050 VLSB
    - Use the back side of your midterm cheat sheet!

Agenda

- Modern Microarchitecture: Intel Nehalem
- Administrivia
- RISC vs. CISC in Retrospect 30 years later
- Misc: Fixed Point, Polling vs. Interrupts (if time)

RISC vs. CISC

- Set up: From 1965 to 1980, virtually all computers implemented instruction sets using microcode (edited wikipedia entry): "Microcode is a layer of hardware-level instructions involved in the implementation of higher-level machine code instructions; it resides in a special high-speed memory and translates machine instructions into sequences of detailed circuit-level operations. It helps separate the machine instructions from the underlying electronics so that instructions can be designed and altered more freely. It also makes it feasible to build complex multi-step instructions while still reducing the complexity of the electronic circuitry compared to other methods. Writing microcode is often called microprogramming and the microcode in a particular processor implementation is sometimes called a microprogram."
- 1980s compilers rarely generated these complex instructions

RISC – CISC Wars

- Round 1: The Beginning of Reduced vs. Complex Instruction Set
  - Instruction set made up of simple or reduced instructions using easy-to-decode instruction formats and lots of registers was a better match to integrated circuits and compiler technology than the instruction sets of the 1970s that featured complex instructions and formats.
  - Counterexamples were the Digital VAX 11/780, the Intel iAPX 432, and the Intel 8086 architectures, which we labeled Complex Instruction Set Computers (CISC).
  - RISC advocates essentially argued that these simpler internal instructions should be exposed to the compiler rather than buried inside an interpreter within a chip.
  - RISC architects took advantage of the simpler instruction sets to first demonstrate pipelining and later superscalar execution in microprocessors, both of which had been limited to the supercomputer realm.

Original RISC Slides

- See slides 1 to 16 from RISCtalk1981v6.pdf
  - Unedited transparencies from 1981 RISC talk + RISC I, RISC II die photos
RISC – CISC Wars

• Round 1: The Beginning of Reduced vs. Complex Instruction Set
  – still amazing that was a time when graduate students could build a prototype chip that was actually faster than what Intel could build.
  – ARM, MIPS, and SPARC successfully demonstrated the benefits of RISC in the marketplace of the 1980s with rapidly increasing performance that kept pace with the rapid increase in transistors from Moore’s Law.

RISC – CISC Wars

• Round 2: Intel Responds and Dominates the PC Era
  • Intel “CISC tax”: longer pipelines, extra translation HW, and the microcode for complex operations but:
    1. Intel's fab line better than RISC companies, so smaller geometries hide some CISC Tax
    2. Moore's Law => on-chip integration of FPUs & caches, over time CISC Tax became smaller %
    3. Increasing popularity of IBM PC + distribution of SW in binary made x86 ISA valuable, no matter tax

RISC – CISC Wars

• Round 2: Intel Responds and Dominates in the PC Era
  • Most executed instructions simple
    – HW translated simple x86 instructions into internal RISC instructions, then use RISC ideas: pipelining, superscalar,...
  • Wikipedia: “While early RISC designs were significantly different than contemporary CISC designs, by 2000 the highest performing CPUs in the RISC line were almost indistinguishable from the highest performing CPUs in the CISC line.”

RISC – CISC Wars

• RISC vs. CISC in the PostPC Era
  • CISC not a good match to the smartphones and tablets of the PostPC era
    1. It’s a new software stack and software distribution is via the “App Store model” or the browser, which lessens the conventional obsession with binary compatibility.
    2. RISC designs are more energy efficient.
    3. RISC designs are smaller and thus cheaper.

RISC vs. CISC 2010 (mobile client)

<table>
<thead>
<tr>
<th>Design</th>
<th>Broadcom</th>
<th>ARM</th>
<th>MIPS</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU core</td>
<td>Hexacore</td>
<td>Quad-Core</td>
<td>Dual-Core</td>
<td>Dual-Core</td>
</tr>
<tr>
<td>CPU speed</td>
<td>1.3GHz</td>
<td>1.3GHz</td>
<td>1.2GHz</td>
<td>1.2GHz</td>
</tr>
<tr>
<td>Instruction set</td>
<td>RISC</td>
<td>RISC</td>
<td>RISC</td>
<td>CISC</td>
</tr>
<tr>
<td>Floating point</td>
<td>64-bit</td>
<td>64-bit</td>
<td>64-bit</td>
<td>64-bit</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>24GB/s</td>
<td>24GB/s</td>
<td>24GB/s</td>
<td>24GB/s</td>
</tr>
<tr>
<td>Memory capacity</td>
<td>3.8GB</td>
<td>10GB</td>
<td>10GB</td>
<td>10GB</td>
</tr>
<tr>
<td>Boot time</td>
<td>3 sec</td>
<td>2 sec</td>
<td>2 sec</td>
<td>2 sec</td>
</tr>
<tr>
<td>Display size</td>
<td>11.6&quot;</td>
<td>10.1&quot;</td>
<td>10.1&quot;</td>
<td>10.1&quot;</td>
</tr>
<tr>
<td>Special features</td>
<td>Faster boot, longer battery life, more memory</td>
<td>Faster boot, longer battery life, more memory</td>
<td>Faster boot, longer battery life, more memory</td>
<td>Faster boot, longer battery life, more memory</td>
</tr>
<tr>
<td>Clock rate</td>
<td>2.0GHz</td>
<td>2.0GHz</td>
<td>2.0GHz</td>
<td>2.0GHz</td>
</tr>
<tr>
<td>No. of Cores</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>CPU power</td>
<td>≤ 45W</td>
<td>≤ 45W</td>
<td>≤ 45W</td>
<td>≤ 45W</td>
</tr>
<tr>
<td>CPU weight</td>
<td>≤ 1kg</td>
<td>≤ 1kg</td>
<td>≤ 1kg</td>
<td>≤ 1kg</td>
</tr>
<tr>
<td>Price</td>
<td>$600</td>
<td>$300</td>
<td>$300</td>
<td>$300</td>
</tr>
</tbody>
</table>

Table 1. Comparison of Broadcom’s BCM98702H CPU with ARM, MIPS, and Intel CPUs. (*Preliminary silicon data rate, no overheads; benchmarks 1.5x faster, except Broadcom, Coremark a rough estimate)

RESOLVING RISC-CISC DEBATE

Products shipped?
2010: 6.1B ARM, 0.3B x86
How USA resolves debates?
We ask celebrities!
Who is the biggest celebrity in the world?

8/9/2010 Fall 2010 – Lecture #38
RESOLVING RISC-CISC Debate

Angelina Jolie as Kate Libby (aka as hacker Acid Burn) in movie “Hackers” (1995)

Angelina Jolie: “RISC architecture is gonna change everything.”

Hackers” (1995)

RESOLVING RISC-CISC Debate

Blue Man Group
“(silence)”

Agenda

Modern Microarchitecture: Intel Nehalem
Administrivia
RISC CISC in Retrospect 30 years later
Misc: Fixed Point, Polling vs. Interrupts

Fixed Point

Currently, we know to use floating point to represent real numbers.
– Lots of advantages - Covers wide range of numeric values, representations for +/- infinity, NaN.
– When might we be unable to use Floating Point?
– Embedded systems or microcontrollers (no floating point unit)
Note: Probably better to use floating point when it’s available!

Fixed Point

Idea: Use an integer to count by fractions of a value. Fractions determined by programmer (must be consistent).
Examples:
– int seconds = 2000; //counts by milliseconds
– int start = 65536; //counts by 2⁻¹⁶ths.
Addition/subtraction simple. Multiplication?
Less range than floating point, greater precision (at certain magnitudes).

Range of Numbers Represented

Fixed Point:

Floating Point:
Polling vs. Interrupts

- Need to communicate asynchronously between two components.
  - Asynchronous - No common clock for timing
  - Often: CPU communicating with Peripheral Device.
- Two communication options:
  - Polling: Repeatedly check to see if the device has sent something.
  - Interrupts: Have device trigger an interrupt when it’s ready.

Polling Advantages

- No overhead of Interrupt Service routine
- Steadier / more predictable performance.
- Easier to support/implement.

Interrupt Advantages

- Don’t waste time repeatedly checking device.
  - Could be less network bandwidth, less CPU utilization overall.

Polling vs. Interrupts

- We’ve seen examples of this so far:
  - Critical section implementation in MIPS - Polling.
    - Repeatedly check the lock to see if it’s free.
  - MapReduce master node checking on worker nodes - Polling.
  - Exceptions - Interrupt.
  - Other examples?

“And In Conclusion”

- Performance, Intel chip manufacturing => x86 ISA dominates Desktops/Servers
  - Speculative execution: branch prediction, out of order execution, data prefetching
  - Hardware translation and optimization of instruction sequences
  - Opportunistic acceleration (Turbo Mode)
- Cost, energy => RISC ISA dominates mobile personal devices, embedded computing, games
- What will the future hold for client+cloud?